Proceedings

17th International Conference
Scientific and Statistical Database Management

27-29 June 2005

———————————————————————————————

Donald Bren School of Environmental Science and Management
University of California
Santa Barbara CA, USA

Edited by
James Frew

In cooperation with
National Center for Ecological Analysis and Synthesis
Long Term Ecological Research Network

Title Page photo by California Coastal Records Project

Table of Contents

17th International Conference
Scientific and Statistical Database Management

Keynote Speakers
Michael Goodchild
David Maier

Session 8: Time / Object Queries

Session 9: Summarization

Session 10: Spatial Queries

Panel Discussion

Committees

Program Chair

James Frew, Donald Bren School of Environmental Science and Management

Program Committee

Amr El Abbadi, *University of California, Santa Barbara*
Chaitan Baru, *San Diego Super Computer Center*
Rajendra Bose, *University of Edinburgh*
Ewa Deelman, *Information Sciences Institute*
Max Egenhofer, *University of Maine*
Ian Foster, *University of Chicago*
Michael Franklin, *University of California, Berkeley*
James French, *The National Science Foundation*
Michael Gertz, *University of California, Davis*
Carole Goble, *University of Manchester*
Michael Goodchild, *University of California, Santa Barbara*
Robert Grossman, *University of Illinois, Chicago*
Oliver Guenther, *Humboldt University of Berlin*
Ralf Hartmut Gueting, *FernUniversitaet in Hagen*
Amarnath Gupta, *San Diego Super Computer Center*
Hans Hinterberger, *Eidgenoessische Technische Hochschule Zuerich*
Greg Janee, *University of California, Santa Barbara*
Christian Jensen, *Aalborg Universitet*
Graham Kemp, *Chalmers University of Technology, Sweden*
Daniel Keim, *University of Konstanz*
George Kollios, *Boston University*
Manolis Kourbarakis, *Technical University of Crete*
Miron Livny, *University of Wisconsin*
Leo Mark, *Georgia Institute of Technology*
Sally McClean, *University of Ulster*
William Michener, *Long Term Ecological Research Network*
Silvia Nittel, *University of Maine*
Beng Chin Ooi, *National University of Singapore*
Dimitris Papadias, *Hong Kong University of Science and Technology*
Norman Paton, *University of Manchester*
Jianwen Su, *University of California, Santa Barba*ra
Alex Szalay, *Johns Hopkins University*
Xiaoyang "Sean" Wang, *University of Vermont*
Jian Yang, *University van Tilburg*
Xiaofang Zhou, *University of Queensland*

Steering Committee

Arie Shoshani, *Lawrence Berkeley National Laboratory, USA (chair)*
Michalis Hatzopoulos, *University of Athens, Greece*
Silvia Nittel, *NCGIA, USA*
Jessie Kennedy, *Napier University, UK*
Menas Kafatos, *George Mason University, USA*

Session 1: Environmental Applications – 1

Retrofitting a Data Model to Existing Environmental Data

Bill Howe David Maier

Department of Computer Science
Portland State University
Portland, Oregon
{howe, maier}@cs.pdx.edu

Abstract

Environmental data repositories are frequently stored as a collection of packed binary files arranged in an intricate directory structure, rather than in a database. In previous work, we 1) show that environmental data is often logically equipped with a topological grid structure and 2) provide a data model and algebra of gridfields for manipulating such gridded datasets. In this paper, we show how to expose native data sources as gridfields without preprocessing, bulk-loading, or other prohibitively expensive operations. We describe native directory structures and file content using a simple schema language based on nested, variable-length arrays. This language is capable of describing general binary file formats as well as custom formats such as those used in the CORIE Environmental Observation and Forecasting System. We provide optimization techniques for extracting arrays by 1) analyzing file structure and 2) generating specialized code. Using extracted arrays, we assemble gridfields for more sophisticated manipulation and visualization. We show results using CORIE data. We find that generic access methods allow logical manipulation of physical data sources via the gridfield algebra without reformatting existing data.

1. Introduction

Integration of data within institutional and regional environmental systems is hindered, in part, by the heterogeneity of data formats. For example, the Northwest Association of Ocean Observing Systems (NANOOS) [1], chartered in response to a congressional initiative, aims to federate various institutional systems to provide a more comprehensive view of the coastal ocean in the Pacific northwest. The NANOOS charter acknowledges the significant number of ocean observing systems, but warns that these systems are not integrated in that they "do not share standards or protocols." In the interest of accelerating federation efforts in the environmental sciences, we have been studying the logical and physical structure of environmental data.

Environmental simulation and observation data are frequently defined over a topological grid structure. For example, a timeseries of sensor measurements might be defined over a 1-dimensional (1-D) grid, while the solution to a partial differential equation using a finite-element method might be defined over a 3-dimensional (3-D) grid. Datasets can be bound to a grid structure, producing what we term a *gridfield*. In previous work [6, 8], we develop a data model and associated query language for manipulating gridfields.

The salient feature of the gridfield model is that the grid structure of the datasets is explicit. Traditionally, data were stored and manipulated as arrays; the logical grid structure appeared only in the code itself. By reifying this hidden grid structure, we are better able to describe and implement a variety of manipulations using a small set of algebraic operators. Further, the data model helps separate logical and physical concerns, insulating software layers from changing physical representations.

However, in order to use gridfields to manipulate data from existing disparate sources, we must be able to read and interpret existing stored data; that is, we need appropriate access methods. Environmental datasets (indeed, most scientific datasets) are stored directly on a filesystem in packed binary files. Legacy applications can interpret these files, but new applications based on gridfields cannot.

One approach is to convert existing datasets to a special format already equipped with a gridfield interface. Indeed, database vendors frequently assume this approach: Before your data can be manipulated using the relational model, you must surrender control to the RDBMS via bulk-load operations. Unfortunately, the growth rate of collected scientific data is sufficiently large that sweeping conversion efforts are unlikely to succeed. Besides scalability issues, legacy analysis tools dependent on a particular format are common in scientific domains; mandatory rewrites of these tools would be unpopular.

Our initial solution was to hand-code custom access methods for each file format and directory structure we encountered. To generate a gridfield, routines to iterate over multiple files are layered on routines to interpret each file's format. The results are used to assemble gridfield objects suitable for manipulation with the gridfield algebra. Creation of these routines became repetitive enough to motivate a more general abstraction. We presented the vision for this approach in previous work [7]. In this paper, we describe languages and tools for accessing filesystem data with arbitrary structure without resorting to mass conversion. We do not discuss the output of gridfield expressions; results are generally piped into a visualization system for interactive analysis.

The context for our interest in grids is the CORIE Environmental Observation and Forecasting System being developed at the OGI School of Science & Engineering at Oregon Health & Science University [2]. The CORIE system is a multi-purpose platform for studying the fluid dynamics of the Columbia River estuary. Customers of CORIE's data products include commercial fisheries, environmental policy makers, and external research institutions. The CORIE repository consists of forecast and "hindcast" simulations covering 1998 to the present. Each day, forecast simulation runs add about 5GB to the data repository, while batches of hindcast runs, batches of calibration runs, and individual researchers' experiments are executed concurrently.

In a particular run of a simulation, 3-D spatial datasets are produced at regular intervals of simulated time, for each of several physical variables. These timestep datasets are distributed across several *checkpoint* files, each one usually covering a 24-hour period of simulated time. Checkpoint files have a custom binary format, and are arranged in a directory structure by week, by code version, and sometimes by purpose; e.g., calibration runs as opposed to final results. For example, the run directory names in the middle column of Figure 4(a) contain the week and the year, while the checkpoint files in the right-most column contain the day of the week. Every application accessing these data must understand the semantics of file and directory names, or interpret custom binary file formats, or both. The resulting situation is that much of the CORIE software is rather brittle with respect to changes in either directory structure or file format.

As we see with the CORIE system, logical datasets are not necessarily one-to-one with the files that house them. The physical organization of logical datasets is subject to operational constraints, and can sometimes be inconvenient for application writers. One dataset may span several files due to file size limits of the OS, for example. Portions of a dataset arriving at different times may be stored in separate files, as are the checkpoint files described above. Several datasets may be stored together in one file to simplify transfer over a network or to share metadata in the path. Second, files are arranged in a potentially intricate directory structure with embedded metadata.

The boundary between file name and file content is not inherent in the logical structure of the data, and can change depending on the situation. For example, the directory tree illustrated in Figure 4(a) has a separate file for each day of the week (per variable). In this case, the day of the week and the week number are not stored within the file, and are therefore inaccessible to tools that reason only about a file's content [13, 17]. This representation reflects the manner in which the data was generated: A checkpoint file was recorded for each day of the week to simplify recovery in case of failure. An individual researcher's ad hoc experiment might not require such caution; she might lump a week's worth of results into a single file without saving any checkpoints. In order to provide transparent access to either of these two representations, the model must allow uniform access to data stored in a file or data stored in the surrounding directory structure. Further, access methods should accommodate changes in physical organization without significant programming effort.

Since existing data comes in two forms – embedded in the directory structure and inside files – two physical access methods are required. However, adopting a single logical interface to both forms is desirable for conceptual economy.

Imagine we wish to visualize the average temperature near the water's surface for each week in 2004. The gridfield model allows us to perform aggregation and visualization, but first we must collect the appropriate data from the filesystem. Pseudo-code to gather the data is of the form

```
for each run in 2004:
  for the temperature variable:
    for each timestep:
      for each horizontal surface node:
        for the 1st two vertical depths:
          include the value in the result
```

The boundary between directory-level data and file content data is not apparent in the pseudo-code, nor should it be. We want the system to accept queries in terms of the logical structure, invoking the appropriate physical access method as necessary. To provide such functionality, the system must understand that a "run" corresponds to a directory, that "temperature" and other variables are each stored in a separate file, and that each of these files contain horizontal and vertical dimensions nested within a time dimension. Further, we need the ability to identify the runs for 2004, and the "first two" depths.

To communicate the physical structure of the data repository to the system, users write *schema files* in which they declare relevant *types*. Each type is associated with either 1) a regular expression identifying a set of files, or 2) an expression describing a block of binary data. With an appro-

priate schema file, we can express the pseudo-code above as follows:

```
run[year=2004].temp.times.horizs.depths[0:2]
```

The result is an array built by copying the values to a sequential block of memory. This array can then be used as part of a gridfield object for further processing. The code to traverse directories, iterate over files, and interpret a file's content efficiently is provided by the system.

The flexibility of accessing directory structure data uniformly with file content data can degrade performance. To maintain efficiency, a user can generate a specialized access program for a schema to improve performance. For binary files, programs can be further specialized by providing an exemplar (i.e., a representative member) of a set of structurally similar files. In this case, we can partially process the exemplar to generate a program tailored for answering queries over other members of the set. For example, although the structure of the checkpoint files can be highly variable, files for the same simulation run often have the same structure. By generating a program for one instance, we can efficiently access all related instances.

1.1. Contributions

Our contributions are the following:

- A data model for describing binary file formats.
- A complementary data model for describing data embedded in directory structures and file names. Together, we refer to these two mini-models as the *Native Data Model*.
- Access methods derived from the Native Data Model for extracting filesystem data.
- Optimization and code generation techniques to efficiently evaluate extraction queries over native content.
- Evidence of utility from the CORIE project.
- Experimental evidence that suitably optimized generic access methods can perform competitively with hand-coded access methods.

We will present our two-level data model in a top-down fashion. In Section 2, we review the salient features of the gridfield data model. In Section 3, we give examples of schema files for accessing binary file content as well as data encoded in the directory structure. In Sections 4 and 5, we focus on evaluation techniques and experimental results, respectively, for accessing binary content. We end by discussing related work, future work, and some conclusions.

2. Gridfield Data Model

A gridfield consists of a *grid*, and one or more *attributes*. A grid is a set of *cells*, partitioned by dimension. Cells of dimension k are called k-cells. A grid has dimension d if it contains no higher-dimensional cells. Cells are connected through an explicit or implicit *incidence relation*. For example, a triangle is a 2-cell to which three 0-cells (the vertices) and three 1-cells (the edges) are incident. Every non-empty grid must have at least one 0-cell. Each attribute is bound to the cells of exactly one dimension d, such that each d-cell maps to exactly one value of the attribute. With this model, we can have geometric attributes x and y bound to the vertices of a triangle, and an area attribute a bound to the triangle itself.

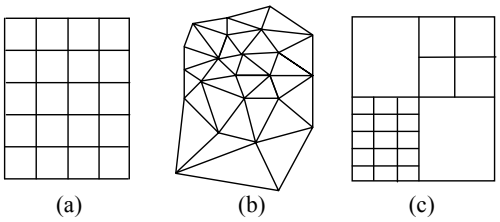
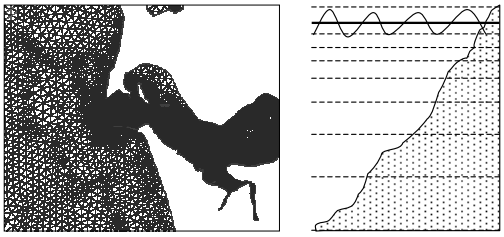


Figure 1. (a) A structured grid. (b) An unstructured grid. (c) A hierarchical grid.

The gridfield model provides an algebra with which to manipulate gridded datasets. Some operations in the algebra are reminiscent of relational operators but equipped to manage topology considerations. These operations include Restrict and Cross, which are similar to relational selection and cross product, respectively, but extended to maintain topological invariants [6]. Other operators are specific to gridfields. These include the Bind operator, which adds additional attributes to a gridfield, and the Aggregate operator, which maps cells of one grid onto another and aggregates the attribute values appropriately.

Grids are said to be *structured* or *unstructured*; our model treats both cases uniformly. The grid in Figure 1(a) is 2-dimensional structured and the grid in Figure 1(b) is a 2-dimensional unstructured grid consisting of triangles. Structured grids have implicit topology and can be modeled naturally by multidimensional arrays. Unstructured grids require explicit topology; the connections between cells must be included as part of the representation. Structured grids are easier to represent and admit very efficient algorithms. However, unstructured grids allow more precise modeling of a complex domain such as a coastline. To model multiresolution grids (Figure 1(c)) and other hierarchical structures, we provide *nested* grids, where the attribute values may themselves be gridfields [6].

The CORIE system uses a 2-dimensional unstructured grid to model the surface of the water around the mouth of the Columbia River Estuary (Figure 2(a)). This horizontal grid is repeated at each depth in a 1-dimensional structured grid, creating a 3-dimensional grid. The sloping bathymetry of the river causes many of the grid cells of this 3-dimensional grid to be positioned underground. Figure 2(b) illustrates the situation. Each dotted line represents a copy of the horizontal surface grid repeated at a particular depth and oriented perpendicularly to the plane of the paper. The shaded region represents the bottom of the river. The hori-



(a) (b)

Figure 2. (a) The horizontal unstructured CORIE grid. (b) Illustration of the river's bathymetry. The shaded region is underground.

zontal levels towards the bottom contain fewer valid "wet" cells than the levels near the surface. These invalid cells must be removed to correctly interpret CORIE datasets.

Given a horizontal grid H and a vertical grid V, the following expression generates the appropriate 3-dimensional gridfield for the CORIE system and associates a dataset $salt$ for further processing.

$$\mathsf{G} = \mathsf{Bind}(salt, 0, \mathsf{Restrict}(b < z, \mathsf{Cross}(H, V))) \quad (1)$$

The cross product of H and V (the Cross operator) represents the full 3-D domain of the Columbia River estuary and surrounding ocean. The Restrict operator cuts away the portion of the grid positioned underground (the shaded region in Figure 2(b)). The Bind operator reads in an array named $salt$ and attaches it as an attribute of the grid's 0-cells.

To use gridfields, programmers can construct them "manually" in their code, or they may write and reuse *gridfield declarations*. Examples of gridfield declarations appear in Figure 3. Each gridfield declaration is written as a function of lower-level types. A gridfield `H` can be derived from a value of type `sim_results` (Figure 4(b)). A gridfield `G` can be derived from a value of type `sim_results` and an integer representing a timestep.

Gridfield attributes and sequences of cells can both be represented as arrays. The italicized expressions in Figure 3 are *extraction queries* over the schema in Figure 8. These expressions will be described in Section 3.

The 0-cells of the grid are usually specified implicitly, using the keyword `implicit` followed by an integer expression. Cells of higher dimensions are defined as sequences of integer references to 0-cells. A triangle will have three references, and so on. Attributes are also specified using extraction queries. To bind the attribute x to cells of dimension 0, we use the syntax in line 4 of Figure 3.

Gridfields can also be declared through expressions in the gridfield algebra, as in line 9 of Figure 3. Given two gridfields `H` and `V` derived using the declarations on lines

```
1:  GridField H(sim_results r)
2:  H.grid.cells[0] = implicit r.nodes
3:  H.grid.cells[d] = r.cells
4:  H.x[0] = r.nodedata.x
5:  H.y[0] = r.nodedata.y

6:  GridField V(sim_results r)
7:  V.grid.cells[0] = implicit r.levels
8:  V.z[0] = r.zcor

9:  GridField G(sim_results r, int t) =
      Restrict( b<v, Cross( H(r), V(r) ) )
10: G.salt[0] =
      r.timedata[t].horizontal.depths.vector.val
```

Figure 3. Examples of gridfield assembly syntax.

1 and 5, we construct their *cross product* on line 9. The declaration on lines 9 and 10 specify the same gridfield as in Equation 1. The details of the gridfield operators can be found in a previous paper [6].

3. Native Data Model

In this section, we discuss the lower-level data models for accessing data encoded in directory structures and data encoded in binary files.

A filesystem-based data repository is described via a collection of declarations housed in a schema file. There are two types of declarations. *FileType* declarations describe relevant directory structures and allow access to data encoded within file and directory names. *BinaryBlockType* declarations describe the layout of portions of binary files.

3.1. Data From Directory Structures

Scientists frequently store and manage their data using direct filesystem interfaces, using filenames and directory structures equipped with metadata. For example, a dataset available from the National Climate Data Center (NCDC) website is stored in a file named `meso-eta_215_20030803_1800_fff` [14]. The string "20030803" is evidently a date, but the other fields require some external information to parse. A schema file can store this external information.

Consider the filesystem in Figure 4(a). The root directory `runs` contains directories with simulation results for each week of 2004 in the form `<week number>-2004`. Each week directory contains 14 files, one for each variable (salinity, temperature) day of the week (1-7).

To access the data embedded in these file names, we can write a *path pattern* in the style of the Unix scanf command.

```
[wk, yr, dy] = /runs/%i-%i/%i_salt.63
```

The left-hand side of this expression is a tuple of variable names. The right-hand side is a pattern matched against the set of all files in some filesystem context. Anonymous wildcards may also be used, as in line 8 in Figure 4(b). Each variable name corresponds to a sequence of values generated by evaluating the pattern in a particular filesystem con-

(a)

```
/run   /01-2004   /1_salt.63
                  /1_temp.63
                  /2_salt.63
                  :
       /02-2004   /1_salt.63
                  /1_temp.63
                  /2_salt.63
                  :
       /grids     /horiz.grd
                  /vert.grd
       /scripts   /do_run.pl
```

(b)

```
1: FileType weekly_run
2:   pattern[wk,yr] = /run/%i-%i/
3: FileType salt63
4:   pattern[day] = %i_salt.63
5: FileType temp63
6:   pattern[day] = %i_temp.63
7: FileType sim_results
8:   pattern[day] = %i_*.6?
```

Figure 4. (a) Simulation results stored on an ordinary filesystem. (b) FileType declarations describing the filesystem.

text. For example, the variable `wk` corresponds to the sequence $(01, 02)$ when the pattern is evaluated in the context of Figure 4(a). Note that the sequence order is determined by the manner in which the directory is traversed by the system calls for a particular OS.

To extract data from a filesystem, we write a path-like expression navigating through the FileTypes, where the rightmost identifier is a variable name.

```
weekly_run.salt63.day
```

This expression returns an "array" of all day values extracted from salt63 files in all weekly run directories. A natural extension to this basic form is to allow XPath-like conditions.

```
weekly_run[week=04].salt63[day<3].day
```

This expression restrict the results to a particular week and particular days. To reflect the array semantics, we can also allow array-style indexing expressions. An expression $\mathsf{name}[n:m]$ returns all elements $\mathsf{name}[i]$ for $n \leq i < m$. For example, the expression

```
weekly_run[0:2].salt63[1:3].day
```

selects the zeroth and first `weekly_runs`, and the first and second `salt63` files. The indexes refer to the ordering of the files as returned by the operating system, which does not necessarily agree with the ordering imposed by the values of the day variable itself.

Note that FileType declarations are not linked to each other; a schema does not prescribe a directory hierarchy. Different queries may express different sequences of FileTypes. Any, all, or none of these sequences may be valid with respect to a particular filesystem context. For example, an individual scientist may have several "loose" `salt63` files stored in his or her home directory. Queries can then reference those files directly, without having to first navigate through a run directory.

BinaryBlockTypes, described in the next section, do impose a particular structure for the content they describe. If a query attempts to navigate the binary data in an manner not supported by the schema, an error is raised.

3.2. Data From Binary Files

Scientists frequently use packed binary encodings of large datasets to conserve space and improve performance. In this section we describe a model of this data and its interface with the model of the directory structure. We model the content of binary files as a sequence of one or more *components*. Each component is either a *primitive* component with an associated name and typecode, or an *array* component, with a name, a length, and an element type. Our examples will use the primitive typecodes 'f', 'i', and '#c', representing floating point numbers, integers, and character arrays of fixed length '#'. The element type of an array is another sequence of components. In the tree resulting from these nested sequences, each leaf is a primitive component and each internal node except the root is an array component. Arrays are one-dimensional; multidimensionality is captured through nesting. Query results are 1-dimensional for portability across programming languages. Multidimensional results are collapsed to 1-dimensional arrays via concatenation, if necessary.

A file format for storing a simple ragged array is described in Figure 5(a), an instance of the file, in ASCII, is shown in Figure 5(b), and a hexadecimal representation is shown in Figure 5(c). The root component in Figure 5(a) is labelled as a `BinaryBlock` and given the name `ragged`. The top-level components, in order of their appearance in the file instance, are a 10-character string named `header`, an integer `n`, an array `r_sizes` and an array `outer`. Primitive components are written $name : type$. Array components are written $x * \{components\}$, where x is a *length expression* evaluating to an integer and $components$ is a sequence of components describing structure of the array's elements. The length expression of an array can be an integer literal, a reference to a primitive component appearing earlier in the file, or an arithmetic expression. In Figure 5(a), the length of the array `outer` is a reference to the component `n`. The element type of the array `outer` is another array component, `inner`, representing a second dimension.

Components referenced in a length expression may appear anywhere in the file prior to the reference. This freedom generalizes other binary description formats that re-

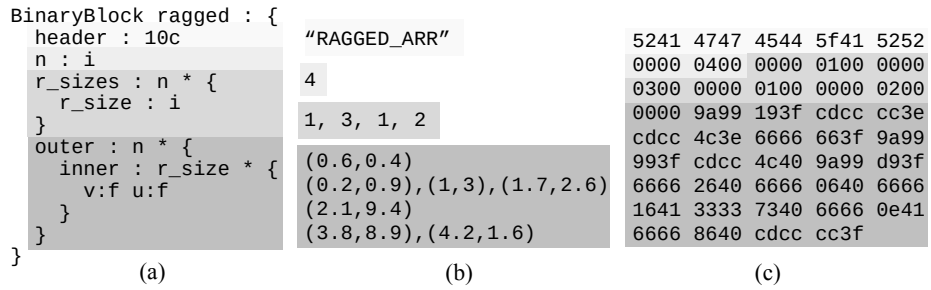


Figure 5. (a) A schema file for extracting binary file content. (b) An ASCII interpretation of a file instance. (c) A hexadecimal representation. Each color of shading represents a different logical component in the schema.

quire that the length of a variable-length array be defined immediately prior to the array's elements [13, 17]. Many formats, including netCDF [9], HDF [4], and CORIE's own internal format require this generalization.

Scanning the raw file instance in Figure 5(c) sequentially, we can interpret the data as in Figure 5(b). First, we encounter a ten-character header "RAGGED_ARR", then the integer 4, then four integers 1, 3, 1, 2, and finally a longer sequence of floating point numbers.

The length of the array `inner` is a reference to the integer component `r_size`, which itself is a sub-component of an array `r_sizes`. For each element of `outer`, a different size is specified by indexing into the array `r_sizes`. The portion of the instance in Figure 5(b) corresponding to the component `inner` consists of $1 + 3 + 1 + 2$ pairs of floating point values. Each pair has a value for the component `u` and the component `v`. Since the two arrays `outer` and `r_sizes` have the same expression for their array length (the expression '`n`'), there is no ambiguity as to which particular element of `r_sizes` is being referenced.

Programmers can access file data by writing path expressions constructed as a sequence of named components from the schema. At each nesting level, an array expression may be used to further restrict which values should be returned. An individual array element can be specified through conventional integer indexing. An array can also be "sliced" to produce another array as in APL or Matlab. All of the following expressions are valid extraction queries over the schema in Figure 5(a).

```
ragged.n
ragged.outer.inner[1]
ragged.outer.inner.u
ragged.outer[0:10].inner.u
ragged.outer.inner[0:20:2].v
```

4. Evaluating Queries Over Binary Data

In this section we describe the query evaluation engine. We wrote the prototype in Python extended with the numarray library for handling large numeric arrays efficiently. C code was emitted for the code generation experiments. After writing a schema file, users can parse and process it by calling Python functions. Figure 6(a) illustrates the internal representation of the schema file in Figure 5(a). Each oval corresponds to a component and each arrow corresponds to a pointer. Figure 6(b) illustrates an instance of the schema. The dashed ovals represent individual data values from a particular file. Query answers are a sequence formed from a subset of these values. For example, the query `outer[1].inner[0].u` returns the value u_{21} in Figure 6(b) and the query `outer.inner.v` returns all the values in Figure 6(b) with labels of the form v_{ij}.

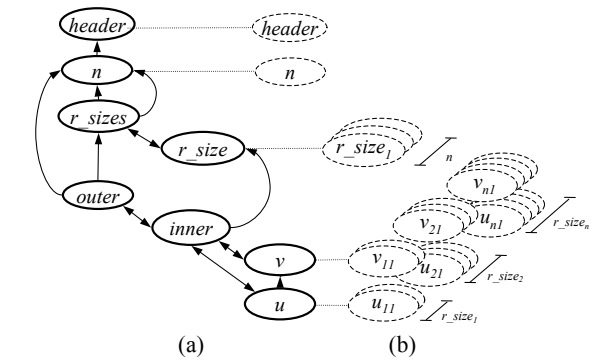


Figure 6. (a) Internal representation of a schema file. (b) An illustrated instance of a schema.

In order to read a datum from a file, we must know its position within the file and its size in bytes. The position of any datum is the sum of the sizes of the data that appear before it. The size of a primitive value is defined by its type (e.g., 4 bytes for floats and integers). The size of an array is its length multiplied by the size of its element type. The length of an array may depend on other data appearing earlier in the file. This situation can lead to significant complexity in file formats.

We now illustrate a naïve computation of size and position using the schema and instance of Figure 6. For a given

file, let $\mathsf{pos}(x)$ be the offset of the datum x, and $\mathsf{size}(x)$ be the total number of bytes occupied by the datum x. Suppose we wish to read the datum u_{xy} in Figure 6(b) in order to answer the query `outer[x].inner[y].u`. The required computation can be expressed as follows.

$$\begin{aligned}\mathsf{pos}(u_{xy}) &= \mathsf{pos}(v_{11}) \\ &\quad + \sum_{i=1}^{x-1} \sum_{j=1}^{r\_size_i} (\mathsf{size}(v_{ij}) + \mathsf{size}(u_{ij})) \\ &\quad + \sum_{j=1}^{y-1} (\mathsf{size}(v_{xj}) + \mathsf{size}(u_{xj})) \\ &= \mathsf{pos}(v_{11}) + \sum_{i=1}^{x-1} 8(r\_size_i) + 8(y-1)\end{aligned}$$

where

$$\begin{aligned}\mathsf{pos}(v_{11}) &= \mathsf{size}(header) + \mathsf{size}(x) \\ &\quad + \sum_{i=1}^{x} \mathsf{size}(r\_size_i) \\ &= 14 + 4x\end{aligned}$$

This computation can be simplified, as shown, by exploiting static information such as the size of primitive types and constant array sizes supplied in the schema. (The latter case does not appear in this example.) We refer to this simplification process as *static optimization*. Once the position is computed, we can invoke a system call to seek there and read the value.

To evaluate the query `outer.inner.u`, we could repeat the computation above for each value u_{ij} in Figure 6(b). To optimize this query, the goal is to minimize the number of read calls required. Since we are subject to a filesystem interface, we rely on the disk controller and the operating system to prefetch data block by block; we concentrate only on minimizing the number of read calls and ensuring that the calls are properly ordered with respect to the physical layout.

The unknown quantities in the computation above, the value of n and the values of the array r_sizes, prevent us from completing the computation statically. If we read these data first, we can derive the position and size of every other datum in the file, and read larger blocks of data at a time. We refer to the process of prereading certain data to further simplify the computations as *dynamic optimization*, since the process requires a conforming file instance.

Static and dynamic optimization allow us to partially evaluate the size and position computations required to evaluate queries. We offer two means of exploiting these partially evaluated computations. The first is by annotating the schema graph representation, in memory, with size and position information as it becomes available. The annotated schema graph can then be used to evaluate queries more efficiently. We liken this evaluation method to an *interpreter*.

(a)
```
BinaryBlock transformed : {
   header : 6c
   n : i
   sizes : n * {
      size : i
   }
   outer_inner : (2 * sum(size)) * {
      uv:f
   }
}
```

(b)

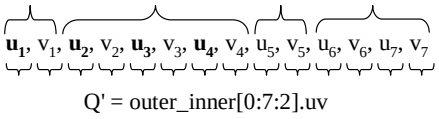


Figure 7. (a) A transformation of the schema in Figure 5(a). (b) A transformed query compatible with the transformed schema.

The second method is to generate programs able to evaluate queries efficiently. This method evaluates queries using a *compiled program*.

In order to perform dynamic evaluation, we must physically read values from a file instance. Using the interpretation method of evaluation, we can perform dynamic optimization on the fly as a query is received. However, we cannot generate, compile, and execute programs at run time in response to user queries without incurring significant cost. The reader might assume that dynamic optimization is therefore not available for use by the compiled program method of evaluation. However, we find in practice that many file instances share the same structure, even if they are not guaranteed to do so. Therefore, we can use a typical file instance, called a *file exemplar*, to perform dynamic optimization and generate programs specialized for answering queries over file that share structure with the file exemplar.

The programmer is responsible for supplying a file exemplar and specifying the components that will be read and used for specialization. For example, consider a set of files S conforming to the schema of Figure 5(a). Consider further that a substantial subset of these files $R \subset S$ agree on their values for n and r_sizes. The programmer can provide a file $r \in R$ as an exemplar, and specify the components n and r_sizes, and the system will generate an optimized program able to efficiently answer queries over all the members of R.

4.1. Schema Transformation

By reading certain components early, we can partially evaluate the computations required to answer queries. Another approach to optimization we have explored is to transform and simplify a schema in response to a particular query. By finding and exploiting a simplified schema, we reduce the cost of evaluation.

Consider the query `outer[0:2].inner.u` over the schema of Figure 5(a). In Figure 7(a), we show a transformed version of the schema. We have blended the two components `u` and `v` into a single component named `uv`. Further, the nested, ragged array has been flattened into a one longer array. The new length is the sum of all the `size` values times two (since there are two floats, `u` and `v`).

In Figure 7(b), we have a sequence of values representing the contents of the schema instance in Figure 5(b). The query Q is the original, and Q′ is the result of transformation. The values to be returned by both queries are the same; they appear in bold. The brackets represent the grouping structure specified by the original schema (the top brackets) and the transformed schema (the bottom brackets). The advantage of this type of transformation is that we can read a large block of data and then "slice" the resulting array to select the appropriate values. In the original structure, we were forced to loop through the internal representation of the schema, looking up the sizes of the inner dimension. The transformed schema results in fewer iterations. In Section 5, we show that this type of transformation can result in significant performance improvement for our Python-based implementation.

Although we identify and exploit simple instances of this transformation, it remains future work to describe the general form of these simplifications for arbitrarily nested variable-length arrays.

5. Experimental Results

Figure 8 shows the schema for the simulation result files on which we ran our experiments. Logically, these files house a timeseries of three-dimensional datasets bound to a grid. For each simulation run, one of these files is produced for each of 7 to 10 variables, for each day of simulated time. The sizes of these files range from 35MB for two-dimensional variables such as surface elevation or wind pressure at the surface, to 655MB for horizontal velocity vectors for each three-dimensional point. The entire data repository holds around 5000 simulation runs, with additional runs executed daily.

The header portion of each file, shown in plain type, contains time-independent information, including the horizontal and vertical grids and the river's bathymetry. The time-varying portion of the file, highlighted in bold type, is a nested array component with four layers of nesting over the actual values of the simulated variables: `timedata`, `horizontal`, `depths`, and `vector`. The array `surf`, similar to the array `bathymetry`, contains indices into the vertical grid. This index represents the surface of the water: the highest level that still holds valid data. However, the water's surface changes over time, so we need a surface array for each timestep.

We compare the performance of four representative queries using seven different techniques. The queries are listed in Table 1 and the results are presented in Table 2. Query 1 extracts the first complete timestep of data from the file, copying it to an array in memory. Query 2 accesses all timesteps, but extracts only a relatively small portion of the horizontal data; 2999 nodes. Query 3 extracts the first two depth levels for every horizontal node, for every timestep. Since we do not know how many depth levels exist at each node until we read the file's header, we cannot derive the result size statically; some nodes may have as few as one depth level. Query 3 returned the largest result size in practice, at 5.6 million floating point numbers. This query also proved most challenging for our software. Query 4 extracts the surface information for timesteps 30 through 44. This query does not directly involve any variable-length arrays and is therefore easier to compute for most techniques. This query also had the smallest result size.

```
magic : 48c, version_string : 48c
start_time : 48c, variable_nm : 48c
variable_dim : 48c, nsteps : i, timestep : f
skip : i, rank : i, idim : i,
vpos : f, zmsl : f, levels : i
zcor : levels * { z : f }
nodes : i
elements : i
nodedata : nodes * {
  x : f,  y : f,  h : f,  bathymetry : i
}
cells : elements * { nodeids : 3 * { id :i } }
timedata : nsteps * {
  tstamp : f, tstep : i
  surfdata : nodes * { surf : i  }
  horizontal : nodes * {
    depths : (levels - bathymetry + 1) * {
      vector : rank * { val : f }
    }
  }
}
```

Figure 8. A BinaryBlock describing the CORIE simulation output.

The experiments were conducted on a platform almost identical to the nodes of the cluster used to run the CORIE simulations. The machine runs Linux on a 2.4 GHz processor with 4GB of main memory. We report the average of eight experiments run on two different days with the machine unencumbered. The variance was less than 1% in all cases, demonstrating stability of the results.

The simplest technique we tested was to use the internal representation produced by the static optimizer directly (Experiment (a) in Table 2). While this basic tool worked well on small files during testing and development, it struggled on the large files of the CORIE system. After 500 seconds, we stopped the experiments. The use of the Dynamic Optimizer leads to improved performance (Experiment (b)). The Dynamic Optimizer prefetches information from the header during query evaluation, and is therefore able to sim-

label	query	result size
Q1	timesteps[0:1].H.V.vector.v	3.3MB
Q2	timesteps.H[2000:5000].V.vector.v	16.3MB
Q3	timesteps.H.V[0:2].vector.v	22.7MB
Q4	timesteps[30:45].surfdata.surf	1.77MB

Table 1. Tested queries with result sizes.

plify or eliminate many computations.

Experiment (c) in Table 2 uses a small class of schema and query transformations (see Section 4). These transformations act as shortcuts, allowing us to read large blocks of data from the file and then "slice" them to extract the appropriate values. For example, Q1 returns an entire timestep. The system can detect that there is no need to iterate over each horizontal node and each vertical level; it can simply compute the size s of a timestep from the header information, seek to the beginning of the relevant section, and read all s bytes. The result is dramatic improvement in performance. The effect is especially pronounced due to our choice of technology. The Python language can perform well when expensive routines are pushed down into the compiled C code underpinning the language. Reading and slicing arrays are examples of these fast operations.

Note that for Q3, we are unable to identify an appropriate transformation. Since the vertical component has a variable length, there is no simple pattern we can use to extract the data values we want. We must access much more data to navigate through the file. The hand-coded reader (experiment (g) in Table 2) is able to evaluate this query efficiently by precomputing cumulative sizes of the variable-length arrays and striding through them as necessary. It remains future work to incorporate the general form of this technique.

Experiment (d) executes identical code to Experiment (c). For this case, we subtract the time spent prefetching and optimizing. The rationale is that for large classes of similar files, this work may be done once rather than for each file.

Experiments (e) and (f) use generated C programs to compute the results. The first is a program generated directly from the static optimizer's output. We performed these experiments to see whether a compiled language would outperform the Python routines even without significant optimization. The results show that traversing these large files without guidance is prohibitively expensive even for a compiled C program. The specialized generated program is created from the results of the dynamic optimizer and is specialized for a particular class of file instances. The rationale is that many of these specialized readers could be generated and stored by the system. When planning evaluation of a query, the specialized programs could be considered as fast alternative access methods. Unfortunately, these generated programs do not perform as well as expected. The reason is

Experiment	Q1	Q2	Q3	Q4
(a) static	>500	>500	>500	>500
(b) dynamic	28.1	138.	284.	8.64
(c) dynamic + transf.	0.83	1.	86.	0.85
(d) spec. + transf.	0.02	0.24	85.	0.26
(e) generated, gen.	65.	104.	>500	3.2
(f) generated, spec.	4.7	22.	32.	2.6
(g) by hand	0.02	0.5	0.6	0.02

Table 2. Response time in seconds by query.

that the transformation optimizations that gave good performance in Experiments (c) and (d) are not incorporated in the generated code. In ongoing work we are improving the quality and performance of the generated programs. Note that the specialized generated program exhibits stability; it was able to evaluate Q3 without incurring the same magnitude of penalty as the other approaches.

6. Related Work

Scientific applications today in some ways resemble business applications circa 1977. Copious amounts of data are stored in files with intricate formats. Skepticism regarding database technology is prolific. Legacy systems are built from efficient but brittle software components. To mitigate the perceived (and real) risk of adopting unproven database systems, early data models were implemented as file transformation engines.

The EXPRESS system [15] provided two languages: one for describing a file's structure, and another for transforming that structure. Transformations were used as a query facility, but also as a bulk-load facility to translate legacy data into a new format. Our approach is similar, though we distinguish two data models: one for source data (directory structures and file content) and another for target data (gridfields). We have not yet considered materializing gridfields assembled using schema files. That is, we do not permanently transform source data into gridfields, but rather retrofit a gridfield interface onto in situ data.

Batory gave a taxonomy of record-oriented file structures used by commercial databases in terms of fields and pointers [3]. Our work similarly provides a description of file structures in terms of arrays.

The Binary Format Description Language (BFD) [13] is an XML dialect that describes binary formats and allows transformation of binary data to XML data. While this tool has a niche, our interest is to support efficient and flexible access to binary data – converting binary data to XML is clearly impractical for large datasets. The BinX [17] library is also related to this proposal. Binary data file formats are described using instances of a specialized XML Schema. An API allows access to the data and automatic reformatting according to the local machine's byte order and bit or-

der. The most recent version added support for nested arrays, but only if their length is fixed.

The External Data Representation standard (XDR) [16] is a data format language focused on machine-level number representation issues. Variable-length arrays in XDR must have homogeneous elements (i.e., their elements are not be variable-length), and their lengths must be encoded directly prior to the first element. Further, XDR obviously does not describe directory structures, preventing access to datasets that span multiple files.

Platforms for scientific query and analysis include AQSIM [11] and the Active Data Repository [10]. Both of these systems require preprocessing of data repositories in order to construct indices, compute statistics, or to bulk load data into a managed environment. Other systems such as Chimera [5] and Godiva [12] supervise the execution of data access code, but rely on users to write them in the first place. We operate in a different space of requirements: We propose convenient and immediate access to data that the user does not control.

7. Future Work and Conclusions

We plan to include constraints as part of the schema language for binary block types. Constraints could be used to validate files before processing. For example, many file formats use a few bytes at the beginning of the file as a "magic" code indicating the format type and version. Using constraints, we could validate that these magic codes are sensible before continuing processing. Currently, nonconforming file instances will cause undefined behavior.

We also plan to add support for value-based predicates on arrays, in order to push some gridfield processing into the native data model subsystem. We are also working on "output schemas" that allow files to be restructured declaratively. Given 1) an input schema and 2) an output schema with a subset of the input's declarations, we aim to allow instances of the input schema to be automatically and efficiently transformed into instances of the output schema.

In conclusion, we advocate in situ processing of large scientific data repositories. Converting Terabytes of data to support new data models is infeasible, and continuously writing access methods for changing formats is time consuming. Our results show that although file formats with high variability can be expensive to process without programmer guidance, hand-coded access methods can be replaced with generic or generated access methods. We recognize that a logical dataset can often span multiple files in practice, and that the directory structure and filename can encode part of the dataset's structure. Our techniques can facilitate data sharing between research groups and institutions with heterogeneous data formats.

Acknowledgements

We would like to thank the CORIE science team for their input and support. This work was was supported by NSF ITR Award No. ACI-0121475.

References

[1] Northwest Association of Networked Ocean Observing Systems. http://www.nanoos.org.

[2] A. Baptista, M. Wilkin, P. Pearson, P. Turner, M. C., and P. Barrett. Coastal and estuarine forecast systems: A multipurpose infrastructure for the columbia river. *Earth System Monitor, NOAA*, 9(3), 1999.

[3] D. S. Batory. Modeling the storage architectures of commercial database systems. *ACM Trans. Database Syst.*, 10(4):463–528, 1985.

[4] N. C. for Supercomputing Applications (NCSA). HDF5: API specification reference manual. http://hdf.ncsa.uiuc.edu/, 2004.

[5] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *14th Conference on Scientific and Statistical Database Management*, 2002.

[6] B. Howe and D. Maier. Algebraic manipulation of scientific datasets. In *Proceedings of the 30th International Conference on Very Large Databases (VLDB2004)*, 2004.

[7] B. Howe and D. Maier. Logical and physical data indpendence for native scientific data repositories. *IEEE Data Engineering Bulletin*, 27(4):30–37, 2004.

[8] B. Howe, D. Maier, and A. Baptista. A language for spatial data manipulation. *Journal of Environmental Informatics*, 2(2), December 2003.

[9] H. L. Jenter and R. P. Signell. NetCDF: A public-domain-software solution to data-access problems for numerical modelers. Unidata, 1992.

[10] T. Kurc, U. atalyurek, C. Chang, A. Sussman, and J. Salz. Exploration and visualization of very large datasets with the active data repository. Technical report, Technical Report CS-TR4208, University of Maryland, 2001.

[11] B. Lee, T. Critchlow, G. Abdulla, C. Baldwin, R. Kamimura, R. Musick, R. Snapp, and N. Tang. The framework for approximate queries on simulation data. *Inf. Sci. Inf. Comput. Sci.*, 157(1-2):3–20, 2003.

[12] X. Ma, M. Winslett, J. Norris, X. Jiao, and R. Fiedler. Godiva: Lightweight data management for scientific visualization applications. In *ICDE*, pages 732–744, 2004.

[13] J. Myers and A. Chappell. Binary format description language. Technical report, Pacific Northwest National Laboratory, 2003.

[14] National Climatic Data Center. NCEP AWIPS Eta Model Data. http://nomads.ncdc.noaa.gov:9090/dods/NCDC_NOAAPort_ETA.

[15] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. EXPRESS: A data EXtraction, Processing, amd RE-Structuring System. *ACM Transactions on Database Systems*, 2(2):134–174, 1977.

[16] R. Srinivasan. XDR: External data representation standard, RFC 1832. Technical report, Sun Microsystems, 1995.

[17] M. Westhead and M. Bull. Representing scientific data on the Grid with BinX - binary XML description language. Technical report, EPCC, University of Edinburgh, 2003.

An Information Theoretic Model for Database Alignment

Patrick Pantel, Andrew Philpot and Eduard Hovy
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
{pantel,philpot,hovy}@isi.edu

Abstract

As with many large organizations, the Government's data is split in many different ways and is collected at different times by different people. The resulting massive data heterogeneity means that government staff cannot effectively locate, share, or compare data across sources, let alone achieve computational data interoperability. The premise of our research is that it is possible to significantly reduce the amount of manual labor required in database wrapping and integration by automatically learning mappings in the data. In this research, we applied statistical algorithms to discover column correspondences across environmental databases. We have seen particular success in an information theoretic model, which we call SIfT, which performs data-driven column alignments. We have applied SIfT to mapping Santa Barbara and Ventura County Air Pollution Control Districts' 2001 and 2002 emissions inventory databases with the California Air Resources Board statewide inventory database. The application of SIfT yielded 75% precision and 72.2% recall on the column alignment task. On a task of integrating new district data with the statewide database, we achieved 55% accuracy for Ventura County and 59% accuracy for Santa Barbara County.

1. Introduction

Due to the wide range of geographic scales and complex tasks that the Government must administer, its data is split in many different ways and is collected at different times by different agencies. The resulting massive data heterogeneity means one cannot effectively locate, share, or compare data across sources, let alone achieve computational data interoperability. A case in point is the California Air Resources Board (CARB), which is faced with the challenge of integrating the emissions inventory databases belonging to California's 35 air quality management districts to create a state inventory. This inventory must be submitted annually to the US EPA which, in turn, must perform quality assurance tests on these inventories and integrate them into a national emissions inventory for use in tracking the effects of national air quality policies.

To date, most approaches to wrap data collections, or even to create mappings across comparable datasets, require manual effort. Despite some promising recent work, the automated creation of such mappings is still in its infancy, since equivalences and differences manifest themselves at all levels, from individual data values through metadata to the explanatory text surrounding the data collection as a whole.

Viewing the data mapping problem as a variant of the cross-language mapping problem in Machine Translation, we employed statistical text alignment and clustering algorithms developed in Natural Language Processing to discover correspondences across comparable datasets. In this paper, we present an information theoretic model, which we call *SIfT* (Significance Information for Translation), that performs data-driven column alignment. The key to our approach is to identify the most informative data elements and then match data sources that share these informative elements. For example, we expect that the word "*the*" will be present in many different columns. However, consider some word, like "*carbon*", which occurs in very few columns. A random pair of columns from two data sources that both contain the data element "*the*" are intuitively not as similar as if both columns contained the data element "*carbon*". Our model automatically detects that "*the*" is less informative than "*carbon*" and will consequently assign a higher similarity to two columns that share only "*carbon*" rather than only "*the*".

This work has the potential to significantly reduce the amount of human work involved in creating single-point access to multiple heterogeneous databases.

The remainder of this paper is organized as follows. In the next section, we review related work in database alignment. Section 3 describes the environmental databases that we use as a testbed for alignment and in Section 4 we present our information-theoretic model for alignment. Our experimental results are presented in Section 4.4. Finally we conclude with a discussion and future work.

2. Related work

A lack of standardization has made it very difficult to integrate various data sources. Integration and reconciliation of data across non-homogeneous databases is an old but unsolved and ever-growing problem. Some mechanism is required to standardize data types, reconcile slightly different views, and enable sharing.

For textual data, the information retrieval approach exemplified in web search engines such as Google and Yahoo! works reasonably well to find exact and close matches (around 40% precision & recall over the past decade, determined at the annual TREC[1] conferences).

For conventional databases, however, search engines are inappropriate. Instead, two approaches are possible. Either one can build a central data model that integrates the specialized metadata for each database, or one can create direct mappings across the data (cells, columns, rows, etc.) of the databases themselves. Both approaches are difficult. With regard to the former, various methods have been developed. The "global-as" view method [2][3] assumes that the central model is complete, but that local databases may deviate from it; access is via the central model. This model requires serious effort to extend. In contrast the "local-as" view method [8] assumes that the central model is incomplete, simply narrowing the sources to be further searched, which may require tedious additional search effort. In contrast, the "ontology method" uses a single overarching super-metadata model (the ontology) into which all databases' metadata descriptions are subordinated hierarchically [1][6].

The second general approach, creating mappings across individual (subsets of) data, is impossible to bring about for real-sized data collections unless (semi-) automated methods are used to find the mappings. Schema-based matching algorithms [13] align databases by matching the meta-data available in the databases (e.g., two tables with column name *zip_code* are aligned; most approaches will also match columns labeled *zip_code* and *zip*). However, since there is often no standardized naming scheme for meta-data, schema-based methods often fail. Instance-based matching algorithms align databases using the actual data [5]. Such data driven methods typically fail when different columns share a common domain (e.g., business vs. residence phone numbers) or when matching columns that exhibit different encodings (e.g., a phone number field stored as a text string in one database and stored as a numerical field in another). Kang and Naughton [7], whose work most resembles ours, propose an information-theoretic model to match unaligned columns after schema- and instance-based matching fails. Given two columns *A.x* and *B.x* that are aligned, the model computes the association strength between column *A.x* with each other column in *A* and column *B.x* and each other column in *B*. The assumption is that the highly associated columns from *A* and *B* are the best candidates for alignment. In this paper, we adopt a similar information-theoretic model, but for instance-based matching. Instead of matching highly associated columns, which requires seed alignments, we find the data elements that are most highly associated to each column and then match columns that share these important data elements

[1] The Text REtrieval Conference (TREC) provides the infrastructure necessary for large-scale evaluation of text within the information retrieval community.

3. Environmental databases

We are working with the following set of domain data. Emissions inventories are being provided by staff at the California Air Resources Board (CARB) in Sacramento, who annually integrate the emissions inventory databases belonging to California's 35 Air Quality Management Districts (AQMD) to create a state inventory. This inventory must be submitted annually to the US EPA which, in turn, must perform quality assurance tests on these inventories and integrate them into a national emissions inventory for use in tracking the effects of national air quality policies.

To deliver their annual emissions data submittal to CARB, air districts have to manually reformat their data according to the specifications of CARB's emission inventory database called California Emission Inventory Development and Reporting System (CEIDARS). Every time the CEIDARS data dictionary is revised (as has happened several times recently, for example in 2002), work is required on the part of AQMD staff to translate emissions data into the new format. Likewise, when CARB provides emissions data to US EPA's National Emission Inventory (NEI), significant effort is required by CARB staff to translate data into the required format.

Our testbed for this research consists of the 2001 and 2002 Santa Barbara County Air Pollution Control District (SBCAPCD) and Ventura County Air Pollution Control District (VCAPCD) emissions inventories, two of the 35 California air districts.

4. Data-driven alignment

The key to our approach is to first identify, using an information-theoretic model, the most informative data elements and then match data sources that share these informative elements. For example, in our case study of matching SBCAPCD and CARB schemas, since the source data is from Santa Barbara County, we expect that many of the columns in SBCAPCD will contain the word "Santa Barbara" (e.g., factory names, locations, addresses, etc.) However, only one column contains the word "Wingerden." Therefore, a random pair of columns from SBCAPCD and CARB that both contain the data element "Santa Barbara" are intuitively not as similar as if both columns contained the data element "Wingerden." Our model automatically detects that "Santa Barbara" is less informative than "Wingerden" and will consequently assign a higher similarity to two columns that share only "Wingerden" rather than only "Santa Barbara."

4.1. Information theoretic model

Informative elements are measured in *SIfT* using an information theoretic model called mutual information. Similar columns are discovered using a clustering algorithm called CBC [9].

In any clustering application, the critical step is representing the data such that elements group together according to the desired output. For example, if we want to cluster medical patients according to their possible diseases, we might represent them by their height, weight, age, gender, whether they smoke or not, etc.; we would not, however, represent them by their favorite board game or favorite movie since with this representation we would likely group the patients according to their entertainment preferences.

The representation of an element is often called a feature vector (or vector space model). Each feature is simply a measurement of the element. For example, in clustering data points on a 3-dimentional graph, we would represent each point using three features: the *x*, *y*, and *z* coordinates. These three measurements completely describe the points.

4.1.1. Feature representation

In aligning inter-database columns *s* and *t*, we assume that *s* and *t* contain similar but not necessarily identical fields (accounting for noise and discrepancies in the data). One representation for columns is simply the data fields they contain. Consider the following database columns taken from two databases *S* and *A*:

```
S.phone.number:
   310-555-6789, 310-555-0987,
   780-433-9393, …
A.area:
   310, 310, 780, …
A.ph:
   555-6789, 555-0987, 433-9393, …
```

We could represent these columns using their field values with a frequency of occurrence as measurement. For the above example, the feature vectors using this representation would be:

```
S.phone.number:
   310-555-6789    1
   310-555-0987    1
   780-433-9393    1
A.area:
   310             2
   780             1
A.ph:
   555-6789        1
   555-0987        1
   433-9393        1
```

Notice that none of these features overlap and consequently a clustering algorithm would not discover any similarity between the columns. In this research, we enrich the feature space by classifying data columns within several feature domains (e.g., zip code, phone number, state, positive integer, …) Once a column is classified within a particular feature domain, the feature types associated with that domain are extracted for the column's feature vector (e.g., *zip5* – the first five digits of a zip code, *zip4* – the last four digits of a nine-digit zip code, *area* – the area code of a phone number, *exch* – the 3-digit phone number exchange, *phone* – the seven-digit local phone number, *ext* – the extension of any digits after a 10-digit phone number). We also add domain specific feature domains. We implemented a total of 20 feature domains.

The algorithm we use for recognizing these domains simply searches for patterns that describe the domain. For example, a 10-digit phone number is recognized if the first three digits are a known area code, the fourth digit is between [2-9], and the rest of the field is numeric. If our patterns do not fire on a particular column (e.g., a column containing international phone numbers), then the catch-all Text feature domain will always fire.

We allow the user of the system to decide which feature domains and associated feature types are active for any given alignment. Suppose a column is identified as a phone number and we decide to extract feature types *area* and *phone* for all phone numbers. Then for each field such as "310-555-6789", the system extracts two features with frequency 1:

```
area:310          1
phone:555-6789    1
```

Similarly, for fields such as "555-6789", we extract a single feature:

```
phone:555-6789      1
```

Now, we see some overlap between the columns *S.phone.number* and *A.ph* from the previous section. A clustering algorithm could therefore discover a similarity between the two columns.

4.1.2. Mutual-information vector-space model

Representing data for clustering requires both a feature representation and a measurement of the features. We now describe our model for measuring the feature types described in the previous section.

Above, we measured each feature by its frequency of occurrence. However, certain features are more informative than others. For example, the common word '*the*' will be present in many text strings. Two strings that happen to contain the word '*the*' does not indicate as much similarity as if they contained an uncommon word such as '*carbon*'.

Pointwise mutual information is commonly used to measure the association strength between two events [4]. It essentially measures the amount of information one event gives about another. For example, knowing that a column contains the word '*the*' is not informative of the contents of that column (because *the* is common across many columns). Conversely, if very few columns contain the word *carbon*, then that word is an informative feature (i.e. if columns *p* and *q* from different databases happen to contain *carbon*, then they are more likely to be aligned than if they shared the word *the*).

The pointwise mutual information between two events *x* and *y* is given by:

$$mi(x,y) = \log \frac{P(x,y)}{P(x)P(y)}$$

Mutual information is high when *x* and *y* occur together more often than by chance. Mutual information compares two models (using KL-divergence) for predicting the co-occurrence of *x* and *y*: one is the MLE (maximum-likelihood estimation) of the joint probability of *x* and *y* and the other is some baseline model. In the above formula, the baseline model assumes that *x* and *y* are independent. Note that in information theory, mutual information refers to the mutual information between two random variables rather than between two events as used in this paper. The mutual information between two random variables *X* and *Y* is given by:

$$MI(X,Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$$

The mutual information between two random variables is the weighted average (expectation) of the pointwise mutual information between all possible combinations of events of the two variables.

For each element (column) *e*, we first construct a frequency count vector $C(e) = (c_{e1}, c_{e2}, \ldots, c_{em})$, where *m* is the total number of features and c_{ef} is the frequency count of feature *f* occurring in element *e*. Here, c_{ef} is the number of times column *e* contained a feature *f*. For example, in column *e* = *A.area* from Section 4.1.1, one feature is *area:310* with count 2.

We then construct a mutual information vector $MI(e) = (mi_{e1}, mi_{e2}, \ldots, mi_{em})$ for each column *e*, where mi_{ef} is the pointwise mutual information between column *e* and feature *f*, which is defined as:

$$mi_{ef} = \log \frac{\frac{c_{ef}}{N}}{\frac{\sum_{i=1}^{n} c_{if}}{N} \times \frac{\sum_{j=1}^{m} c_{ej}}{N}}$$

where *n* is the number of columns and $N = \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij}$ is the total frequency count of all features of all columns.

A well-known problem is that mutual information is biased towards infrequent elements/features. We therefore multiply mi_{ef} with the following discounting factor [9]:

$$\frac{c_{ef}}{c_{ef}+1} \times \frac{\min\left(\sum_{i=1}^{n} c_{ei}, \sum_{j=1}^{m} c_{jf}\right)}{\min\left(\sum_{i=1}^{n} c_{ei}, \sum_{j=1}^{m} c_{jf}\right)+1}$$

4.2. Similarity metric

To cluster elements, we need a measure of similarity (or distance) between them. We construct a matrix containing the similarity between each pair of columns e_i and e_j using the cosine coefficient of their mutual information vectors [11]:

$$sim(e_i, e_j) = \frac{\sum_f mi_{e_i f} \times mi_{e_j f}}{\sqrt{\sum_f mi_{e_i f}^{\,2} \times \sum_f mi_{e_j f}^{\,2}}}$$

This measures the cosine of the angle between two mutual information vectors. A similarity of 0 indicates orthogonal vectors whereas a similarity of 1 indicates identical vectors. For two very similar elements, their vectors will be very close and the cosine of their angle will approach 1. A nice property of the cosine metric is that it is not very sensitive to 0-valued features. Hence, given a column containing all California EPA facilities and another containing only Santa Barbara facilities, cosine will find a similarity even though all non-Santa-Barbara facilities will have frequency 0 in

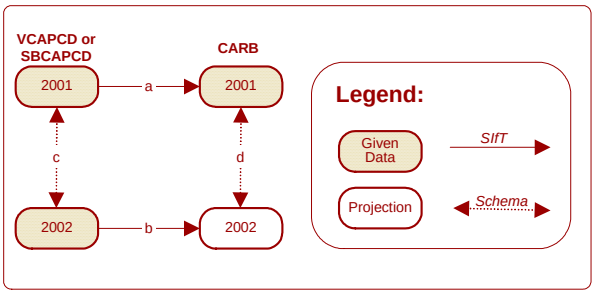


Figure 1. Experimental design for automatically generating a data transfer between VCAPCD/SBCAPCD and CARB for 2002 given historical 2001 data.

the second column. In other words, the absence of a matching feature is not as strong an indicator of dissimilarity as the presence of one is an indicator of similarity. Other measures like the Euclidean distance do not make this distinction.

4.3. Alignment

Applying the clustering algorithm described in [9], *SIfT* generates a similarity matrix containing the cosine similarity between each pair of columns across databases. For each column from source database *A*, we simply align it with its most similar columns from target database *S* such that the similarity between the pair of columns is above a certain threshold θ.

4.4. Scalability

Computing the similarity matrix described in Section 4.2 is daunting for large element and feature spaces. The complexity of a brute force algorithm is $O(n^2f)$, where n is the number of elements and f is the feature space.

However, a simple heuristic drawing on the properties of mutual information drastically reduces the actual running time of the algorithm. For each element e, we must compute its similarity with every other element by comparing their feature vectors. By sorting the features of element e in decreasing order of mutual information and applying a conservative minimum threshold (e.g., we used a threshold of 0.5 in our experiments), we can reduce the feature vector to only the most informative features. Using reverse indexing on the features, we need only compare e with the elements, E, which share at least one of e's features. Remember that a feature of an element will have high mutual information only if that feature does not coexist with many other elements. Consequently, the size of E will be much smaller than the total number of elements n.

We use the above heuristic in our system. Another option is to use randomized algorithms, which have been shown to reduce the complexity of cosine from $O(n^2f)$ to $O(nf)$; see [10] for details.

5. Evaluation

Given two heterogeneous databases, the goal of our task is to automatically generate the same alignment that a human expert would generate. A step forward is to greatly reduce the number of alignment decisions considered by a human expert.

We evaluate our system on environmental databases. In the next section, we describe our experimental setup. In Section 5.2, we measure the precision and recall of *SIfT* alignments against a manually constructed gold standard as well as measure the reduction in human effort to generate a manual alignment. Then, in Section 5.3, we measure *SIfT*'s accuracy in automatically integrating 2002 California air quality districts' data with the California-wide emissions inventory database using historical data.

5.1. Experimental setup

The source material we use for mappings, in the form of individual data sets, metadata schemas, etc., was provided by the Santa Barbara County Air Pollution Control District (SBCAPCD) and Ventura County Air Pollution Control District (VCAPCD). Both provided a complete archive of the emissions inventory it conducted for 2001 and 2002, covering facilities, devices, processes, permitting history, as well as criteria and toxic emissions. Mapping target material, including an integrated database and its metadata schema, was provided by the California Air Resources Board (CARB), in the form of the statewide emissions inventories for 2001 and 2002.

To be of practical use to our governmental partners, our challenge lies both in the post-analysis of a data transfer between district and state and on integrating new data as it becomes available each year. This is a challenge since the data formats may change on both sides (the collectors and the integrators). Since, however, changes year by year are not likely to be large, we can try to reconcile the possibly divergent evolutions automatically, thereby closing the loop by automatically generating the data integration.

Figure 1 shows our experimental design for automatically generating a data transfer between California districts and CARB for 2002 given historical 2001 data. First, applying *SIfT* as a post-analysis to the 2001 transfer (arrow *a*), we learn the mappings between the data columns for 2001 (evaluation of this step is shown in Section 5.2). Then, given the schema

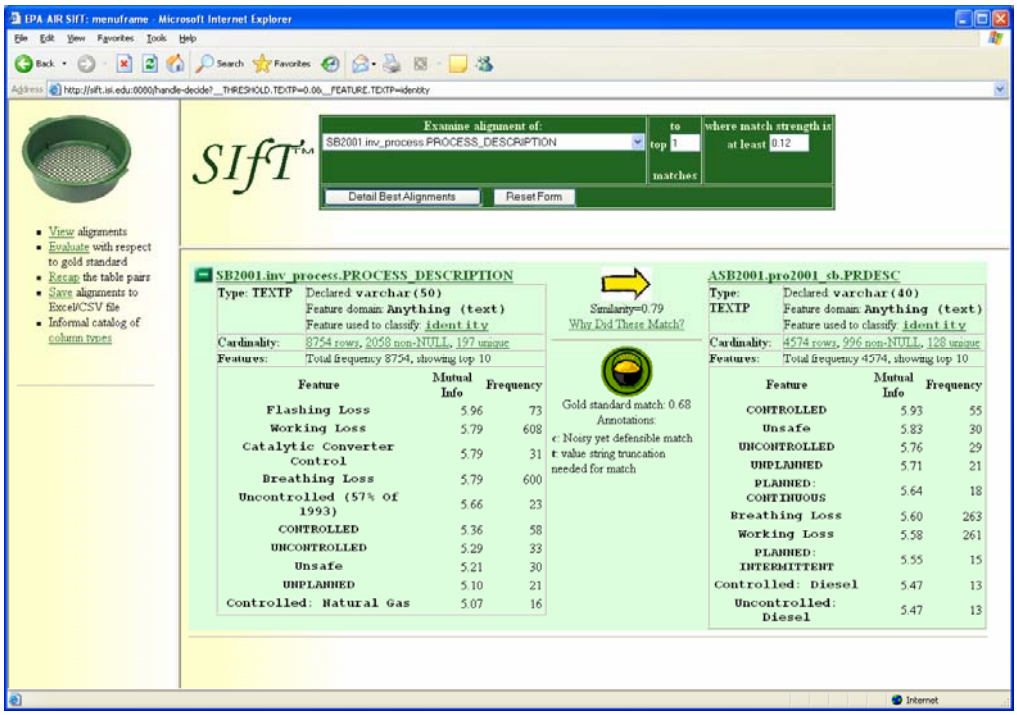


Figure 2. A correct alignment discovered by *SIfT* between the *Process Description* columns in the SBCAPCD and CARB databases. Here, the feature type is "TEXTP" so the fully qualified features are "TEXTP:Flashing Loss", "TEXTP:Working Loss", …

Figure 3. A view of the feature vectors for the *Process Description* columns in the SBCAPCD and CARB databases. The features are sorted in descending order of mutual information. Underlined features are shared by both columns whereas dark gray features are solely from the SBCAPCD column and light gray features are solely from the CARB column. *SIfT* aligns the two columns because they share many high-mutual information features.

changes for both the districts and CARB for 2002 (arrows *c* and *d*), we determine which mappings from *a* still hold. It is not unreasonable to expect that arrows *c* and *d* can be obtained from the district and CARB since schema changes are usually tracked from year to year within a department. Given a district's 2002 data, we can then follow the arrows *c*, *a*, and *d*, to integrate it with the CARB 2002 database, which is arrow *b* (evaluation of this step is presented in Section 5.3).

5.2. Alignment results

In this section, we evaluate *SIfT*'s ability to align columns across data sources (arrow *a* in Figure 1). For the purposes of this evaluation, we focused on the 2001 SBCAPCD and 2001 CARB data. The SBCAPCD and CARB emissions inventory databases used in our experiments each contain approximately 300 columns, thus a completely naïve human must consider approximately 90,000 alignment decisions in the worst case.

5.2.1. Gold standard

We manually aligned the SBCAPCD and CARB emissions inventories from year 2001. This alignment noted and estimated the strength of probable intra-source matches (e.g., likely foreign key or other join relationships) as well as inter-source links (typically, equivalence or subset relationships between columns of the two different databases). While some table-table correspondences and row-row partial equivalences were detected, the primary recorded results consisted of inter-source column associations.

The methodology for constructing the "gold standard" alignment was informal. It would be preferable to have a column-to-column alignment catalog agreed upon by the two agencies, but this was not available as it would require a large investment of labor on the part of our local government partners to develop. The entirety of the gold standard, including annotations, is available from the *SIfT* url: http://sift.isi.edu/.

5.2.2. Precision and recall

SIfT outputs for each column in a source database the columns to which it aligns in the target database[2]. Figure 2 illustrates a screenshot of a correct alignment discovered by *SIfT* between the *Process Description* columns in SBCAPCD and CARB. Figure 3 illustrates why these columns are aligned by *SIfT*. It shows a view of the feature vectors for both columns. The features are sorted in descending order of mutual information; underlined features are shared by both columns whereas dark gray features are solely from the SBCAPCD column and light gray features are solely from the CARB column. *SIfT* aligns the two columns because they share many high-mutual information features.

The precision of the system is the percentage of correct alignment decisions:

$$Precision = \frac{C}{T_A}$$

where C is the number of correct alignment pairs and T_A is the total number of alignment pairs in the system alignment. This type of precision is often called micro-precision. Another precision, called macro-precision, averages the average precision of each column that is being aligned.

The recall of the system is the percentage of gold standard alignment pairs, T_G, which were retrieved by the system:

$$Recall = \frac{C}{T_G}$$

Precision and recall measure the tradeoff between identifying alignments correctly and getting all the possible alignment. For example, a system that returns all possible alignment pairs would achieve a recall of 100% but with an abysmal precision. Increasing the threshold θ from Section 4.3 increases the recall of the system but decreases the precision.

It is sometimes useful to have a single measure that combines precision and recall aspects. One such measure is the *F*-measure [12], which is the harmonic mean of recall and precision:

$$F = \frac{1}{\alpha\frac{1}{R} + (1-\alpha)\frac{1}{P}}$$

where R is the recall and P is the precision. Typically, $\alpha = ½$ is used:

$$F = \frac{2RP}{R+P}$$

F weighs low values of precision and recall more heavily than higher values. It is high when both precision and recall are high.

Table 1 shows the results comparing the precision, recall, and *F*-measure of various different feature representations. The *Simple* system simply represents each column by the data elements it contains. The *Trigram* representation extracts letter trigram features for each field whereas the *Rich* representation extracts all

[2] A customizable interface to the *SIfT* toolkit is available at http://sift.isi.edu/, allowing users to create new alignments, navigate the information theoretic model, and inspect alignment decisions.

Table 1. Precision, recall and F-measure of different feature representations.

	Precision		*Recall*		*F-measure*	
System	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*	*Macro*
Simple	75.0%	65.0%	72.2%	65.0%	73.6%	65.0%
Trigrams	44.4%	44.4%	81.5%	80.0%	57.5%	57.1%
Rich	62.5%	57.7%	79.6%	75.0%	70.0%	65.2%

Table 2. Top-5 precision of different feature representations

System	*Top-1*	*Top-2*	*Top-3*	*Top-4*	*Top-5*
Simple	71.4%	92.9%	92.9%	92.9%	92.9%
Trigrams	66.7%	83.3%	83.3%	83.3%	83.3%
Rich	62.5%	93.8%	93.8%	93.8%	93.8%

possible features domains and feature types described in Section 4.1.1. Each representation uses the information-theoretic vector space model presented in Section 4.1.2.

Curiously, the *F*-measure of the simple representation is higher than the more complicated representations. This is due to the power of the mutual-information vector-space model which in effect automatically discovers the key values of a particular data domain. By inspection, we see that the feature domains in the *Rich* representation are only useful if they have very high precision and recall (e.g., zip codes).

Table 2 shows the precision of our system where the precision indicates the percentage of columns that have at least one correct alignment in the top-5 alignments. Interestingly, if the system can find a correct alignment for a given column, then the alignment will be found in the first two returned candidate alignments. Considering only two candidate alignments for each possible column will greatly reduce the number of possible decisions made by a human expert. Assuming that each of the 90,000 candidate alignments must be considered (in practice, many alignments are easily rejected by human experts) and that for each column we output at most k alignments, then a human expert would have to inspect only $k \times 300$ alignments. For $k = 2$, only 0.67% of the possible alignment decisions must be inspected, an enormous saving in time.

5.3. Projecting *SIfT* alignments

In the previous section, we evaluated *SIfT*'s ability to align columns across databases. Now, we evaluate the task of integrating new data as it becomes available each year (arrow *b* in Figure 1). Using the design in Figure 1, we automatically integrated 2002 VCAPCD and 2002 SBCAPCD databases with CARB's 2002 database using historical 2001 data. Unlike in Section 5.2, since CARB provided us with their 2002 databases, we have a true gold standard against which to compare our integration.

For both VCAPCD and SBCAPCD, we randomly sampled 50 columns in the automatically integrated CARB 2002 databases. A human judge was asked to classify each aligned column according to the following guidelines:

Correct: The column is aligned correctly according to the gold standard.

Partially Correct: The aligned column is a subset or superset of the gold standard alignment. This situation arises when only a selection of the column is transferred to CARB or when a join must be performed on the district tables to match the CARB schema. We must look beyond simple column alignments to solve these problems, which is beyond the scope of this paper.

Incorrect: The column is not aligned correctly according to the gold standard.

Table 3 shows the results of our evaluation. The accuracy of the system is computed by adding one point for each *correct* alignment, half a point for each *partially correct* alignment, no points for each *incorrect* alignment, and then dividing by the sample size.

Some district columns do not get integrated into the CARB database (i.e., *SIfT* does not find any alignment for these columns). In our 50 random samples for

Table 3. Evaluation results for automatically generating a CARB 2002 database from VCAPCD and SBCAPCD 2002 databases. A human judge evaluated random column alignments against a gold standard provided by CARB.

	Sample Size	*Correct*	*Partially Correct*	*Incorrect*	***Accuracy****
VCAPCD	50	25	5	20	**55%**
SBCAPCD	50	22	15	13	**59%**

* Alignments judged as partially correct count ½ points towards the accuracy.

Table 4. Accuracy of the *Top-K* alignments, according to the similarity metric described in Section 4.2, for the 50 random samples from VCAPCD and SBCAPCD.

	Top-1	*Top-5*	*Top-10*	*Top-25*	*Top-50*
VCAPCD	100%	100%	60%	70%	55%
SBCAPCD	100%	100%	95%	76%	59%

VCAPCD, nine columns were left unaligned by *SIfT*, of which six were correct and three were incorrect.

Error analysis shows that *SIfT* is particularly bad at aligning binary (Yes/No or 0/1) columns. Here, the mutual information vector-space model is not useful since binary values are shared by many columns. Such columns, which are easily identified, should be aligned by a separate process. For example, we might simply compare the ratio of 0's vs. 1's or even compare the raw frequency of 0's and 1's. Likely, however, more complex table and row analysis is needed. A possible avenue for future work is to use Kang and Naughton's algorithm [7], described in Section 2, to align these uninformative columns using the other alignments discovered by *SIfT* as seeds.

Each *SIfT* alignment includes a similarity score, as described in Section 4.2. This similarity can be viewed as *SIfT*'s confidence in each alignment. For both VCAPCD and SBCAPCD, we sorted the 50 randomly sampled alignments in descending order of *SIfT* confidence and measured the accuracy for the Top-*K* alignments, for $K = \{1, 5, 10, 25, 50\}$. Note that for binary columns, *SIfT* disregards the similarity score and assigns a 0 confidence score. The results are illustrated in Table 4. As expected, the higher the confidence *SIfT* has in a particular alignment, the higher the chances that this alignment is correct.

6. Conclusions and future work

We proposed an information theoretic model, called *SIfT*, for performing data-driven column alignments. We have applied *SIfT* to the task of aligning the Santa Barbara County Air Pollution Control District and Ventura County Air Pollution Control District's 2001 and 2002 emissions inventory databases with the California Air Resources Board statewide inventory database. *SIfT* yielded 75% precision and 72.2% recall on the column alignment task. On the task of integrating new district data with the statewide database, we achieved 55% accuracy for Ventura County and 59% accuracy for Santa Barbara County.

This work has the potential to significantly reduce the amount of human work involved in creating single-point access to multiple heterogeneous databases. This problem is faced by thousands of large enterprises with numerous data collections, from Government agencies at all levels to the chemical and automotive industries to startup companies that link together and integrate websites. By automatically postulating mappings across databases/metadata, our algorithms can enable the database wrapper builder (whether fully manual or semi-automated) to work more quickly and effectively. It will also help with the creation of metadata standards.

7. References

[1] Ambite, J.L.; Arens, Y.; Gravano, L.; Hatzivassiloglou, V.; Hovy, E.H.; Klavans, J.L.; Philpot, A.; Ramachandran, U.; Ross, K.; Sandhaus, J.; Sarioz, D.; Singla, A.; and Whitman, B. 2002. Data Integration and Access: The Digital Government Research Center's Energy Data Collection (EDC) Project. In W. McIver and A.K. Elmagarmid (eds), *Advances in Digital Government*. pp. 85–106. Dordrecht: Kluwer.

[2] Baru, C.; Gupta, A.; Ludaescher, B.; Marciano, R.; Papakonstantinou, Y.; and Velikhov, P. 1999. XML-Based Information Mediation with MIX. In *Proceedings of Exhibitions Program of ACM SIGMOD International Conference on Management of Data.*

[3] Chawathe, S.; Garcia-Molina, H.; Hammer, J.; Ireland, K.; Papakonstantinou, Y.; Ullman, J.; and Widom, J. 1994. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *Proceedings of IPSJ Conference*. Tokyo, Japan. pp. 7–18.

[4] Church, K. and Hanks, P. 1989. Word association norms, mutual information, and lexicography. In *Proceedings of ACL-89*. pp. 76–83. Vancouver, Canada.

[5] Doan, A.; Domingos, P.; and Halevy, A.Y. 2001. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of SIGMOD-2001*. pp. 509–520. Santa Barbara, CA.

[6] Hovy, E.H. 2003. Using an Ontology to Simplify Data Access. In *Communications of the ACM,* Special Issue on Digital Government. January.

[7] Kang, J. and Naughton, J.F. 2003. On schema matching with opaque column names and data values. In *Proceedings of SIGMOD-2003*. San Diego, CA.

[8] Levy, A.Y. 1998. The Information Manifold approach to data integration. *IEEE Intelligent Systems* (September/October), 11–16.

[9] Pantel, P. and Lin, D. 2002. Discovering word senses from text. In *Proceedings of SIGKDD-02*. pp. 613–619. Edmonton, Canada.

[10] Ravichandran, D.; Pantel, P.; and Hovy, E. 2005. Randomized Algorithms and NLP: Using Locality Sensitive Hash Functions for High Speed Noun Clustering. To appear in *Proceedings of Association for Computational Linguistics (ACL-05)*. Ann Arbor, MI.

[11] Salton, G. and McGill, M.J. 1983. *Introduction to Modern Information Retrieval*. McGraw Hill.

[12] Shaw Jr., W. M.; Burgin, R.; and Howell, P. 1997. Performance standards and evaluations in IR test collections: Cluster-based retrieval methods. *Information Processing and Management*, 33:1–14.

[13] Tova, M. and Zohar, Sagit. 1998. Using schema matching to simplify heterogeneous data translation. In *Proceeding of VLDB-1998*. pp. 122–133.

U.S. Air Force Weather Database and XML Web Services Implementation

Daniel J. Hebert
MITRE
DHebert @MITRE.org

G. Jason Mathews
MITRE
Mathews@MITRE.org

Joseph S. Wood
MITRE
JSW@MITRE.org

Abstract

The Air Force Weather mission is to provide accurate, relevant and timely air and space weather information to many users. Air Force Weather systems store, process, and disseminate large quantities of fine-scale, highly accurate weather information including observations, forecasts, imagery, warnings, and alerts.

This paper discusses the Air Force Weather database—based upon the Joint METOC Conceptual Data Model (JMCDM)—and the Joint METOC Broker Language (JMBL).

JMCDM is a logical data model that integrates the data requirements of the Defense Weather community. Physical database segments are being implemented in compliance with the JMCDM. Current data segments cover observations, gridded data, imagery, text bulletins, etc.

JMBL is an XML data specification that provides the basis to broker the exchange of information between METOC data providers and user applications.

JMCDM and JMBL ensure that terminology, formats, and metadata are standardized across the Defense Weather community.

1. Introduction

The Air Force Weather mission is to maximize our nation's aerospace and ground combat effectiveness by providing accurate, relevant and timely air and space weather information to Department of Defense, coalition, and national users, and by providing standardized training and equipment to Air Force Weather. Air Force Weather systems store, process, and disseminate large quantities of fine-scale, highly accurate weather information. Air Force Weather consists of two major capabilities: terrestrial weather and space weather. Terrestrial weather systems provide weather information in the lower atmosphere (e.g., troposphere from surface up about 50,000 ft or 10 km) and the stratosphere from 10-50 km above the earth's surface, while space weather systems provide weather information in the upper atmosphere to the sun (e.g., solar, interplanetary space, magnetosphere, ionosphere). Air Force Weather systems collect and generate four terabytes of data, including observations, forecasts, imagery, warnings, and alerts each day.

Air Force Weather is divided into three echelons, the strategic center, the operational weather squadrons, and the combat weather teams. The strategic center builds the world's most comprehensive weather database of observation, forecast, climatological, and space weather products available on the World Wide Web. Operational Weather Squadrons provide weather support covering specified regions of the world. Professional meteorologists and weather technicians operate the squadrons around the clock ensuring continuous monitoring of any terrestrial and space weather activity. The Combat Weather Teams are located at installations around the world and are the prime interface with that installation's flying and ground operations. They are the eyes forward, using real-time radar, satellite imagery, sensor readouts, and visual observations to observe and forecast local or deployed conditions.

There are many different types of meteorological data that are generated and disseminated and therefore have to be accounted for in the database schemas and XML representations. Surface weather observations, made at periodic times, measure sky cover, state of the sky, cloud height, atmospheric pressure reduced to sea level, temperature, dew point, wind speed and direction, amount of precipitation, and special phenomena that prevail at the time of the observation or have been observed since the previous specified observation. Terminal Aerodrome Forecasts (TAFs) provide a forecast of weather conditions at airports. Weather warnings indicate that severe weather is occurring or is highly probable and may be issued from six to twelve hours in advance. Surface analysis charts contain fronts and analyzed pressure fields, with the solid lines representing isobars. Upper Air observations are made in the free atmosphere either directly or indirectly and typically monitor temperature, pressure, relative humidity, wind speed and direction, and height of levels. Numerical weather prediction models present an objective forecast of the future state of the atmosphere by solving a set of equations that describe the evolution of variables (temperature, wind speed, humidity, pressure, etc.) that

define the state of the atmosphere. Meteorological satellite (METSAT) data not only encompasses imagery, but also provides input to the preparation of numerical forecasts and research projects. Space weather data includes measures of solar activities such as sunspots and solar flares, and calculations of the effects they may have on the Earth. Radar data indicates motion toward or away from the radar as well as the location of precipitation areas. Lightning detection data captures the number and location of lightning strikes. Environmental effects data capture information on dust, smoke, and volcanic ash in the atmosphere.

2. **The Air Force weather database**

The Joint METOC (Meteorological and Oceanographic) Conceptual Data Model (JMCDM) is a conceptual/logical data model that integrates the data requirements across the Defense Weather community. Physical database segments are being implemented in compliance with the JMCDM, which ensures that terminology, formats, and attribution are standardized across the Defense Weather community. There are ten database segments within the Air Force effort, covering imagery, gridded data, observations, etc. Weather data is being ingested and stored to the standardized database segments, and all information provided is derived from this common database.

The effort to build a common weather database, standardized across the Defense Community, started at the conceptual/logical level. Over 100 individual user views of required weather data were modeled separately, using IDEF1X data models. Examples are air temperature, atmospheric turbulence, cloud observation, etc. The user views overlap with one another. A set of data tables, common to the user views and fully attributed with data elements and primary keys, were then defined. In the next step, mappings were derived between the user views and the central repository of common data tables and data elements. This data modeling and definition effort resulted in the JMCDM with more than 3000 data elements being defined.

A set of physical data models were still required for implementation. These were drafted by the Air Force, and then scrubbed by the Defense Data Standards Working Group to be in compliance with the JMCDM definitions. The overall data modeling approach, from individual user views to a "conceptual view" of all data tables/elements, resulting in a set of implementable physical data models compliant with the standardized data tables and elements is illustrated in figure 1 below. The number of views, data tables, elements, and physical data models are shown on the right.

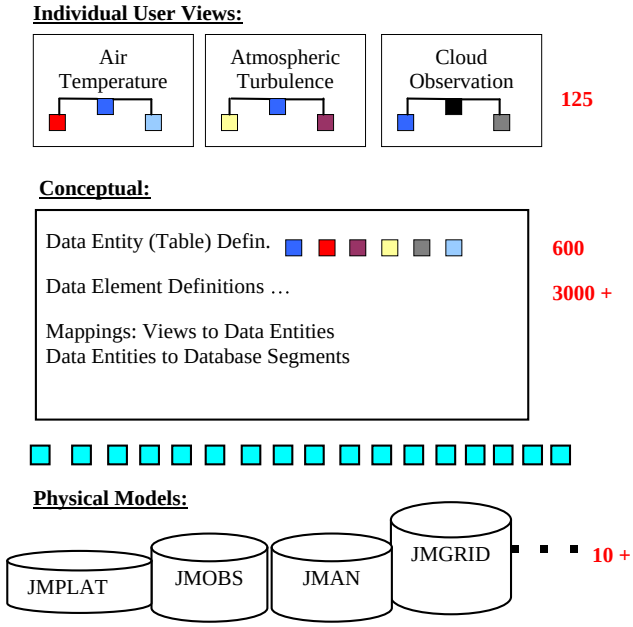


Figure 1. Joint METOC conceptual data model

Everything above the dotted blue line is referred to as the JMCDM (user views, definitions, mappings). The physical data models are derived from the JMCDM and are the most important, implementable subset, with efficiencies introduced such as packing of data tables and combining of data elements, where reasonable to do and when needed for efficiency. The candidate physical data models are organized by major data type: observations, bulletins, weather stations, gridded numerical weather prediction models, catalog, imagery/vector graphics, remote-sensed observations, climatology products, security, solar, meteorological satellite, oceanography, and mission effects. The vision for much of the physical Joint METOC database is shown in figure 2 below.

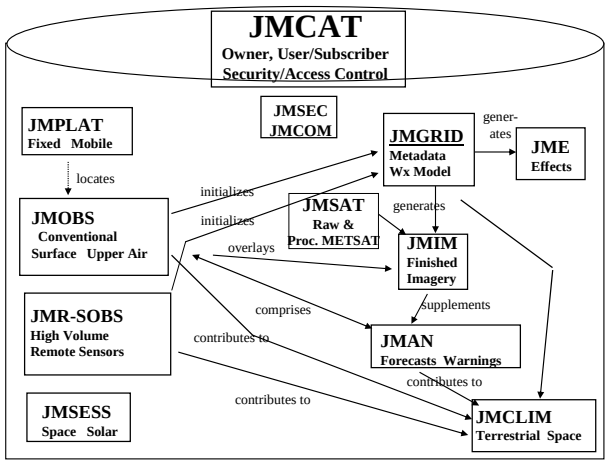


Figure 2. Joint METOC physical data model

3. **The Joint METOC Broker Language (JMBL)**

JMBL is a specification for a set of XML Schemas and Web Service Definition Language (WSDL), which provides the basis to broker the exchange of information between METOC data providers and user applications. JMBL defines the XML tags to represent and query weather data and their associated metadata. Data is requested by data type with further restriction by geographic location, time, platform and other characteristics. JMBL is an abstraction layer on the physical database, where the end user is concerned only with which data types and weather parameters are needed.

A weather parameter list table supports run-time access to the physical database segments from incoming JMBL requests. The JMBL requests carry standard weather parameter names. Data access layer software matches each parameter name in the request with parameter name table entries, and picks up the physical database segment's data element name associated with each of the JMBL request's weather parameters. See figure 3 below.

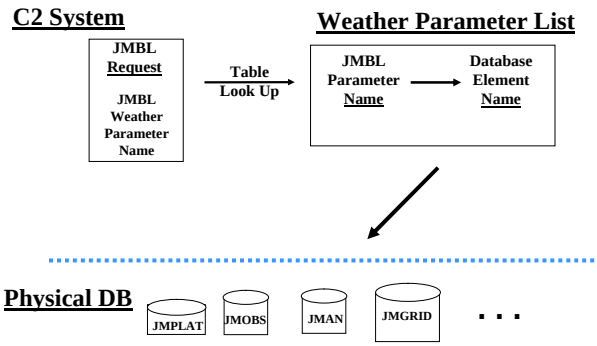


Figure 3. System access to Joint METOC database

The JMBL request structure first decomposes the data into 10 data or product types (gridded data, imagery, observations, alphanumeric, climatology, remote sensed, product, platform, space weather, and best source) each of which has its own specific constraints and properties (e.g. forecast period, resolution, image type, etc). Common request criteria (e.g., time, location, format, distribution method, etc.) are also specified in the request structure. The data type combined with other request criteria represent the who, what, where, when and how to return the data.

Extensibility of JMBL is assured with the addition of new data type choices as needed without breaking existing clients that are not aware of these new models. Likewise, the response structure parallels the request structure with the who, what, where, and when needed to describe the data.

Consumers can request a catalog of available products or the raw weather products (GRIB, METAR, SPECI, TAF, etc.) or extract only the pieces of information that are relevant to them such as specific parameters (e.g., temperature and wind speed) and returned units of measure. A typical request such as one for surface observations at the Kerrville Municipal Airport (noted with the platformId *KERV*) is shown below. In this example the wind speed and air temperature are requested along with the raw observation data.

```
<RequestList xmlns="urn:metoc:jmcbl">
<Request>
<InformationType>
<MetocDataType>
<Observation>
<PlatformCode>
<PlatformList><PlatformId>KERV</PlatformId>
</PlatformList>
</PlatformCode>
<ObservationParameters>
<Parameter parameterName="observationDateTime" />
<Parameter parameterName="temperatureAir"
parameterUnit="degreesCelsius" />
<Parameter parameterName="windSpeed" />
<Parameter parameterName="observationRawData" />
</ObservationParameters>
<Time>
<TimeRange startTime="2005-01-12T12:51:00Z"/>
</Time>
<ObservationReportTypeCode>FM-
15</ObservationReportTypeCode>
<ObservationReportTypeCode>FM-
16</ObservationReportTypeCode>
</Observation>
</MetocDataType>
</InformationType>
</Request>
</RequestList>
```

Figure 4. JMBL request for surface observations

A response for such a request would look something like the following if data was available:

```
<ResponseList xmlns="urn:metoc:jmcbl:jmbl">
<Response>
<DataItem>
<Time>
<TimeRange ns2:startTime="2005-01-12T12:51:00Z"/>
</Time>
<METOCdata>
```

<METOCdata dataElementName="observationDateTime" parameterUnit="dateTime">
<Value>
<LongValue>20050112180500</LongValue>
</Value>
</METOCdata>
<METOCdata dataElementName="temperatureAir" parameterUnit="degreesCelsius">
<Value>
<DoubleValue>18.0</DoubleValue>
</Value>
</METOCdata>
<METOCdata dataElementName="windSpeed" parameterUnit="metersPerSecond">
<Value>
<DoubleValue>6.1</DoubleValue>
</Value>
</METOCdata>
<METOCdata METOCtype="JMO_MESSAGE" dataElementName="observationRawData">
<Value>
<StringValue>METAR KERV 121805Z AUTO 18012G20KT 10SM BKN019 BKN023 BKN028
18/14 A2967 RMK AO2</StringValue>
</Value>
</METOCdata>
</METOCdata>
</DataItem>
<ResponseStatus orderStatus="Request Filled"/>
</Response>
</ResponseList>

Figure 4. JMBL response of surface observations

With the representation offered by JMBL, weather data can be expressed in both requests and responses. Next, for a standard common across multiple implementations a number of business rules are needed. Air Force programs have defined business rules such as determining if an image matches the request criteria when the requested boundary region intersects the region of the image or is fully contained within. Likewise, rules are needed if a boundary region (request) is smaller than the available image, since returning a cropped image is one option. Or, with a gridded data request, should the gridded data (response) be the full grid, or only a subgrid to the requested region. At present images are returned as-is when the region is fully inside the image boundary while gridded data are subgridded to the requested region.

4. Summary

This paper has provided information on the development of XML and database schemas within the Air Force Weather systems. It provided an overview of Air Force Weather operations and data and then provided a detailed description of the database design approach and the use of XML and SOAP-based web services.

Author biography

Daniel J. Hebert is currently the Chief Engineer for the AF Weather Program at the MITRE Corporation. He holds a Masters degree in Computer Science from Florida State University. With over 25 years of experience in the field, he has contributed to a broad range of programs including the Space Shuttle, Joint Stars, and the IRS Modernization. Primary areas of expertise include database technologies, operating systems, software architectures, and geographic information systems. He is also an adjunct professor at Boston University teaching courses in databases, data warehousing/mining, XML Technologies, and geographic information systems.

G. Jason Mathews is a Lead Software Engineer for the MITRE Corporation currently supporting the AF Weather Program. He holds a MS in Computer Science from the George Washington University. Jason has over ten years experience developing complex and challenging software applications. His current interest is web service and XML technologies with a focus on interoperability issues. Prior to MITRE, Jason was a Senior Computer Engineer at NASA's Goddard Space Flight Center where he pioneered Web-based data analysis and visualization systems for space imagery and gridded data.

Joseph S. Wood is a Principal Engineer specializing in database systems at The MITRE Corporation. He earned a Masters degree in Computer Systems/Operations Research from American University and a Bachelor's degree in Mathematics from Tufts University. With 35 years experience, he has worked on an assortment of large-scale data management system projects in the defense, intelligence, and corporate sectors.

Creating and Providing Data Management Services for the Biological and Ecological Sciences: Science Environment for Ecological Knowledge

Samantha Romanello[1], James Beach[2], Shawn Bowers[4], Matthew Jones[3], Bertram Ludäscher [4], William Michener[1], Deana Pennington[1], Arcot Rajasekar[5], Mark Schildhauer[3]
[1]Univ. New Mexico; [2]Univ. Kansas; [3]UC-Santa Barbara; [4]UC-Davis; [5]UC-San Diego
sroman@LTERnet.edu, beach@ku.edu, sbowers@ucdavis.edu, jones@nceas.ucsb.edu, ludaesch@ucdavis.edu, wmichene@lternet.edu, dpennington@lternet.edu, sekar@sdsc.edu, schild@nceas.ucsb.edu

Abstract

The Science Environment for Ecological Knowledge (SEEK) [1] is an information technology project designed to address the many challenges associated with data accessibility and integration of large-scale biocomplexity data in the ecological sciences. The SEEK project is creating cyberinfrastructure encompassing three integrated systems: EcoGrid, a Semantic Mediation System (SMS) and an Analysis and Modeling System (AMS). SEEK enables ecologists to efficiently capture, organize and search for data and analytical processes (i.e., scientific workflows) from their desktop in a user friendly interface -- ultimately providing access to global data and analytical resources typically out of reach for many ecologists. The prototype application is ecological niche modeling.

1. Introduction

The spread of the West Nile Virus, the emergence of invasive species and the effects of climate change on biodiversity and the environment are challenging ecological issues that rely heavily on acquisition of data from diverse sources and intensive computational effort. Ecological issues like these and others highlight the critical need of scientists, researchers and policy makers' to have rapid access to available data. The objective of the SEEK project is to increase the speed and efficiency of data acquisition, integration, analysis and synthesis in the biological and ecological sciences. SEEK scientists and developers are building a three-tiered information technology infrastructure composed of the EcoGrid, the Semantic Mediation System and the Analysis and Modeling System (Figure 1). The EcoGrid is an open architecture for data access across organizational and institutional boundaries. The Semantic Mediation System (SMS) is a "smart" data discovery and integration system based on domain-specific ontologies. The Analysis and Modeling System (AMS) implemented thru the Kepler workflow system supports semantically integrated analytical workflows. With the development of this infrastructure SEEK is poised, not only to provide global access to ecological data and information but also to facilitate ecological and biodiversity forecasting.

SEEK enhances the national and global capacity for observing, studying, and understanding biological and environmental complexity in several ways. First, through the development of intelligent analytical tools and an infrastructure capable of semantically integrating diverse, distributed data sources, it removes key barriers to knowledge discovery. Second, SEEK enables scientists to exercise powerful new methods for capturing, reproducing, and extending the analysis process. Third, by expanding access to distributed and heterogeneous ecological data, information, and knowledge, SEEK creates new opportunities for scientists, resource managers, policy makers and the public to make informed decisions about the environment. Finally, it provides an infrastructure for educating and training the next generation of ecologists in the information technology skills that are critical for scientific breakthroughs in the future. This paper begins with a brief description of the SEEK project, describes the three-tiered information technology infrastructure of the SEEK project and the prototype application, and concludes with a report on significant findings to date.

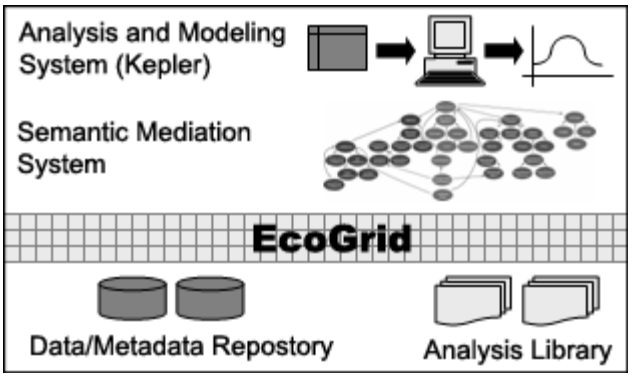


Figure 1. SEEK architecture

2. SEEK Community

SEEK is a multi-disciplinary, multi-institutional and multi-national effort designed to create cyberinfrastructure for ecological, environmental, and biodiversity research and to educate the ecological community about ecoinformatics. SEEK infrastructure development is supported by software engineers and computer scientists dispersed across the eight institutions involved in the project. The design and development of the SEEK cyberinfrastructure is informed by three multidisciplinary teams of scientists organized in Working Groups. The Biodiversity and Ecological Analysis and Modeling Working Group (BEAM) informs development through evaluation of SEEK efficacy in addressing biodiversity and ecological questions. A Knowledge Representation Working Group (KR) develops formal ontologies that enable the assembly of analytical workflows in the Analysis and Modeling System and access to source data in EcoGrid. A Biological Classification and Nomenclature Working Group (Taxon) investigates solutions to mediating among multiple taxonomies for naming organisms. Additionally, a multifaceted Education, Outreach and Training (EOT) program ensures that the SEEK research products, software, and information technology infrastructure optimally benefit the target communities via the project website (http://seek.ecoinformatics.org) [2, 3].

3. The EcoGrid

The EcoGrid [4] is a collection of distributed ecological, biodiversity and environmental data and analytic resources (data, metadata, analytic workflows and processors) that are often located at different sites and in different organizations. The EcoGrid uses the Open Grid Services Architecture (OGSA, http://www.globus.org/ogsa/) framework to provide a set of standardized interfaces for accessing data resources through a service-oriented framework. The current prototype implementation of the EcoGrid uses the OGSA 'Factory" service to enable scalable deployment of the access services, but in other aspects is more similar to a traditional web service implementation. As the Web Services Resource Framework (WSRF) matures we will investigate migrating our services to the WSRF specification. EcoGrid combines features of a Data Grid for ecological data management and a Compute Grid for analysis and modeling services. EcoGrid forms the underlying framework for data and service discovery, data sharing and access and analytical service sharing and invocation.

Biodiversity and ecological data include, but are not limited to the heterogeneous data collected at field stations, as well as remote sensing data and data from museum collections. Computational models and analyses include well-known biodiversity and ecosystem models such as GARP (Genetic Algorithm for Ruleset Production; Stockwell and Noble, 1992 [5]; University of Kansas Center for Research, 2002 [6]) and CENTURY (Natural Resource and Ecology Lab, 2003 [7]) as well as custom models and analyses written for a single experiment or study. The SEEK EcoGrid is being designed to provide the infrastructure for managing these diverse data and computational resources.

4. Analysis and Modeling System

The Analysis and Modeling System in SEEK is a multiplatform, open-source, visual programming tool (i.e., the Kepler [8, 9, 10] workflow system) that allows users to create executable analytical pipelines and workflows based on research models. Kepler, based on Ptolemy II [11], is a collaborative project involving contributing members from SEEK, the GEOsciences Network (GEON) http://www.geongrid.org/, the Scientific Data Management Center (SDM) part of the Scientific http://sdm.lbl.gov/sdmcenter/, the Ptolemy Project http://ptolemy.eecs.berkeley.edu/ptolemyII/, the ROADNet (Real-time Observatories, Applications, and Data Management Network) Project http://roadnet.ucsd.edu/ and the EOL (Encyclopedia of Life) http://eol.sdsc.edu/. Scientific workflows are a formalization of the scientific research process (Figure 2). That is, typically a scientist will generate a research question, collect data, analyze the data using several models, programs, software and hardware, and physically coordinate the data transformation, exporting and importing.

Kepler allows scientists to design, execute, monitor, re-run and communicate analytic procedures with minimal effort. Therefore, scientific workflows in Kepler include the analysis steps as well as the data acquisition, integration, transformation, synthesis and archival

information. Scientists can create and save workflows on the EcoGrid within Kepler. These workflows can then be searched and downloaded by other researchers for replicating or expanding upon the analysis.

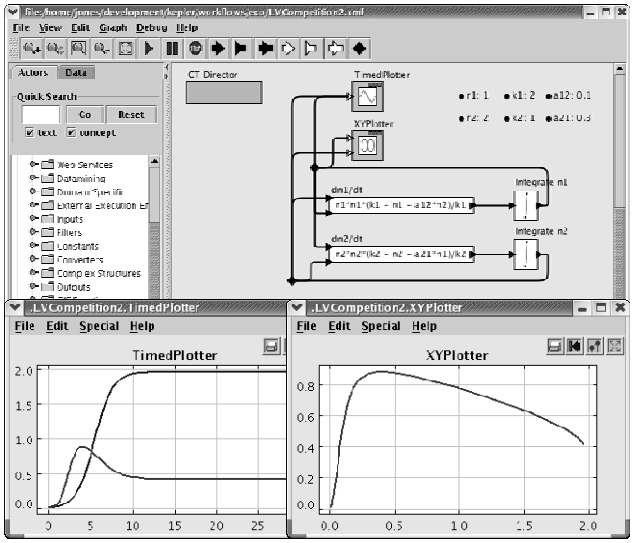


Figure 2. The Kepler workflow system showing the Lotka-Volterra predator-prey model.

5. Semantic Mediation System

The Analysis and Modeling System (i.e., Kepler) in the SEEK architecture leverages the Semantic Mediation System (SMS) [12, 13]. The goal of the SMS layer is to support scientists' workflow modeling and design processes. In particular, SMS exploits domain ontologies to facilitate (1) "smart discovery" of data sets and components (individual actors and complete workflows), (2) "smart binding" of data sets to components, and (3) "smart linking" of components to each other as part of the overall design process.

The Semantic Mediation System provides a generic set of ontology-based languages and tools for storing and exploiting "superimposed" semantic annotations [15], which explicitly link existing data sets and workflow components to ontologies. Through semantic annotations, the mediation layer provides knowledge-based data integration and workflow composition services [13,14], as well as basic services used in workflow modeling, such as ensuring that workflows are "semantically" type-safe (based on annotations) and component and data discovery via concept-based searching.

Ontology development is a major part of SEEK. We have developed initial ontologies for ecological data and workflows, focusing on measurements, basic ecological concepts, symbiosis, and biodiversity. We are also building tools to support the editing and curation of ontologies, with the goal of making these tools accessible and easy to use for domain scientists.

6. Prototype application

A new and promising paradigm in biodiversity informatics is the use of ecological niche modeling to extrapolate and anticipate implications of global climate change for biological diversity [17, 18, 19]. Future scenarios based on general circulation models (GCMs) present diverse visions of global climate futures. The implications of these different futures for biodiversity are only now being explored. While data suggest that climates are changing, the implications of these changes remain unclear and little explored. At this time there are no hemisphere-wide evaluations or broad comparative analyses of implications of different GCM modeling scenarios, due to the prohibitive time costs for large-scale analyses. With the use of distributed resources and the building of analytic workflows for automated processing of climate change and biodiversity analyses, the first application for the SEEK project is a large scale ecological niche modeling assessment of mammals of Western Hemisphere to look at the implications of climate change on current and projected habitat range. This application models distributions of all mammal species in the Western Hemisphere and generates projections of distribution change under multiple Intergovernmental Panel on Climate Change scenarios (http://www.ipcc.ch). This project includes the analysis of integrated field data for over 3000 mammal species, under 20+ climate scenarios, using 2-3 dispersal scenarios (180,000+ model runs).

7. Significant results

To date, a variety of SEEK tools have been created including a protoype of the Ecogrid has been created (Table 1).

1. Currently the Ecogrid provides: access to different data catalogs (Metacat, SRB/MCAT, DiGIR); a search, read and write interface; and uploading of metadata and data.
2. The first alpha-quality users' release of Kepler was in May 2004. Kepler has an Ecological Metadata Language -aware data plug-in, an EcoGrid plug-in, web service actors and a web service harvester.
3. Toolkit for reasoning and data conversion, and an access API for ontologies called “Sparrow” was developed. A user-friendly editor for OWL ontologies (grOWL) is currently in its second alpha release.

4. The Biological Classification and Nomenclature (Taxon) working group has created a "Taxonomic Object Service" (TOS) that provides information about the relationships among taxa via SOAP and web interfaces.

Table 1. SEEK Tools

Application	Description	url
Kepler	Is a flexible workflow system designed to process and ingest heterogeneous ecological data from ecologists and other domain scientists.	http://kepler.ecoinformatics.org
Sparrow	Aims at combining algorithms and techniques from logic-based knowledge representation and databases into a single, open-source toolkit.	http://seek.ecoinformatics.org/
GrOWL	Is a visualization and editing tool for Ontology Web Language (OWL) and Description Logics (DL) ontologies based on a semantic network knowledge representation paradigm	http://ecoinformatics.uvm.edu/dmaps/growl
EcoGrid	Is a thin layer to allow various data and computer services already in existence to interoperate base on GRID technology.	http://seek.ecoinformatics.org/

Additional information about these and other SEEK tools can be found at http://seek.ecoinformatics.org.

8. Acknowledgements

This work is supported in part by NSF grants ITR 0225674 (SEEK), 0225673 (GEON), EF 0225665 and DBI 0129792; DARPA N00014-03-1-0900 and DOE SciDAC DE-FC02-01ER25486 (SDM).

9. References

[1]SEEK: Science Environment for Ecological Knowledge, http://seek.ecoinformatics.org

[2]W. K. Michener, J. H. Beach, M. B. Jones, B. Ludaescher, D. D. Pennington, R. S. Pereira, A. Rajasekar, and M. Schildhauer, "A Knowledge Environment for the Biodiversity and Ecological Sciences", *Journal of Intelligent Information Systems*, 2004.

[3]W K. Michener, "Building SEEK: the Science Environment for Ecological Knowledge ", *DataBits: An electronic newsletter for Information Managers*, Spring, 2003.

[4]M. B. Jones, "SEEK EcoGrid: Integrating Data and Computational Resources for Ecology", *DataBits: An electronic newsletter for Information Managers*, Spring 2003.

[5]D.R.B.Stockwell, and I.R. Noble. "Induction of Sets of Rules From Animal Distribution Data: A Robust And Informative Method of Data Analysis". *Mathematics and Computers in Simulation*, 32, 1992, pp. 249–254.

[6]http://www.lifemapper.org/desktopgarp/#acknowledge

[7] http://www.nrel.colostate.edu/projects/century/

[8]Kepler: An Extensible System for Scientific Workflows, http://kepler.ecoinformatics.org

[9]I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, S. Mock, Kepler: Towards a Grid-Enabled System for Scientific Workflows, *Workflow in Grid Systems, GGF10,* Berlin, 2004.

[10] I. E. Altintas, E. Jaeger, K. Lin, B. Ludaescher, and A. Memon. A Web Service Composition and Deployment Framework for Scientific Workflows. In *2nd Intl. Conference on Web Services (ICWS),* San Diego, California, July 2004.

[11] E. A. Lee, "Overview of the Ptolemy Project," *Technical Memorandum UCB/ERL M03/25*, University of California, Berkeley, CA, July 2, 2003.

[12] C. Berkley, S. Bowers, M. Jones, B. Ludaescher, M. Schildhauer, J. Tao. Incorporating semantics in scientific workflow authoring. In *Proceedings of the 17th International Conference on Scientific and Statistical Database Managemen*t (SSDBM'05)

[13]B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, Y. Zhao, "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice & Experience,* Special Issue on Scientific Workflows, to appear, 2005

[14] S. Bowers and B. Ludaescher, "An Ontology-Driven Framework for Data Transformation in Scientific Workflows," In *Proceedings of the International Workshop on Data Integration in the Life Sciences (DILS'04),* volume 2994.A, Springer, LNCS, Leipzig, Germany, March 25-26, 2004

[15] S. Bower, D. Thau, R. Williams and B. Ludaescher, "Data Procurement for Enabling Scientific Workflows: On Exploring Inter-Ant Parasitism" In *Proceedings of the 2nd International Workshop on Semantic Web and Database*s (SWDB'04), Toronto, Canada, 29-30 August, 2004.

[16] S. Bowers, K Lin, and B. Ludaescher, "On Integrating Scientific Resources through Semantic Registration," In *Proceedings of the 16th International Conference on Scientific and Statistical Database Managemen*t (SSDBM'04), 21-23 June 2004, Santorini Island, Greece.

[17]A.T. Peterson, and D. A. Vieglais. "Predicting Species Invasions Using Ecological Niche Modeling," *BioScience*, 51, 2001, pp. 363-371.

[18]Peterson, A.T., H. Tian, E. Martínez-Meyer, B. Huntley, J. Soberón, and V. Sánchez-Cordero. Modeling distributional shifts of individual species and biomes. In T. E. Lovejoy and L. Hannah (eds.), *Biodiversity and Climate Change*, Yale University Press, New Haven, Conn, 2005f.

[19]A. T. Peterson, M. A. Ortega-Huerta, J. Bartley, V. Sanchez-Cordero, J. Soberon, R. H. Buddemeier, and D. R. B. Stockwell, "Future Projections for Mexican Faunas Under Global Climate Change Scenarios," *Nature*, 416, 2002b, pp. 626-629.

Session 2: Environmental Applications – 2

DEX: Increasing the Capability of Scientific Data Analysis Pipelines by Using Efficient Bitmap Indices to Accelerate Scientific Visualization

Kurt Stockinger, John Shalf, Wes Bethel, and Kesheng Wu
Lawrence Berkeley National Laboratory
University of California
1 Cyclotron Road, Berkeley, CA 94720, USA

Abstract

We describe a new approach to scalable data analysis that enables scientists to manage the explosion in size and complexity of scientific data produced by experiments and simulations. Our approach uses a novel combination of efficient query technology and visualization infrastructure. The combination of bitmap indexing, which is a data management technology that accelerates queries on large scientific datasets, with a visualization pipeline for generating images of abstract data results in a tool suitable for use by scientists in fields where data size and complexity poses a barrier to efficient analysis. Our architecture and implementation, which we call DEX (short for dexterous data explorer), directly addresses the problem of "too much data" by focusing analysis on data deemed to be "scientifically interesting" via a user-specified selection criteria. The architectural concepts and implementation are applicable to wide variety of scientific data analysis and visualization applications. This paper presents an architectural overview of the system along with an analysis showing substantial performance over traditional visualization pipelines. While performance gains are a significant result, even more important is the new functionality not present in any visualization analysis software – namely the ability to perform interactive, multi-dimensional queries to refine regions of interest that are later used as input to analysis or visualization.

1 Introduction

Bitmap Indices are index data structures for efficiently querying high-dimensional data sets. Such queries form the basis of data analysis, which is a central task in the scientific process. Several data warehouses and scientific applications use bitmap indices to efficiently access large amounts of read-only data. Over the last few years, we have developed and deployed bitmap index software (FastBit) that is now used in production analysis of data produced by high-energy physics experiments [26] and for feature extraction and 2D/3D region growing in astrophysics and combusion applications [24, 19]. The performance and functional gains that bitmap indexing provide to data-intensive applications are particularly germane to scientific visualization where increasing data size and complexity often exceed the capacity of current visualization architectures as well as overwhelm the scientist with too much visual data.

FastBit software permits scientists to define the subset of data cells that "are interesting" using compound Boolean expressions. For example, in the field of combustion, datasets typically have dozens of quantities per cell representing the concentration of various chemical species involved in the combustion process along with fluid dynamics variables like pressure, density and velocity. Combustion researchers are often concerned with tracking combustion processes on the flame front, but the definition of a flame front turns out to be difficult to objectively define in a simple way. Instead, a flamefront is best defined as a set of criteria expressed as a conjunction of boolean clauses: cells where temperature exceeds some threshold and the concentration of one or more chemical species lies within some range. Modern scientific datasets are so large and complex that applying visualization techniques to the entire dataset often results in a "thicket of visual noise" where interesting features are visually obscured. It is difficult to remove the visual noise using the clipping, cropping, and sub-setting techniques that are typically available in visualization tools because the feature of interest, the flame front, is topologically complex. Instead, researchers construct a boolean expression specifying conditions likely to contain the flamefront, then FastBit rapidly selects the volume of interest, or data cells that match the selection criteria. Within the DEX tool, these volumes of interest are then used as input to a standard visualization pipeline where other methods can be applied to visualize the data selected by the query operation.

FastBit keeps a bitmap for each distinct attribute value

or attribute range (see Section 2). Queries are processed by evaluating a subset of all bitmap indices, which is often considerably smaller than the entire data set. Hence, bitmap indices are able to resolve complex queries using only a fraction of the memory and time that would be required to process the entire dataset. As a consequence, the time complexity of the visualization algorithm can also be significantly reduced for large datasets.

Therefore, the advantages of DEX are two-fold. DEX helps reduce the visual complexity of a visualization by focusing the visualization algorithms exclusively on potentially topologically complex regions of interest that are defined by the query. DEX also offers significant advantages for large datasets because the efficient indexing scheme enables the visualization system to load only the data cells selected by the query rather than examining all cells in the entire dataset. We refer to this visualization methodology as "Query-Based Visualization."

The main contributions of this paper are as follows:

- DEX combines highly efficient data management techniques with visualization. The combination represents a promising novel approach for high capacity and capability analysis and visualization implementations. We describe our implementation of the DEX tool and the underlying architecture of FastBit.
- We describe visualization methods enabled by "query-based visualization" technology and support them with example use-cases in combustion and astrophysics research.
- Using two scientific datasets, we compare the performance of DEX with that of traditional visualization algorithms.

  This paper focuses on two datasets that are representative of the output from cutting-edge scientific codes in High Performance Computing. The first dataset is a temporally evolving reacting methane-air jet from the TeraScale High-Fidelity Simulation of Turbulent Combustion with Detailed Chemistry [21]. The second dataset is a supernova explosion from the TeraScale Supernova Initiative [22].

The paper is organized as follows. In Section 2 we review the prior work on bitmap indices and visualization problems that are within the scope of DEX. Section 3 discusses index-based data extraction and explains how bitmap indices are used for region growing and feature extraction. In Section 4 we introduce the architecture of DEX along with an informal time complexity analysis. Typical scientific data exploration and visualization use cases a given in Section 5. Performance results are presented in Section 6. We conclude the paper in Section 7 and introduce some future research challenges.

2 Related Work

In the following section, we review the concept of bitmap indexing with particular emphasis upon how the bitmap index accelerates multi-dimensional data query operations. One of the first visual query systems is VisDB [14] that combines database techniques with novel visualization methods. Our work also uses database techniques for query processing but the main goal of our query processing is to identify regions of interest for later processing. In the examples we present in this paper, we display the cells in the regions of interest using a cuberille-style rendering [11]. Future work will include using the selected regions of interest as input to analysis algorithms, computation of derived quantities, and more sophisticated visualization techniques.

To set the stage for later sections of this paper, we discuss the visualization pipeline, with particular emphasis upon isocontouring. Isocontouring is a staple visualization technique that performs an operation on cells that satisfy a single criteria. To that end, the part of the isocontouring algorithm that finds cells satisfying a single criteria is similar to the more general and difficult problem of efficient, multi-dimensional data searching.

2.1 Bitmap Indices

Bitmap indices are one of the most efficient indexing schemes available for speeding up multi-dimensional range queries for read-only or read-mostly data [17, 25]. For an attribute with c distinct values, the basic bitmap index [6] generates c bitmaps with N bits each, where N is the number of records (cells) in the dataset. Each bit in a bitmap is set to 1 if the attribute in the record is of a specific value, otherwise the bit is set to 0. For example, the integer attribute **I** shown in Figure 1 can be one of four distinct values, 0, 1, 2, and 3. The corresponding bitmap index has four bitmaps. Since the value in record 5 is 3, the fifth bit in b_4 is set to 1 and the same bits in other bitmaps are 0. In short, 4 bitmaps are required to encode 4 distinct attribute values.

Bitmap indices are efficient for processing multi-dimensional range queries such as "$\mathbf{I} < 2$ and $\mathbf{J} > 3$". The queries are evaluated with bitwise logical operations that are well-supported by computer hardware.

For data sets where a given variable may span a large number of distinct values, one concern with bitmap indexing is that the amount of space required by the bitmap index could become quite large. One way of reducing the storage requirement is to use bitmap compression. Another is to use a binning strategy, which is described below. Note that an efficient bitmap compression scheme not only has to reduce the size of bitmaps but also has to perform bitwise Boolean operations efficiently.

Several bitmap compression methods were studied in

RID	I	bitmap index			
		=0	=1	=2	=3
1	0	1	0	0	0
2	1	0	1	0	0
3	3	0	0	0	1
4	2	0	0	1	0
5	3	0	0	0	1
6	3	0	0	0	1
7	1	0	1	0	0
8	3	0	0	0	1
		b_1	b_2	b_3	b_4

Figure 1. A sample bitmap index where RID is the record ID and I is the integer attribute with values in the range of 0 to 3.

[1, 13]. The authors demonstrated that the scheme named Byte-aligned Bitmap Code (BBC) [2, 3] shows the best overall performance characteristics. More recently a new compression scheme called Word-Aligned Hybrid (WAH) [25] was introduced. It has been shown that even in the worst case, the bitmap indices can be compressed to a size that is comparable with a typical B-tree index. The time required to answer a range query using a compressed bitmap index is in fact optimal. In the worst case, the response time is proportional to the number of hits of the query [25].

The bitmap indices discussed so far encode each distinct attribute value as one bitmap vector. This technique is very efficient for integer or floating point values with low attribute cardinalities. However, scientific data is often based on floating point values with high attribute cardinalities. The work presented in [20] demonstrated that bitmap indices with binning can significantly speed up multi-dimensional queries for high-cardinality attributes.

2.2 Visualization

The initial demonstration of the DEX tool displays the cells selected by the query. Visually, the selection appears as a "blocky" isosurface. While there is visual similarity between an isosurface computed over a scalar field and the set of cells returned from a complex multidimensional query, the two methods – isosurfacing and bitmap indexing – cannot be directly compared due to a fundamental difference in generality. Specifically, bitmap indices are evaluating multi-dimensional comparisons to define a *volume of interest* whereas the isosurface is evaluating a scalar to find a *surface of interest.* Despite the fundamental difference in generality, a comparison between DEX and isosurface algorithms is warranted because the isosurface is the one of the most commonly employed visualization techniques. A time-consuming processing step in any isocontouring algorithm is the search for the data cells that satisfy a criteria, namely, that a surface passes through a cell. We are focusing our performance comparison between bitmap indexing and isocontouring on the task of searching for cells that satisfy a single criteria.

To satisfy a search consisting of multiple criteria in a traditional visualization pipeline, one can compute a "derived field" that represents an evaluation of a multi-dimensional objective function producing a scalar field that can then be isosurfaced. Such derived values are akin to an expensive join operation. The isosurface algorithm is then used to draw a contour (equipotential surface) around the scalar to identify the topologically complex region of interest. If the objective changes, then the algorithm must reload all of the data in order to derive a new scalar using the new objective function. DEX provides this very functionality, but does not need to reload data or derive new fields to evaluate a new objective function. We will demonstrate the efficiency benefits of our approach in Section 6 of the paper.

The bitmap indices are far more general than isosurfaces because they identify a *volume of interest* rather than a *surface of interest*. Not only do the bitmap indices support evaluation of equality ($x = v$, the definition of isosurface), they can find regions that are less than a value ($x < v$), greater than a value ($x > v$), or any complex expression that can be constructed from a Boolean combination of those expressions ($x_1 > v_1$ and $x_2 \leq v_2$ and ...). The isosurface algorithm generates surface normals, which can be used to implicitly identify the interior of a volume of interest. However, using the surface normals to convert the selected surface-of-interest into a volume of interest requires an additional, potentially expensive algorithmic step. Consequently, the locus of comparison between DEX and the isosurface algorithm is restricted to evaluation of equality (surface finding). This artificial limit helps provide a basis for comparing the performance of isocontouring to bitmap index queries by evaluating simple equality expressions (e.g. find all cells satisfying the expression (SELECT data from t WHERE cell[i]=scalar_value).

The most widely used isosurfacing technique is Marching Cubes [16]. Marching cubes improves the efficiency of surface generation at each cell that intersects the surface, but must examine every cell in the dataset to find the cells that intersect the surface. Marching cubes accelerate surface generation by using a state table that enumerates the finite number of surface-edge intersections. A number of algorithms have been developed to accelerate the process of finding cells that intersect the desired surface. These operate by reducing the number of cells that must be evaluated by culling cells that are either out of range (value based) or are not visible due to occlusion (view based). According to [8] isosurfacing algorithms can be classified as either view-dependent or view-independent. The view-dependent algo-

rithms mainly perform computation on regions that make up substantial parts of the final image. View-independent algorithms, on the other hand, generate geometry on all cells containing the surface regardless of whether or not they are visible. Since one of the goals of DEX is to perform interactive, feature-based analysis, view-independent algorithms form the basis of comparison in this paper.

One key technique used in view-independent isosurface acceleration is I/O-optimal isosurfacing where interval-tree indexing structures help rapidly locate those cells containing the surface. A number of methods make use of octrees [23] for searches, but the octree approach is impractical for data containing small-scale fluctuations or noise since most of the tree will be traversed. Another example of an index-accelerated isosurfacing algorithm is NOISE [15], which makes use of a k-d tree [4] to accelerate the search. NOISE searches over points in 2D rather than a full 3D interval search. ISSUE [18] further improves on NOISE by using a 2D regular lattice rather than a k-d tree in the search phase.

The isosurfacing algorithms described above rely on index data structures that remain entirely memory-resident to speed up isosurface extraction. By contrast, FastBit can operate almost entirely out-of-core, thereby minimizing the memory footprint when used on extremely large datasets that otherwise would not fit into main memory. An interactive isosurface extraction method based on an out-of-core, i.e. non-memory resident, index data structure, is presented in [7].

The isosurface algorithms found in typical visualization frameworks operate only on scalar data values. They do not directly support multi-dimensional feature-based searches for interactive refinement of feature values such as $temperature$ or $pressure$. As described earlier, locating cells that satisfy a multi-dimensional feature query using traditional visualization tools requires generating a derived field, which is akin to an an expensive join operation. DEX, however, evaluates multi-dimensional feature bitmap index queries, which do not require expensive joins.

In DEX, isosurface extraction is only one of several supported features. Apart from computing scalar equipotential surfaces, we are mainly interested in reconstructing the whole data volume of the extracted regions. Typically this data volume is further processed in scientific applications such as analysis of flame fronts [24].

3 Index-Based Data Extraction

Bitmap indices have been successfully applied in large-scale scientific analysis. Recently we demonstrated that bitmap indices can also be applied efficiently for 2D and 3D region growing problems [24, 19]. In this section we will review some of these fundamental assumptions and ideas that are important for understanding the complete visualization pipeline that we introduce in Section 4.

Many scientific datasets are spatio-temporal in nature because they compute or are measurements of physical quantities that vary in space and time. For example, a simulation of the combustion process computes the concentrations of all chemical species along with pressure and temperature [9, 12]. One common operation in mining these datasets is to derive quantities on regions of interest, for instance, the total heat output from an ignition kernel in the combustion simulation. Computing derived quantities, which can itself be an expensive operation, is accelerated by efficiently identify regions of interest (feature extraction).

Our assumption is that the datasets are based on regular discretization of space as used in the Direct Numerical Simulations of combustion on uniform 2D or 3D meshes [9, 12]. In these cases, the space is discretized into small cells according to the raster scan order [24]. The quantities on each cell are computed at varying time values and are grouped by time steps.

After the user specifies the selection critera, the process of identifying regions of interest is usually divided into two steps. The *feature extraction step* (search step) locates cells that satisfy the search criteria. The *region growing step* groups the selected cells into connected regions.

In order to identify regions of interest, data structures like the Quad-tree and R-Tree [10] partition cells according to their spatial coordinates. Apart from the well-known fact that these data structure are efficient only for relatively low dimensional data, they also separate cells that are neighbors in space. As a result of the spatial separation, the efficiency of region growing algorithm is often impeded [24].

DEX preserves the spatial order of the cells, thus avoiding the need for base data reordering and reducing the time required to build the bitmap indices. Another benefit is that the compressed bitmap, which is produced as the result of the feature extraction step, can be easily turned into blocks of connected cells.

4 DEX - Dexterous Data Explorer

4.1 Design and Functionality

The DEX tool combines the FastBit query engine with 3D visualization methods. The result is the ability to perform interactive feature-based analysis and region finding for high-dimensional queries. By displaying the resulting regions of interest, application scientists can quickly identify characteristic features of their data. We refer to this approach to visualization as *query-driven visual data analysis.* Query-driven data analysis methods allow a scientist to define a search criteria as a Boolean expression. The search only returns the subset of data that matches the search criteria, thereby reducing load on the downstreaming visual-

ization pipeline as well as reducing the "visual load" on the scientist. Visualization processing and visual interpretation is limited only to data defined to be relevant. The scope-limiting afforded by query-driving visual data analysis represents a leap forward in capabilities for scientific researchers.

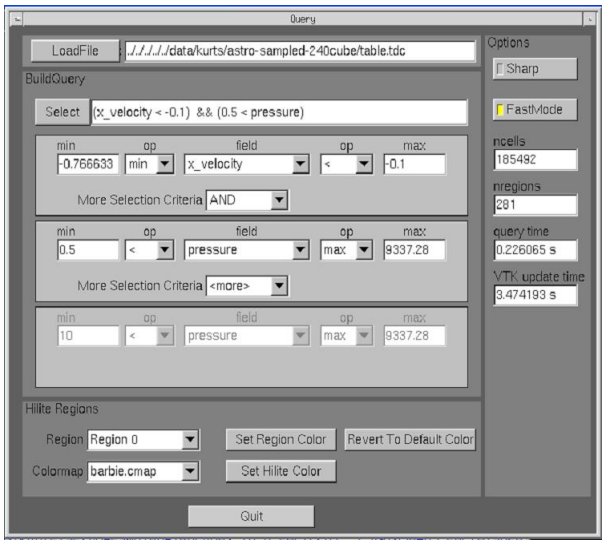


Figure 2. Graphical User Interface of DEX. 2-dimensional query on supernova data.

The version of DEX we describe in this paper uses the Fast Light Toolkit (FLTK) for the graphical user interface (GUI), the Visualization ToolKit (VTK) for visualization processing, and OpenGL for hardware accelerated 3D rendering. The graphical user interface, shown in Figure 2, demonstrates a two-dimensional query for extracting regions of interest. The result of this query selection is then visualized as shown in Figure 3.

The DEX user interface, seen in Figure 2, lays out the typical pipeline for a data analysis task in a top-to-bottom flow. The top portion of the user interface allows the user to select a dataset by either typing a filename or using the file browser dialog. The middle section of the GUI guides the user through the process of building a complex query, which is displayed just below the file selection dialog as it is being constructed. Finally, the bottom section of the GUI supports various ways of manipulating the regions of interest identified by the region growing algorithm. Future versions of the tool will add controls on the bottom portion of the interface that control a broader range of visualization techniques that could be applied to the cells returned from the search. These techniques include but are not limited to slicing, transfer function manipulation, thresholding, vector/tensor visualization algorithmic control, etc.

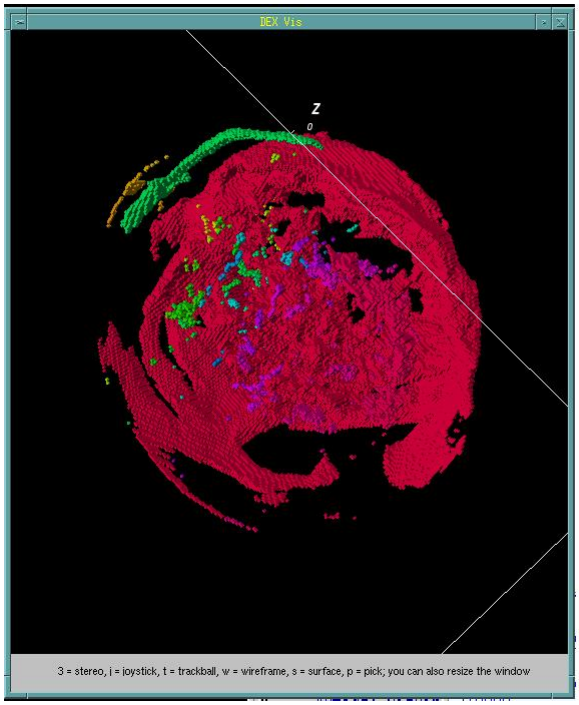


Figure 3. 3D Visualization of supernova explosion based on user query of Figure 2.

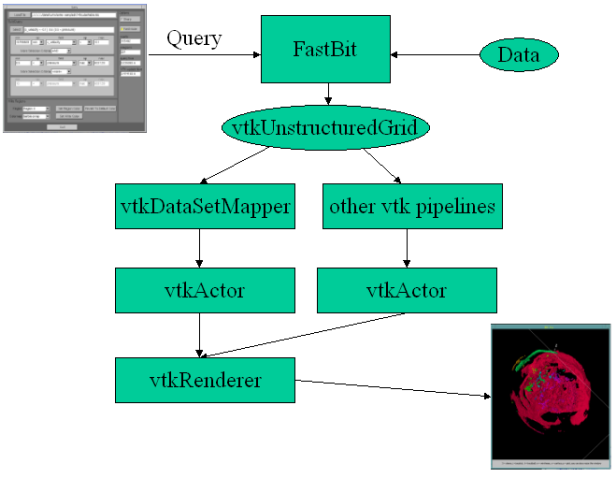


Figure 4. A typical visualization pipeline in VTK.

The majority of the code in DEX translates the query results into a dataset that can be processed efficiently using the VTK pipeline shown in Figure 4. FastBit operates on uni-

form structured grids like those stored in HDF or NetCDF format. However, the data returned by a selection is expressed as a list of disjoint rectilinear regions that match the selection criteria. Consequently, the selected data is encoded as an unstructured grid comprised of hexahedral cells. The resulting unstructured dataset is assembled in memory as a vtkUnstructuredGrid dataset and handed off to a standard VTK pipeline suitable for use with unstructured grids (see Figure 4). The user can then interactively view the geometric model resulting from the selection in 3D using the hardware accelerated graphics.

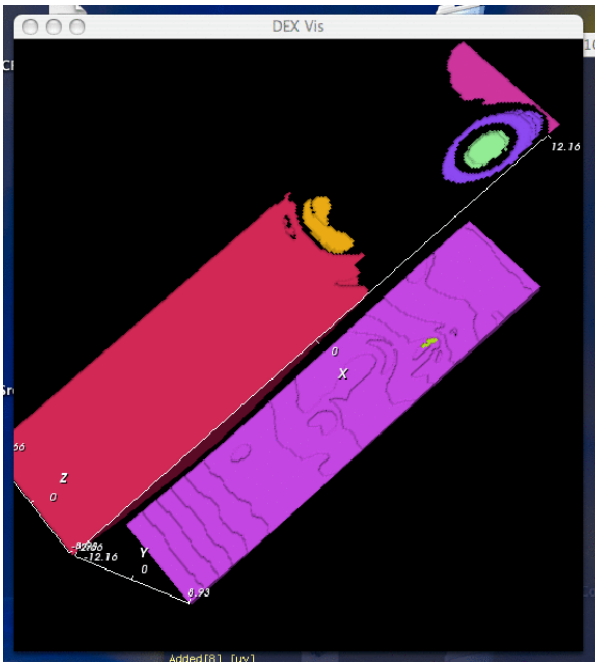


Figure 5. 3D Visualization of combustion simulation. The image is an example taken from the combustion studies where the goal is to track the ignition kernel of a flame.

The FastBit query engine can also rapidly find connected regions of cells using region growing algorithms and assign to each a unique region label. The connected regions are identified by assigning each a distinct color as shown in Figure 5. However, the rendering of a depiction of the cells is but a very limited example of the visualization methods that are possible using a visualization pipeline. The unstructured cell data generated by the query is amenable to the full complement of visualization algorithms available within VTK and other visualization tools. Future variations of DEX will support a broader range of visualization techniques as well as coupling with other visualization pipelines.

4.2 Time Complexity Analysis

To set the stage for our experiments and the results, we present an informal analysis of comparing DEX's time complexity with that of traditional isocontouring algorithms within the context and contraints set forth in this paper. Specifically, we are comparing the performance of isosurface extraction with DEX to that of well-understood isocontouring algorithms. Using DEX, the isosurface can be extracted by specifying a simple query, e.g., $temperature > C$. In this case, the boundary of the region of interest is the isosurface for $temperature = C$.

For the purpose of this discussion, we consider three algorithms: Marching Cubes, NOISE and ISSUE. We also disregard the memory requirements and time complexity for preprocessing of all algorithms. The time complexity of Marching Cubes is $O(n)$, where n is the total number of cells in the dataset. Since Marching Cubes does not attempt to use any strategy to accelerate locating cells that intersect the surface, all grid cells must be examined. In contrast, the NOISE algorithm uses a spanning tree to accelerate the search process, and was shown to have a worst-case time complexity of $O(\sqrt{n}+s)$, where n is the number of cells in the grid and s is the number of cells intersecting the surface in [15]. ISSUE has a time complexity of $O(log(n/L) + \sqrt{n}/L + s)$, where L is a tunable parameter [18]. Both NOISE and ISSUE are considered *nearly* optimal because the optimal algorithm should have a complexity of $O(s)$.

In the case of DEX, the steps required to generate an isosurface are: querying, region growing and surface computation. Again, we are disregarding the cost of storage and preprocessing. The querying step uses bitmap indices to identify those cells that satisfying the specified conditions. The complexity of this step is linear in the number of cells selected [25]. On data from regular grids, the compression scheme used in FastBit actually groups consecutive cells into *cell blocks*. This reduces the time complexity to be proportional to the number of cell blocks involved [24]. In most cases, each of these blocks has two cells that touch the isosurface, therefore, the time required for querying is nearly proportional to the number of cells intersecting the isosurface, $O(s)$. In theory there are lower order terms in the time complexity of this step, however, it is shown to be negligible in practice [25].

The time required by the region growing step is also proportional to the number of blocks [19, 24]. The main reason for this is that we work with cell blocks in the region growing algorithms. The region growing algorithms requires only one scan of the cell blocks [24]. During the scan each cell block is visited a small number of times. On the average, the number of cell blocks is proportional to the number of cells intersecting the isosurface. The time required for

region growing is proportional to the number of cells intersecting the isosurface. The region growing step hands the cell blocks to the VTK functions that actually prepare to display the surface. The time required by this step is also proportional to the number of cells intersecting the surface.

Overall, the total time required to extract an isosurface with DEX is nearly proportional to the number of cells intersecting the isosurface. This complexity is same as the best isosurface extraction algorithms [15, 18]. To verify this analysis, in Section 6, we will present some timing results to compare DEX against the best isosurface extraction algorithm available to us.

5 Use Cases

In this section we discuss two use cases where DEX is employed to produce query-driven visual data analysis. These two uses cases – one from Combustion and one from Astrophysics – form the basis for our experiments, which are described in the next section.

5.1 Combustion

Combustion research involves tracking numerous species of molecules through complex chemical reaction networks. Tracking the flame front helps researchers better understand the properties required for efficient combustion. However, the definition of the flame front is ambiguous in practice – it is identified by a complex set of criteria. DEX uses the FastBit infrastructure to rapidly select the data satisfying a set of user-specified conditions believed to characterize the flame front, then performs visual analysis on the resulting data. Each distinct, fully grown region representing a flame front is labeled with color to help the researcher visually identify and track these features over multiple time-steps of the dataset.

5.2 Astrophysics

Like the combustion research example, astrophysics simulations produce data that have many fields at each grid point. Simulations of stellar phenomena like supernovae require tracking the mass fractions of many different chemical species, radiation emission and absorption profiles for radiation transport, baryonic densities, along with typical fluid dynamics properties (e.g. pressure, temperature, flow vectors). DEX helps researchers understand the complex relationships between different fields in the data using query-based exploration methods. The 3D viewing interface provides all of the advantages of typical interactive visualization tool approaches, but the FastBit query mechanism ensures that even large datasets can be explored at interactive rates via the accelerated searches.

6 Experiments

In this section we evaluate the efficiency of DEX with two different scientific datasets. For one-dimensional queries, we compare the performance of DEX with three different isosurface algorithms of VTK. The isosurface algorithms are *vtkMarchingCubes*, *vtkContourFilter* and *vtkKitwareContourFilter*. The experiments were carried out on a 2.8 GHz Intel Pentium IV with 2 GB RAM. The I/O subsystem is a hardware RAID with two SCSI disks. In our tests, we compare the performance of DEX with three different isosurface algorithms provided by VTK.

6.1 Combustion

The first dataset is a temporally evolving reacting methane-air jet from the TeraScale High-Fidelity Simulation of Turbulent Combustion with Detailed Chemistry [21]. The data set consists of some 2.7 million data points with 10 feature values that include chemical species and fluid dynamics variables. For each feature value we built a compressed, range-encoded bitmap index with 100 bins.

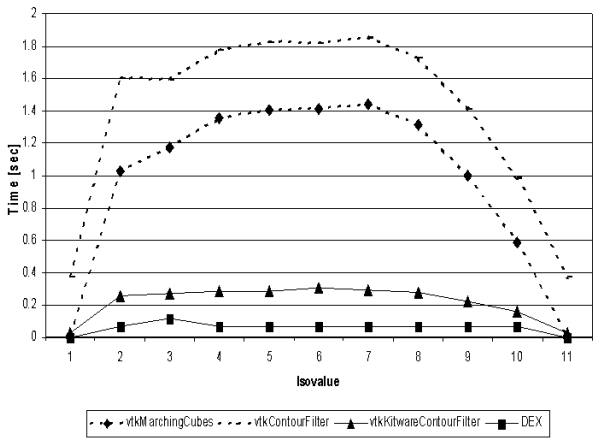


Figure 6. Isosurface extraction for combustion data. DEX vs. three different isosurface algorithms of VTK.

In this set of experiments, we compare the performance of DEX with three different isosurface algorithms of VTK. We measure the time for extracting an isosurface with VTK for the feature value CH_4 with 11 different, randomly chosen isovalues covering the entire domain space. For our experiments we are only interested in the data processing time and do not report on the time for rendering. The results are shown in Figure 6. *vtkContourFilter* is the least performant algorithm, *vtkKitwareContourFilter* is the fastest

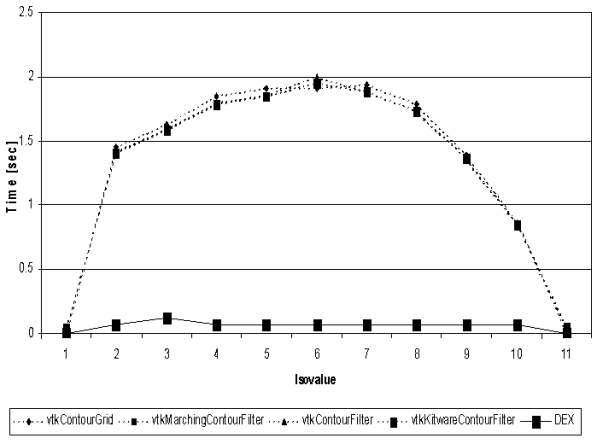


Figure 7. Isosurface extraction for combustion data. DEX vs. four different isosurface algorithms of VTK. Note: We forced the usage of scalarTree for accelerating isosurface extraction in VTK.

VTK algorithm. In all cases DEX performs significantly better than any of VTK's isosurfacing algorithms. On average DEX outperforms the best isosurface algorithm in VTK by a factor of four. Note that DEX not only extracts an isosurface, but also finds the entire volume of cells that lie inside the surface. This is a notable functional difference between DEX and traditional isosurface algorithms.

In the next set of tests we forced the usage of "scalarTree" in VTK. This is an index structure for accelerating isosurface extraction in VTK. As we can see in Figure 7, the acceleration technique did not improve the performance of VTK's algorithm. On the contrary, the performance gain of DEX is even more significant.

6.2 Astrophysics

The second dataset is based on a simulated supernova explosion computed by the TeraScale Supernova Initiative [22]. It consists of a 240^3 mesh, i.e. some 13.8 million data points, with six variables per grid point. They are the *x-velocity*, *y-velocity*, *z-velocity*, *entropy*, *density* and *pressure* of the supernova explosion. For each variable,we built a compressed, range-encoded bitmap index with 100 bins.

As in the previous experiments, we measure the time for extracting an isosurface with VTK for the variable value *x-velocity* using eleven different, randomly chosen isovalues covering the entire domain space. In our experiments, we are interested only in the data processing time and do not report on the time for rendering. The results for the three different isosurface algorithms of VTK are shown in Figure 8. We can see that in all cases DEX is significantly faster than the best VTK isosurface algorithm. On average, DEX's isosurface extraction is three times faster than the best VTK isosurface algorithm.

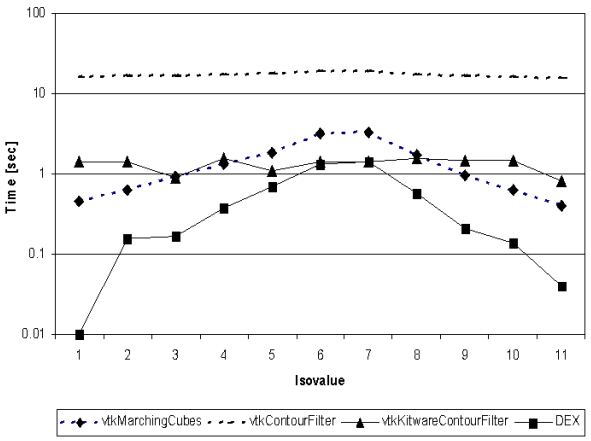


Figure 8. Isosurface extraction for supernova data. DEX vs. three different isosurface algorithms of VTK.

We also ran the benchmarks using the "scalarTree" option in VTK. Similar to the results we obtained from the combustion data, the performance gain of DEX is even more significant (see Figure 9).

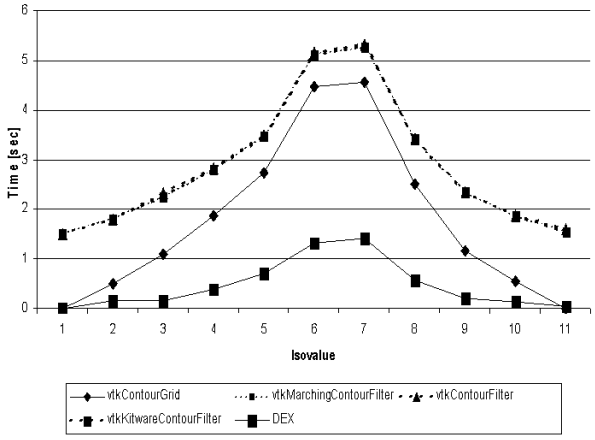


Figure 9. Isosurface extraction for supernova data. DEX vs. four different isosurface algorithms of VTK. Note: We forced the usage of scalarTree for accelerating isosurface extraction in VTK.

The major advantage of DEX over traditional visualization frameworks is that it also supports multi-dimensional feature-based queries. This is a novel research area that is not supported in previous visualization frameworks.

6.3 Observations

Our tests show that DEX outperforms the isosurface extraction algorithms available from VTK. The scalarTree used by VTK builds a spanning tree to accelerate the isosurface extraction algorithms. This is similar to the approach used by DEX where compressed bitmap indices are used. The spanning trees and bitmap indices all reduce the volume of data used during isosurface extraction. Testing results suggest that our bitmap index scheme is more effective since it uses less time. Because the test data are from regular meshes, DEX directly takes advantage of this fact, while the particular version of spanning tree used by VTK was designed for irregular meshes. This may explain why the VTK functions require more time in most cases.

7 Conclusions and Future Work

In this paper we presented the architecture of DEX (short for dexterous data explorer). We showed that DEX combines highly efficient data management techniques with traditional visualization pipelines to produce a new capability we refer to as "query-driven data analysis". Bitmap indices are used to quickly locate features in data and grow them into connected regions. The results are then used as input to the visualization pipeline.

We compared the performance of DEX to traditional isosurfacing, which is a common visualization task. We showed that our approach outperforms the fastest isosurface algorithm in VTK by, on the average, a factor of four when considering the time required to locate cells that satisfy a search criteria. While traditional isosurface algorithms find cells that meet a single criteria – where a surface passes through a cell – our approach supports complex multi-dimensional queries. The main advantage of DEX, however, is that it combines multi-dimensional feature extraction queries with 3D visualization. This new capability, which is not supported in traditional visualization frameworks, allows scientists to get a better visual understanding of the analysis results and has to potential to open the door for new science. It reduces the processing load in the visualization pipeline by limiting processing to data that is "scientifically interesting," and as such is a new approach for visual analysis of large and complex scientific data.

To build on the results we present in this paper, we suggest the following as logical next steps for future research.

- Expand the capabilities of DEX to support queries on adaptive mesh refinement (AMR) [5]) data. Many computational science projects make use of Berger-Colella hierarchical adaptive mesh algorithms, but these data structures pose unique challenges for FastBit methods because data values on refined grids overlap those on the coarser grids. A feature that is apparent in the refined grid may not meet the selection criteria in the coarser grid – or vice verse.

- Provide direct support for multi-resolution data queries. Currently, a query may return selections that are larger than the available memory of a workstation. A multi-resolution query operation will use a low-resolution query to estimate the size of the selection and use that information to select an intermediate level of resolution for the query. This can be keyed off of the current viewing angle of the dataset so that the queries only return with a Level-of-Detail that is warranted by the current viewing angle and screen resolution so that no features that are smaller than a single pixel in screen space need to be returned. This can also be extended to support view-culled queries where portions of the selection that would otherwise be outside of the viewable screen area will be excluded from the query.

8 Acknowledgments

We thank Jacqueline Chen (Sandia National Laboratories, California) and John Blondin (North Carolina State University) for the scientific datasets used in this paper. This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

References

[1] S. Amer-Yahia and T. Johnson. Optimizing Queries on Compressed Bitmaps. In *International Conference on Very Large Data Bases*, Cairo, Egypt, September 2000. Morgan Kaufmann.

[2] G. Antoshenkov. Byte-aligned Bitmap Compression. Technical report, Oracle Corp., 1994. U.S. Patent number 5,363,098.

[3] G. Antoshenkov and M. Ziauddin. Query Processing and Optimization in ORACLE RDB. *VLDB Journal*, 5:229–237, 1996.

[4] J. L. Bentley. Multidimensional binary search trees used for associative search. *Comm. ACM*, 18(9):509–516, 1975.

[5] M. Berger and P. Colella. Local Adaptive Mesh Refinement for Shock Hydrodynamics. *Journal of Computational Physics*, 82:64–84, May 1989.

[6] C. Y. Chan and Y. E. Ioannidis. An Efficient Bitmap Encoding Scheme for Selection Queries. In *SIGMOD*, Philadelphia, Pennsylvania, USA, June 1999. ACM Press.

[7] Y.-J. Chiang, C. T. Silva, and W. J. Schroeder. Interactive Out-Of-Core Isosurface Extraction. In *IEEE Visualization*, Research Triangle Park, North Carolina, USA, October 1998. IEEE Computer Society Press.

[8] C. S. Co, B. Hamann, and K. I. Joy. Iso-Splatting: A Point-based Alternative to Isosurface Visualization. In *Proceedings of the Eleventh Pacific Conference on Computer Graphics and Applications - Pacific Graphics 2003*, Canmore, Alberta, Canada, October 2003.

[9] T. Echekki and J. H. Chen. Direct Numerical Simulation of Autoignition in non-Homogeneous Hydrogen-Air Mixtures, 2003. Combustion and Flame.

[10] V. Gaede and O. Günther. Multidimension Access Methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[11] G. T. Herman and H. K. Lun. Three-Dimensional Display of Human Organs from Computed Tomograms. *Computer Graphics and Image Processing*, 9(121), 1979.

[12] H. G. Im, J. H. Chen, and C. K. Law. Ignition of Hydrogen/Air Mixing Layer in Turbulent Flows. In *27th International Symposium on Combustion, The Combustion Institute*, pages 1047–1056, Boulder, CO, 1998.

[13] T. Johnson. Performance Measurements of Compressed Bitmap Indices. In *International Conference on Very Large Data Bases*, Edinburgh, Scotland, September 1999. Morgan Kaufmann.

[14] D. A. Keim and H.-P. Kriegel. VisDB: Database Exploration using Multidimensional Visualizat ion. *IEEE Computer Graphics and Applications*, 14(5):40–49, 1994.

[15] Y. Livnat, H. W. Shen, and C. R. Johnson. A Near Optimal Isosurface Extraction Algorithm Using the Span Space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1), March 1996.

[16] W. E. Lorensen and H. E. Cline. Maching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21(4):163–169, 1987.

[17] P. O'Neil. Model 204 Architecture and Performance. In *2nd International Workshop in High Performance Transaction Systems*, Asilomar, California, USA, 1987. Springer-Verlag.

[18] H. W. Shen, C. D. Hansen, Y. Livnat, and C. R. Johnson. Isosurfacing in Span Space with Utmost Efficiency (ISSUE). In *IEEE Visualization '96*, San Francisco, CA, USA, October 1996.

[19] K. Stockinger and K. Wu. Improved Searching for Spatial Features in Spatio-Temporal Data. Technical report, Lawrence Berkeley National Laboratory, Berkeley, California, USA, September 2004. LBNL-56376.

[20] K. Stockinger, K. Wu, and A. Shoshani. Evaluation Strategies for Bitmap Indices with Binning. In *International Conference on Database and Expert Systems Applications (DEXA)*, Zaragoza, Spain, September 2004. Springer-Verlag.

[21] TeraScale High-Fidelity Simulation of Turbulent Combustion with Detailed Chemistry. http://www.scidac.psc.edu.

[22] TeraScale Supernova Initiative. http://www.phy.ornl.gov/tsi/.

[23] J. Wilhelms and A. Van Gelder. Octrees for Faster Isosurface Generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.

[24] K. Wu, W. Koegler, J. Chen, and A. Shoshani. Using bitmap index for interactive exploration of large datasets. In *International Conference on Scientific and Statistical Database Management (SSDBM)*, Cambridge, Massachusetts, USA, July 2003. IEEE Computer Society Press.

[25] K. Wu, E. J. Otoo, and A. Shoshani. On the Performance of Bitmap Indices for High Cardinality Attributes. In *International Conference on Very Large Data Bases*, Toronto, Canada, September 2004. Morgan Kaufmann.

[26] K. Wu, W.-M. Zhang, V. Perevoztchikov, J. Lauret, and A. Shoshani. The Grid Collector: Using an Event Catalog to Speedup User Analysis in Distributed Environment. In *Computing in High Energy and Nuclear Physics (CHEP) 2004*, Interlaken, Switzerland, September 2004.

Iteration Aware Prefetching for Large Multidimensional Scientific Datasets

Philip J. Rhodes
Dept. of Computer Science
University of Mississippi
rhodes@cs.olemiss.edu

Xuan Tang
EMC Corporation
tang_xuan@emc.com

R. Daniel Bergeron
Dept. of Computer Science
University of New Hampshire
rdb@cs.unh.edu

Ted M. Sparr
Dept. of Computer Science
University of New Hampshire
tms@cs.unh.edu

Abstract

Most caching and prefetching research does not take advantage of prior knowledge of access patterns, or does not adequately address the storage issues associated with multidimensional scientific data. Armed with an access pattern specified at run time as an iteration over a multidimensional array stored as a disk file, we use prefetching to greatly reduce the number of disk accesses and mitigate the cost of read latency. We call this iteration aware prefetching.

We assume the pattern of access is not known until runtime, in contrast to chunking methods that preprocess a file for a particular access pattern. Our approach results in dramatic performance improvements over file system caching. We also significantly outperform chunking without having to reorganize the data, and can do even better by applying our approach on top of a chunked file.

1 Introduction

Scientists often work with data represented in an *n*-dimensional space in which data values are associated with a location in the space [Cigno97, Hibbard95]. For example, satellite data is typically considered to be organized in a two dimensional space, while medical CT and MRI data usually exists in a three dimensional space. We consider these kinds of scientific data to be *multidimensional*. Multidimensional data presents special challenges when designing efficient access methods because elements that are nearby in the data space may not be nearby in the underlying data file. The caching and prefetching schemes present in most operating systems do not take into account the natural spatial relationships in the data, so they tend to cache, discard, or prefetch the wrong information.

Over the last fifteen years there has been a thousandfold increase in processor speed, along with even larger gains in memory and disk capacity. During the same period, the size of scientific data sets increased even into the terabyte range. However, the average seek time of hard disk drives has improved only modestly over the same period [Coughlin, Chang01]. The work described here is motivated by the need to minimize the now comparatively high latency or *stalling* costs associated with modern disk drive media. Using our system, a researcher can take advantage of improved I/O performance without spending time on the minutiae of efficient file access.

To implement this abstraction while still maintaining efficiency, the researcher must be able to define the application's data access pattern. We are developing a toolkit of *iterators* that succinctly describe the access pattern and also perform the iteration through the data space. Using knowledge of the access pattern, we can create a cache and a prefetching strategy that usually provides significant speedup for the application.

A unique aspect of our approach is that we create and prefetch cache blocks with *n*-dimensional shape, as opposed to the 1 dimensional pages of file system caches and similar methods. *N*-dimensional cache blocks can be given a shape that is tuned to a particular iteration and to the storage organization of the data. We choose a shape that minimizes the total number of disk accesses while reading data that is sure to be visited in the near future by the iteration. We call this method *spatial prefetching*, an example of *iteration aware prefetching*.

Unlike other methods for achieving efficient I/O performance [Sarawagi94, More00], our approach does not require any reorganization of the data. That is, we work with the original data file, rather than making a copy with a different storage organization.

The work described here is done in the context of the *datasource* component of the Granite Scientific Database System, which is in turn an implementation of our multisource multiresolution data model for scientific data [Rhodes01]. The datasource layer handles multidimensional data in which sample points are arranged in a regular and rectilinear fashion throughout the domain. As with many other scientific databases, the design of the Granite system assumes that *update* operations are infrequent or entirely absent, so the work described here is aimed toward a read-only data environment.

After a brief overview of related work, the next several sections describe the functionality and implementation of the datasource, iterator and cache classes, all of which contribute to the support of transparent and efficient out-

This work is supported by the National Science Foundation under grants IIS-0082577 and IIS-9871859

of-core access. We then present performance test results that demonstrate the significant advantages of this approach. Finally, we end with future work and conclusions.

2 Related Research

Providing efficient access to huge scientific datasets is a challenging problem, and has attracted a lot of attention from both the operating system and scientific data management communities. Work has focused on either providing comprehensive scientific data management systems, or optimizing file systems using techniques like prefetching, caching and parallel I/O.

2.1 File Access

Reorganizing datasets on disk to speed access has been explored by a number of researchers. Sarawagi and Stonebraker [Sarawagi94] describe *chunking*, which uses the expected access pattern to group spatially adjacent data elements into *n*-dimensional chunks which are then used as a basic I/O unit, making access to multidimensional data an order of magnitude faster. They also arrange the storage order of these chunks to minimize seek distance during access. Following this work, many other reorganization methods have been developed. More and Choudary [More00] reorganize their data according to the expected query type, and the likelihood that data values will be accessed together. The Active Data Repository (ADR) uses chunking to reduce overall access costs and to achieve balanced parallel I/O [CChang00, CChangADR].

2.2 Prefetching and Caching

Software prefetching has been used by many researchers to hide or minimize the cost of I/O stalling. In the file systems arena, approaches to this problem can be distinguished by whether or not prefetching is guided by explicit information about the access pattern. Albers *et al*. [Albers98] describe an algorithm that produces an optimal schedule for prefetching and discarding cache blocks when the entire access pattern is given in advance. Other researchers have explored the case where the access pattern is disclosed less completely in the form of *hints*. Patterson *et al*. [Patterson95] developed a framework for informed caching and prefetching based on a cost-benefit model. This model has been extended to account for storage devices with very different performance characteristics [Forney02]. Cao *et al*. demonstrate success by letting applications have control of data cache replacement strategy in their share of cache blocks [Cao96]. Brown *et al*. [Brown01] describe a hint based method that effectively accelerates paged virtual memory performance using an operating system that takes advantage of compiler generated hints and multiple disks. Kotz [Kotz97] describes *disk directed I/O*, a method for aggregating and prefetching data requests in a parallel environment. Mowry [Mowry94] presents software controlled prefetching for hiding or reducing the latency experienced by a processor accessing memory.

When no explicit information about access pattern is available, the history of prior accesses can be used to predict future accesses. Amer *et al*. group files together based on historical file access patterns [Amer02]. Other researchers have used probability trees or graphs to represent the likelihood of future block accesses given past and current block accesses [Vellanki99, Highley03, Griffioen94]. Madhyastha *et al*. use a hidden Markov model to automatically predict file access patterns over time; the file system adaptively selects appropriate caching and prefetching policies according to the detected pattern [Madhyastha96, Madhyastha97].

At the application level, Chang [Chang01] adds a separate thread to the user program that performs prefetching by mimicking the I/O behavior of the main thread and preloading data. Doshi [Doshi03] describes a system that adaptively selects a prefetching strategy based on user behavior. The VisTools [Nadeau] system is most similar to our approach. It provides an application level data prefetching and caching service for huge multidimensional datasets, using the Paged-Array schema. It reads formatted pages of elements from the underlying files when the first element in the page is requested. The formatted pages are then stored in a *page cache* for fast future re-access. When the cache size limit is reached, the paged-arrays are deleted or written to a swap file. Like our own work, paged-arrays also support intelligent prefetching guided by the iterators that have an *n*-dimensional view of the dataset. However, the one dimensional nature of pages fails to take into account the proximity of elements in *n*-dimensional space. By using pages as its unit of cache storage, VisTools and other page based methods may make poor decisions about what data to retain or discard. The following section examines this issue in greater detail.

2.3 Advantages of the Granite Approach

Reorganizing data into chunks is a very effective and general technique, but the required reorganization (and implied duplication) of the dataset can be inconvenient, especially when working with large datasets. Also, performance may suffer if the data is accessed in a different way than was expected when the reorganization was performed. The approach adopted by the Granite system works with the original data, and requires no such reorganization.

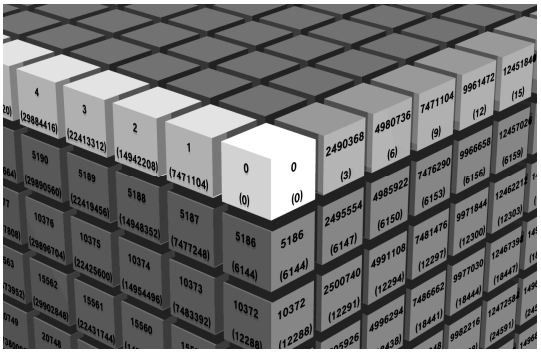


Figure 1. The elements fetched by the file system (medium gray, on the right) are not the elements needed by the iteration (light gray, on the left). The top number in each block indicates iteration order, while the numbers in parentheses indicate the offset in the file.

Systems that access the data in pages suffer from not taking into account the multidimensional nature of the data. In particular, elements that are nearby in *n*-dimensional space may be far apart in the one dimensional file space. Since paging is essentially a one dimensional method, it may be inefficient for an *n*-dimensional access pattern.

Figure 1 shows a conceptual view of a portion of the 39GB Visible Woman dataset, provided by the National Institutes of Health. This dataset consists of 5186x2048x1216 elements of 3 bytes apiece [Rhodes05]. Each block in the figure represents a single element. The number in parentheses at the bottom of each block represents the byte offset of that element from the beginning of the file and the number at the top of each block represents the order in which that block will be visited by an iterator. The light gray blocks show the initial path of the iteration, beginning with the white corner element. When this initial element is accessed, the file system will load a page of data, typically 4K in size, indicated by the series of medium gray elements. Unfortunately, the next element to be visited (the light gray block at offset 7471104) is not contained in this page, so the disk must be read again. In fact, separate reads must be made for each element in the light gray series, since they are all separated by over 7MB in the one dimensional file. When, at step 5186, the iteration returns from the far end of the data set to begin a new "row", a new page will still have to be read, since this element is not contained in any of the pages that have been read so far. In fact, the next medium gray element would be used at step 2490368 of the iteration, but this first page will certainly have been discarded at this point, unless the file system is able to retain 10GB of disk pages in memory. This is clearly beyond the capacity of today's commonly available systems.

File systems also commonly prefetch pages following an explicitly accessed page in the hope that the prefetched pages will be accessed next, and reads to disk will be reduced. This helps slightly in this example, because elements that are vertically adjacent are only 6K apart. So, if the file system prefetches at least one additional page for each read to disk, the elements immediately beneath the light gray elements will be read from pages loaded during the traversal of the light gray elements. Although this is an improvement, it still means that only two elements will be read out of each 4K page before it is discarded, only to be reread later in the iteration. For datasets with even larger dimensions, the distance between vertically adjacent elements may be too large for pages to ever be used more than once before being discarded. In this case, prefetching just makes the situation worse by increasing the number of inappropriate pages loaded into memory. Because the file system has no information about the dimensionality of the data or the path of the iteration, it is grossly unsuited for the job.

We address these issues by creating cache blocks that are *n*-dimensional, and shaped according to the iteration. Elements that are contiguous in the file are loaded in a single *read()* call. This method has several beneficial effects. First, it uses more data from each file system page that is read, thereby reducing the number of redundant reads made to disk. Second, it reduces the number of *read()* calls made to the operating system. Third, since the cache block can be filled in any order, we choose to fill it in a way that most closely matches the ordering of the data in the file. This allows us to sometimes take advantage of the file system prefetching that is otherwise a liability.

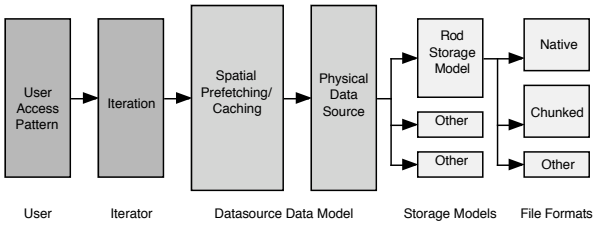


Figure 2. Spatial prefetching and the Datasource data model serves as a bridge between the user access pattern expressed as an iteration, and the data as it lies on disk.

We utilize nearly complete information about the access pattern given by our iterators. We don't have to guess which data to prefetch, and we don't discard needed data before it is used. Because of this, the various caches we have developed require at most two cache blocks to be maintained in memory at a time, which can extend the reach of an application to much larger datasets than would otherwise be possible.

3 The Multidimensional Data Model

Figure 2 is a conceptual diagram of the pipeline relating a user access pattern to the file. The datasource is the representation seen by a Granite user, and uses a *storage model* to help translate the *n*-dimensional data space to the one dimensional file space. The storage model may be able to work with more than one *file format*. For example, the rod storage model discussed later in this section represents both chunked data files and files that have been left in their native plane-row-column order.

3.1 Datasources

We model the data to be processed as a *datasource*, which is conceptually an *n*-dimensional array containing a set of sample points. We call the space defined by the array indices an *index space*. Each location in the index space has a collection of associated data values, which we call a *datum*. Although our datasource model allows datasources to be built on top of other datasources or to be associated with a network stream, we limit our discussion in this paper to datasources that are associated directly with a file on disk.

Datasources must handle two basic kinds of queries. A *datum query* specifies a single location in the index space, and is satisfied by the return of a single datum. A *subblock query* specifies an *n*-dimensional rectangular region of the index space, and is satisfied by the return of a *data block*, which is conceptually an array of datums, with a dimensionality matching the datasource.

3.2 The Rod Storage Model

While the file is a one dimensional entity, a datasource has an index space that is *n*-dimensional. The datasource is responsible for satisfying queries expressed in its index space by reading data from the file. It must therefore map its index space to file offsets. It does this with the help of an *axis ordering,* which is simply a ranking of axes from *outermost* to *innermost*. "Innermost" and "outermost" suggest position in a set of nested *for* loops used to access the file in its storage order on disk. The innermost axis changes most frequently and is called the *rod* axis when referring to the *storage ordering* of a datasource. Each axis is labeled with an integer that identifies the position of coordinates of that axis in a tuple used to specify locations in the index space. Consequently, an axis ordering is just a list of integers that defines an ordering of axis coordinates from least to most frequently varying. For example, an axis ordering of {0,2,1} indicates that coordinates of axis 0 change least frequently, followed by axis 2, and then by axis 1, which changes most frequently.

The number of separate read requests made to the storage device strongly impacts I/O performance, so it is important to minimize the number of reads when satisfying a subblock query. Toward this end, the rod storage model views the datasource as being conceptually composed of *rods*. A rod is a one dimensional sequence of elements that are contiguous in both the *n*-dimensional index space and the 1-dimensional file space. Consequently, rods are always aligned with the rod axis. Because it is contiguous in the file space, a rod can be accessed in one read. Note that rods are composed of datums when the native file format is used, or whole chunks of datums with the chunked file format. When a subblock query is processed, the requested region of index space is decomposed into a collection of rod segments contained in the region. We then retrieve the subblock data from disk in rod-by-rod fashion where each rod segment corresponds to a separate read. To maximize locality, we read this set of rods according to the storage ordering. In the case where a set of rods is itself contiguous (or nearly so) in the file, we issue only one read and retrieve the entire set of rods in one disk operation.

Like the *order line* model described in [Wu03], the rod storage model does not take into account the physical layout of the file on the disk, but only the logical layout presented by the file system. However we have found that this approximation serves as an effective foundation for our iteration aware prefetching, which shows significant performance improvements over other techniques. In addition, applying the rod model to chunked data and other formats extends the reach of iteration aware prefetching to a wider range of data representations.

4 Iterators

Since our system aims to improve I/O performance for particular access patterns, we use iterators to represent access patterns as well as to perform the actual iteration through the datasource index space. Iterators have a value that changes with each call to the iterator's *next()* method. This value might denote a single location in the index space, or perhaps a rectilinear region. In either case, the iterator value can be used directly in datum and subblock queries.

The pattern of iteration is determined when the iterator is constructed. An axis ordering is used to help represent the behavior of iterators that proceed through the index space in rectilinear fashion. In this context, the innermost axis of the iteration is called the *run axis*. While the datasource is conceptually composed of rods, the space being traversed by a rectilinear iterator is conceptually composed of *runs*.

The *iteration space* is the space traversed by the iterator. It may be the entire index space of a datasource, or

some subset of that space. We also represent the starting point and the *stride* through the iteration space in cases where the iterator skips over some locations. The iteration space, starting point, stride and axis ordering all contribute to the creation of a prefetching cache that is tuned to the iteration.

5 Iteration Aware Prefetching

As noted in section 2, much of the literature in caching and prefetching concerns when to load new blocks from disk, and choosing blocks to be discarded. Because we have nearly complete information about the access pattern from the iterator, these problems are vastly simplified in our system. We call our approach *Iteration Aware Prefetching*.

The standard caching and prefetching view of files as one dimensional entities is not adequate for scientific applications involving multidimensional datasets because it misses the neighborhood relationships inherent in the data. The problem becomes even more acute as the dimensionality of the dataset increases. To address this issue we have designed a *multidimensional cache* that preserves the iterator's spatial data view. The iteration space is conceptually partitioned into an *n*-dimensional array of *n*-dimensional cache blocks. Data is read from disk one block at a time, and is retained in memory to quickly satisfy user data requests.

Our iteration aware prefetching approach includes two independent components — *spatial prefetching* and *threaded prefetching*. Their different roles become clear when considering an iterator reading a series of blocks from a datasource. Spatial prefetching reduces the latency costs incurred while reading data from a single block. Threaded prefetching reduces or eliminates the amount of time an application must wait for a complete block to be read by overlapping application processing with I/O.

5.1 Spatial Prefetching

The key contribution of our prefetching strategy is based on adjusting the shape of the cache blocks to minimize the number of separate reads made to disk.

An important characteristic of our approach is that we can perform effective prefetching using a single conceptual cache block. Although there are sometimes practical reasons for breaking a single conceptual cache block into 2 or more physical blocks, the discussion in this section addresses the construction of a single conceptual block that is tuned to the user's access pattern.

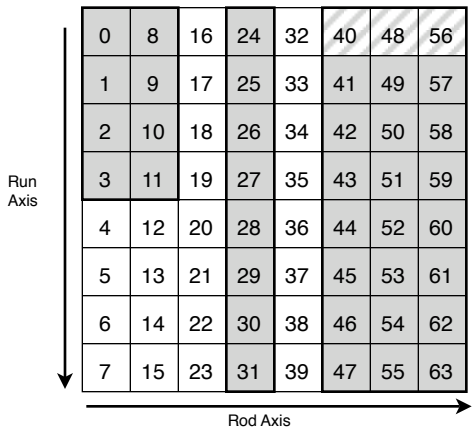


Figure 3. For a {1,0} iteration over a {0,1} datasource, the shape on the right is the only one that is both well formed and practical.

5.1.1 Examples

Figure 3 shows three potential cache block shapes. The numbered sequence indicates a column-by-column iteration over a datasource stored in row-by-row fashion.

Suppose that the shaded shape in the upper left region of the figure were assigned to the cache block. Such an assignment would be poorly suited for a single block cache since step 4 of the iteration causes the block to be discarded, only to be reloaded at step 8. The algorithm shown in figure 4 would extend the block shape all the way down to the bottom of the space before attempting to extend it in the horizontal direction. The cache block shape shown is poorly suited to a single block cache because step 4 of the iteration causes the block to be discarded, only to be reloaded at step 8.

The middle shaded shape in figure 3 does not have this problem, since it extends over the full length of the vertical axis. However, this block cannot reduce the number of read operations. Since the rod axis is the horizontal axis, filling this cache block would require eight separate reads, which is the same number needed with no cache at all.

The shaded shape on the right is much better, since it can be filled with 8 reads of length 3. The striped region represents a single rod subset for this block. Depending on the characteristics of the platform, this shape may produce a useful increase in performance.

5.1.2 Well Formed Cache Blocks

Typically, when a cache needs to load data from disk to satisfy a request, it loads a larger set of data in the neighborhood of the original request. Hopefully, the nearby data can be used to satisfy future requests without returning to the disk. If the pattern of future accesses is already known, however, we can choose a cache block shape that guarantees that all the needed contents will be used before being

Algorithm A1:
Input:
Iterator Ordering $A_n = \{a_0, a_1, a_2, \ldots a_{n-1}\}$,
Iteration space dimensions $S_n = \{s_0, s_1, s_2, \ldots s_{n-1}\}$,
available memory M

Output:
A set of cache block dimensions $B = \{b_0, b_1, b_2, \ldots b_{n-1}\}$ that represent a cache block shape that is well formed with respect to the iterator ordering.

Note:
M(B) indicates the bytes occupied by a cache block of shape *B*

```
begin
 B = {1, 1, 1, …1}

 for i = n-1 downto 0  // from innermost to outermost

   axis = a_i     // for the next innermost axis…
   b_axis = s_axis  // extend B to end of iteration space

   if (M(B) > M) then  // is memory exceeded?
     b_axis = 1              // then return B to previous shape
     b_axis = M / M(B)       // extend B as far as memory allows

    done
   end
 end
end
```

Figure 4. Algorithm A1 produces a cache block shape that is well formed with respect to a given iteration.

discarded. We say such a cache block is *well formed* with respect to the iteration. A more formal definition follows:

Definition D1:
We denote a rectilinear iteration *I* over a rectilinear iteration space *D* using an axis ordering *A* as *I(A,D)*. Consider a rectilinear region *R* of shape *B* that is a subset of *D*. We say the shape *B* is *well formed* with respect to *I(A,D)* if for any region *R* of shape *B* in *D*, once *I* leaves *R*, it will not revisit *R*.

If we can construct a cache containing blocks that are well formed with respect to a given iterator, we can be assured that no cache block will need to be read more than once, and that once the iterator is done with a cache block, we can discard it. Therefore, most iterations only require a single cache block to be used at one time. Overlapping block iterators require at least two cache blocks, as does threaded prefetching.

Algorithm A1 generates a well formed cache block shape for a datum iterator that visits single elements in the index space. It must be given the iterator ordering, the space over which the iterator travels, and the amount of memory that is available for constructing a cache block.

The algorithm works by marching through the iterator's axis ordering from innermost to outermost axis, setting the corresponding dimension of the cache block shape to equal the extent of the iteration region along that axis. Below is a proof that algorithm A1 produces a well formed cache block shape for a datum iterator.

Proof P1:
Claim: Algorithm A1 produces a well formed shape *B* for the given iterator, iteration space, and available memory.

Base Case: An n-dimensional shape with a single element, $B_0 = \{1, 1, 1 \ldots 1\}$ is well-formed with respect to an iterator *I*(*A, D*) using any axis ordering *A*.

Assumption: Algorithm A1 produces a shape at step *k*, B_k, that is well formed with respect to *A*.

Induction step: At step *k+1*, we know that block B_k extends across the entire extent of the iteration space for the *k* least significant axes and that it is well-formed. Algorithm A1 then extends the block along axis *k+1* to either the entire extent of the iteration space in that axis or as much as will fit in the available memory. In both cases the shape is well formed since the iteration will not return to that shape after leaving it. If the algorithm cannot add the entire extent of the iteration space in that axis, it terminates, leaving a well formed block.

The algorithm and proof can be easily modified to account for block iterators rather than datum iterators. Since block iterators represent a sequence of block accesses, we can set the initial dimensions of the cache block shape to match a single iterator block. The algorithm then proceeds as before. The proof still holds for this case if we consider an element to be a block instead of a single position in the index space. The block version of the algorithm can also be used to handle the case where an iterator has gaps or overlap between visited elements.

5.1.3 Practicality

Whether the shape of a cache block is well formed is related only to a particular iteration. It is possible that a well formed cache block will not enhance performance with a certain dataset because of the way the data lies on disk. In order to guard against this possibility, we must check to see if a cache block shape is *practical* with respect to the storage model. We currently only consider the rod storage model, and our definition of practicality concerns the extent of the cache block shape along the rod axis.

Definition D2:
A cache block shape is *practical* with respect to a rod storage model if it has extent greater than *r* elements along the rod axis, where the value of *r* is determined by cache overhead and the performance characteristics of the I/O subsystem, and must be greater than 1.

This definition is motivated by the fact that in order to get any gain in performance, we must reduce the number of reads made to disk. It follows that we must therefore make each read longer than would be performed without the cache. The extent of the cache block shape along the rod axis determines the length of these reads, so this value must be sufficiently long to provide a performance gain, even in the face of cache overhead.

5.2 File Formats

When the rod storage model is used on top of the native file format, the rods consist of a series of datums stored sequentially on disk. We refer to this file format as "native" because it requires no preprocessing — the file is handled "as is". In this situation, using a well formed cache block also guarantees that no data is read from disk more than once. This is because the cache block is defined in terms of the same units (datums) as the file format.

The rod storage model can also be used on top of chunked files. In this case, the rods consist of a series of contiguous chunks that can be loaded with a single read operation. Here, the file format is defined in terms of units different from what was used to define the cache block. Because of this, data may be read more than once, even with well formed cache block shapes. Currently, we solve this problem by ensuring that for each dimension a cache block will either extend through the entire iteration space, or have length equal to one chunk. This ensures that the cache block is well formed with respect to the *n*-dimensional chunked space.

5.3 Threaded Prefetching

Threaded prefetching uses a separate I/O thread to fetch the next cache block while the current one is being processed. Unlike other systems using I/O threads, we don't have to guess which block should be read next, because that information is contained in the iterator. Currently, we have only implemented and tested threaded prefetching for a single disk, so we can achieve at most the doubling of performance that occurs when the I/O time perfectly matches the computation time for each block.

Our current approach is very effective in hiding the cost of loading a block of data from disk, but even greater performance improvements should be possible if multiple disks are available.

When the rate at which an application consumes data is less than or equal to the rate at which data can be read from disk, threaded prefetching can yield performance similar to the in-core case. The combination of threaded prefetching (even with only one I/O stream) and spatial prefetching can be particularly effective in an interactive application. Figure 5 shows an image created with the

Figure 5. A view of the 39GB Visible Woman Dataset produced with *Slicer*, an interactive slice based volume visualizer.

Slicer application described in [Rhodes05]. *Slicer* uses a combination of threaded and spatial prefetching to view progressive slices of a user defined subset of the 39GB Visible Woman dataset. By matching the frame rate to the capabilities of the I/O subsystem, threaded prefetching allows for smooth animation.

```
// Create datasource
Datasource ds = Datasource.createDS("8gig.xml");

// Create ordering for iterator
AxisOrdering
       iterOrdering= new AxisOrdering(new int[]{0, 1, 2});

// Create an iterator that traverses the entire datasource
ISIterator
    iter=new ISIterator(ds.getBounds(), iterOrdering);

// Create a spatial prefetching cache for the
// given datasource and iterator
CacheDataSource
    cds = CacheMaker.createCDS(ds, iter, freeMem);

// Create a datum to receive data values.
Datum d = new Datum(ds.getNumAttributes());

// Traverse the entire datasource index space,
// accessing the data through the cache.
for( iter.init(); iter.valid(); iter.next() )
{
    cds.datum(d,iter); // Process datum
}
```

Figure 6. Example code for a datum iteration over a cache.

6 Example Code

Figure 6 shows a small example of a datum iteration using the Granite system. We first create the datasource from an xml file that describes such properties as dimen-

	Datum Iteration over native file				Datum Iteration over chunked file			64^3 Block Iteration over native file			64^3 Block Iteration over chunked file		
	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**
Ordering	File System Cache	128MB SP Cache	512MB SP Cache	Max Speed up	512MB LRU Cache	512MB SP Cache	Speed up	File System Cache	512MB SP Cache	Speed up	512MB LRU Cache	512MB SP Cache	Speed up
{0, 1, 2}	9963	868	961	11.5	4628	3634	1.27	705	304	2.3	3288	342	9.6
{1, 2, 0}	16786	1311	1294	13.0	9175	3880	2.4	664	279	2.4	3081	342	9.0
{2, 1, 0}	360000 (est)	11349	3719	96.8 (est)	9607	4485	2.14	7847	2777	2.8	3736	966	3.7

Table 1. Results for a complete traversal of an 8GB file of 1024x1024x2048 floats. Native files are in plane-row-column order, while chunked files consist of 4K chunks. All execution times are in seconds.

sionality, size along each axis, and the number of attributes at each location in the index space. Next, we define an axis ordering and iterator that will traverse the datasource. We are now able to create a cache that is tuned to the iteration we wish to use. Finally, we create a datum object for retrieving data values and perform the iteration.

This code is very flexible, and requires very minimal changes in order to work with different datasources and iterator orderings. To make the code work on another file of entirely different size and shape, we only need to change the name of the *xml* file given in the first line of code. The iteration order is just as easily changed, and an appropriate cache will be created without further thought from the programmer.

This flexibility is especially attractive in situations where a user wants to process a large file using several different traversals. With spatial prefetching, it is a simple matter to create caches that are tuned to each iteration. With preprocessing methods, some compromise must be made when deciding the chunked format, unless the user is willing to make a separate file for each iteration.

7 Results

We have run our tests on a variety of machines and found that machines with fast I/O show smaller performance improvement simply because the I/O is a smaller portion of the total execution time.

We present results from the machine with the fastest I/O available to us. This is a single processor Pentium 4 machine with a 2.4Ghz CPU and 2GB of RAM running the Linux operating system. The disk on this machine is a fast 15,000 RPM SCSI disk with a 3.6ms average read seek time. Though we show here very substantial gains in performance, we saw even greater gains on other platforms, since a fast disk actually minimizes the benefits of spatial prefetching.

The Linux file system cache loads and stores 4k blocks of data from disk whenever a file is accessed. Since the file system cache is persistent across task execution, it is possible for a task to request an I/O block for the first time, but still get a cache hit if another task had previously read that block. Although this is generally a good thing, it is problematic for our testing environment. We therefore ran all tests with a cold (i.e., empty) file system cache. In addition to guaranteeing a consistent environment by always starting with an empty cache, this approach more realistically portrays the behavior that a researcher might expect when dealing with very large datasets.

In the following sections, we present results for both datum and block iteration over the entirety of a three dimensional 8GB dataset. ([Rhodes05] examines subset iteration in an interactive context.) On our test machine, running the unix *cp* command with this dataset takes approximately 400 seconds. The dataset has dimensions 1024x1024x2048, where each datum is a single floating point value. Tests were run on both native and chunked file formats. In all cases, the files had a storage ordering of {0,1,2}.

Table 1 shows our results for three different iterator orderings over both native and chunked file formats. Both datum and block access were tested. We have performed extensive testing with a wide range of machine characteristics, file sizes, and cache sizes. For clarity and simplicity, we present results here for a single 8GB data set on one machine configuration and we concentrate on a cache size of 512MB. Considering the current affordability of memory and the recent introduction of commodity 64 bit machines, we feel 512MB is a reasonable memory cost for working with very large data sets. However we still see significant performance improvements for smaller cache sizes.

7.1 Datum Iteration over Native Files

Our datum iteration tests ran code very similar to the example in section 6. Columns *a* through *d* of table 1 show the execution times for traversals using the file sys-

tem cache and spatial prefetching (SP) caches of 128MB and 512MB. In all three iterator orderings, the SP cache provides a very substantial improvement in performance. Notice that the {0,1,2} ordering shows somewhat less improvement than the other orderings. This is because the file system is prefetching blocks in the same order that the iterator will request them. File system prefetching is much less effective for the other orderings, so our spatial prefetching offers more improvement in these cases. In fact, the file system cache test for {2,1,0} ordering did not complete within twelve hours. We determined that the test was making forward progress in a linear fashion, but very slowly, due to the awkward nature of this access pattern. A very simple C program that mimicked the access pattern for this test but performed no type conversion or copying of data took over 37 hours to run, so we are confident that disk access is causing the excessive runtime. Using a simple extrapolation, we estimated the completion time for the Java implementation using the file system cache to be about 100 hours, and we report this estimated value in the table.

The test with a 512MB Spatial Prefetching cache does considerably better in the {2,1,0} direction than the 128MB cache. For this ordering, the rods span the shortest dimension of the cache block, so increasing the available memory increases the length of the rods, meaning more data is read with each read operation.

Clearly, it would be beneficial to develop an automatic means of choosing how much memory to allocate to a cache based on storage ordering, iterator ordering, system characteristics, and total memory available. We plan to extend the notion of practicality to support this functionality in future work.

7.2 Datum Iteration over Chunked Files

Chunking is a common method for speeding access to spatial data, so it is important to compare spatial prefetching alone with the performance of chunked file access. The chunked format typically divides the file into chunks equal to the file system page size. The dimensions of the chunks are chosen to best suit a particular access pattern [Sarawagi94].

An important assumption of our work is that the user access pattern is not known until runtime. A generic chunking method chooses chunk dimensions that are equal or nearly equal in all directions. This method provides a substantial performance improvement for most access patterns without being tailored specifically to a particular one. We therefore chose to compare spatial prefetching with this form of chunking.

Chunking generally requires some kind of cache in order to be effective with datum access, so we implemented a simple LRU cache that holds a collection of chunks. We compared the performance of our spatial prefetching cache against the performance of this LRU cache. In all of these tests, the memory used for both caches was always 512MB, and the file was in chunked format.

Columns *e* through *g* show the execution times for both caches. Comparing LRU performance with the file system datum iteration in column *a*, it is clear that chunking is a very effective technique. However, we get even better performance by applying spatial prefetching on top of chunked files, especially in the last two orderings listed in the table. On machines with larger disk latency, speedup is substantial even in the first case.

Of even greater interest is the fact that the performance of spatial prefetching over a native file presented in column *c* is markedly superior to the performance of the LRU cache over a chunked file shown in column *e*. For each ordering, spatial prefetching produces speedups of 4.8, 7.1, and 2.6 compared with chunking. That such performance can be achieved without preprocessing or duplicating the file makes spatial prefetching a particularly attractive technique.

7.3 Block Iteration over Native Files

Block iteration involves loading successive *n*-dimensional subsets of the data from disk. The rod storage model by itself facilitates this form of access since it reads rods according to the storage ordering, which improves locality. However, spatial prefetching is still able to provide a useful performance increase by reading data for many blocks at one time. Columns *h* through *j* show the execution times for a 64^3 block traversal over the same dataset used in the previous section.

7.4 Block Iteration over Chunked Files

Our fourth group of tests compared the performance of our spatial prefetching cache over a chunked file with the LRU cache on the same file. Columns *k* through *m* show that spatial prefetching over chunked files provides much more meaningful speedup for block access than for datum access. Since datum access involves many more cache lookup operations, it is likely that in this case, cache overhead erodes gains in I/O efficiency.

8 Conclusions and Future Work

Mismatch between iteration and storage patterns is a well-known problem addressed by many systems in an *ad hoc* manner. Generally, these approaches are based on a one-dimensional view of the data and do not provide a convenient application level interface to the prefetching facility. We have developed a comprehensive environment

for seamless integration of the data access pattern and the prefetching mechanism. The future multidimensional access pattern is specified implicitly during construction of an iterator. The iterator, in turn, is used to determine an effective prefetching strategy tuned for the particular combination of file storage order and iteration pattern.

Spatial prefetching can provide a very meaningful performance increase when large data files are accessed in a rectilinear manner. We have shown that performance is superior to generic chunked file access, yet does not require a preprocessing step. Since spatial prefetching can be used "on the fly", it is particularly well suited to situations where the pattern of access is not known until runtime, or when several different patterns will be used on the same file.

The Granite system lets the user take advantage of the efficiencies of spatial prefetching and other iterator aware prefetching methods while abstracting away the details of storage organization.

For future work, we plan to expand our use of iterators to include traversal through a collection of data values of interest to the experimenter. We will also expand our support of the current iterators to include a way to automatically determine an amount of cache memory that provides a good tradeoff between performance and memory use. Lastly, we are exploring the application of our methods to a distributed context. Since network latency can be even more severe than disk latency, we expect promising results in that environment.

9 References

[Albers98] S. Albers, N. Garg and S. Leonardi, Minimizing Stall Time in SIngle and Parallel Disk Systems, *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 454-462, 1998

[Amer02] A. Amer, D. Long, and R. Burns. Group-Based Management of Distributed File Caches. In *Proc. of the 17th International Conference on Distributed Computing Systems,* 2002

[Brown01] A.D. Brown, T.C. Mowry, Compiler-Based I/O Prefetching for Out-of-Core Applications, *ACM Trans. on Computer Systems, Vol. 19, No. 2,* May 2001

[Cao96] P. Cao and E. Felten, Implementation and Performance of Integrated Application-Controlled File Caching, Prefetching, and Disk Scheduling, *ACM Transactions on Computer Systems, vol. 14, No. 4*, 1996

[Chang01] F. Chang, Using Speculative Execution to Automatically Hide I/O Latency, *Ph. D. Dissertation*, Carnegie Mellon University, 2001

[CChang00] C. Chang, T. Kurc, A. Sussman, J. Saltz, Optimizing Retrieval and Processing of Multi-dimensional Scientific Datasets, In *Proc. of the Third Merged IPPS/SPDP Symposiums*. IEEE Computer Society Press, May 2000

[CChangADR] C. Chang, T. Kurc, A. Sussman, J. Saltz, *Active Data Repository Software User Manual*, http://www.cs.umd.edu/projects/hpsl/ResearchAreas/ADR-dist/ADR.htm

[Cigno97] P. Cignoni, C. Montani, E. Puppo, Roberto Scopigno, Multiresolution Representation and Visualization of Volume Data, *IEEE Transactions on Visualization and Computer Graphics, Volume 3, No. 4,* IEEE, Los Alamitos, CA, 1997

[Coughlin] T. Coughlin, High Density Hard Disk Drive Trends in the USA, tech report at http://www.tomcoughlin.com /techpapers.htm

[Doshi03] P.R. Doshi, G.E. Rosario, E.A. Rundensteiner, and M.O. Ward, A Strategy Selection Framework for Adaptive Prefetching in Data Visualization, *Proc. of SSDBM,* 2003

[Forney02] B. C. Forney, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, Storage-Aware Caching: Revisiting Caching for Heterogeneous Storage Systems, In *The First USENIX Conference on File and Storage Technologies (FAST '02)*, 2002.

[Griffioen94] J. Griffioen and R. Appleton, Reducing File System Latency Using A Predictive Approach, *University of Kentucky Technical Report #CS247-94*

[Hibbard95] W. L. Hibbard, D. T. Kao, and A. Wierse, Database Issues for Data Visualization: Scientific Data Modeling, *Database Issues for Data Visualization, Proceedings of the IEEE Visualization '95 Workshop, LNCS 1183,* Springer, Berlin, 1995

[Highley03] T. Highley and P. Reynolds, Marginal Cost-Benefit Analysis for Predictive File Prefetching, *Proc. of the 41st Anuual ACM Southeast Conference (ACMSE 2003)*

[Kotz97] D. Kotz, Disk-Directed I/O for MIMD Multiprocessors, *ACM Trans. on Computer Systems,* Vol. 15, No. 1, 1997

[Madhyastha96] T. M. Madhyastha, C. L. Elford, and D. A, Reed, Optimizing Input/Output Using Adaptive File System Policies, In *Fifth NASA Goddard Conference on Mass Storage Systems and Technologies*, September 1996

[Madhyastha97] T. M. Madhyastha, and D. A. Reed, Input/ Output Access Pattern Classification Using Hidden Markov Models, In *Workshop on Input/Output in Parallel and Distr. Systems*, Nov. 1997

[More00] S. More, A. Choudhary, Tertiary Storage Organization for Large Multidimensional Datasets, *8th NASA Goddard Conference on Mass Storage Systems and Technologies,* 2000

[Mowry94] T. C. Mowry, Tolerating Latency Through Software-Controlled Data Prefetching, *Ph.D. Dissertation*, Stanford University, 1994

[Nadeau] D. R. Nadeau, An Architecture for Large Multi-Dimensional Data Management, *Scalable Visualization Tools White Paper*, http://vistools.npaci.edu/

[Patterson95] R.H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, Informed Prefetching and Caching. In *Proceedings of the 15*th ACM Symposium on Operating Systems Principles, December 1995, PP. 79-95.

[Rhodes01] P.J. Rhodes, R.D. Bergeron, and T.M. Sparr, A Data Model for Distributed Multisource Scientific Data, *Hierarchical and Geometrical Methods in Scientific Visualization*, Springer-Verlag, Heidelberg, 2001

[Rhodes05] P.J. Rhodes, X. Tang, R.D. Bergeron, and T.M. Sparr, "Out of core visualization using Iterator Aware Multidimensional Prefetching", *Conference on Visualization and Data Analysis 2005*

[Sarawagi94] S. Sarawagi, M. Stonebraker, Efficient Organizations of Large Multidimensional Arrays, *Proc. of the Tenth International Conference on Data Engineering*, Feb. 1994

[Vellanki99] V. Vellanki and A. Chervenak, A Cost-Benefit Scheme for High Performance Predictive Prefetching, *Proc. of Supercomputing '99*, Nov. 1999

[Wu03] K. Wu, W. Koegler, J. Chen, A. Shoshani, Using Bitmap Index for Interactive Exploration of Large Datasets, *Proc. of SSDBM,* 2003

WebGD Framework and WebGD-Gen Application Generator

Toshimi Minoura, Surya Halim, and Bader Albader

School of Electrical Engineering and Computer Science
Oregon State University, Corvallis, Oregon, 97331-4602, minoura@cs.orst.edu

Abstract

A typical Internet map-server application allows only *retrieval* of maps and map-related data. We have been developing *web-based GIS/database* (WebGD) applications that allow users to *insert*, *query*, *update*, and *delete geographical features* and the data associated with them from standard Web browsers. The code shared by these applications is organized as the WebGD framework. We have also built a WebGD application generator (WebGD-Gen) that *automatically* produces a WebGD application from a database schema. This application generator greatly simplifies the process of creating a complex Web-based GIS/database application and significantly reduces the development time and maintenance cost. The WebGD framework and WebGD-GEN currently support such advanced features as *tight integration* of a Web-based map interface with a database, *automatic selection* of the *spatial reference* and *map layers* for the current region, and *automatic generation* of Web forms. The forms generated can be used to *insert*, *search*, *update*, and *delete* geographical features and the data associated with them.

1. WebGD Applications

The Web interface of one of the WebGD applications, Oregon Natural Heritage Information System, is shown in Figure 1. This application provides a map interface for a copy of the Biotics 4.0 database maintained by the Oregon Naural Heritage Information Center. Biotics 4.0 is a desktop GIS application built on the database developed by NatureServe. The key elements in this database are *element occurrences* (EOs), which are *areas* of land and/or water in which species are or were present [1]. EO records have both *spatial* and *tabular* data, and the database contain approximately 700 relational tables [2]. The Biotics Mapper implemented with ArcView by NatureServe provides a map interface that allows EO representations and associated data to be created, updated, and deleted. In our implementation, we can perform these operations with standard Web browsers. Also, Web forms, approximately 3500 in total, are provided for all the tables in the database.

The NHIS application enables *bi-directional* movement of *geospatial data* as well as ordinaly data. Scientists and others with proper authentication can *insert*, *query*, and *delete* geographical features such as EO polygons, lines, and points, as well as the data associated with them. Queries can be executed by spatially *selecting an area* on the map or by using a traditional web form. In addition, one-meter resolution *digital orthographic quadrangles* (DOQ), or aerial images, are included as a layer. When DOQ images are combined with other map layers such as highways, county boundaries, streams, and streets, locations can be easily pinpointed by taking advantage of features between map layers [3].

The major operations supported by the map interface of a WebGD application are as follows.

1. To retrieve information on the geographical features in the area of interest, the user can zoom in/out to that area by using the map navigation tools. If the user zoom-in enough, in the case of the NHIS application, to the coastal areas or the areas along Interstate highways in Oregon, one-meter resolution aerial photos are displayed. The user can also go to a new area by selecting an entry in the **Quick View** menu.

2. To get information about a geographical feature, the user can select a layer in the legend and **Information** in the function menu, and then click the boundary of the feature.

3. Function **Insert** allows a geographical feature to be added with mouse clicks on the map. **Done** need be pressed after all points are entered.

Figure 1: Interface of WebGD Application: Natural Heritage Information System.

[4] Function **Search by Area** allows the user to retrieve the list of features that are within a *bounding box* specified on the map and that satisfy a search condition. The features that satisfy the search condition are *highlighted* on the map. Furthermore, the user can select features in the list by marking the checkboxes associated with them. Then, if the map is refreshed, the selected features are highlighted.

[5] The data administration interface can be activated by clicking on the **Database** entry in the menu bar below the banner. A tree icon can be clicked to display a *treeview* for browsing. The treeview for **Higher Taxonomy** is the major one.

Several WebGD applications produced with the WebGD framework and WebGD-Gen can be accessed at the following URLs:

```
http://yukon.een.orst.edu/ms_apps/nhis_cs540/gmap75_main.phtml
http://yukon.een.orst.edu/ms_apps/digir/gmap75_main.phtml
http://yukon.een.orst.edu/ms_apps/calflora/gmap75_main.phtml
http://yukon.een.orst.edu/ms_apps/w6grin_cs549/gmap75_main.phtml
http://yukon.een.orst.edu/ms_apps/soilviewer3/gmap75_main.phtml
```

The first application is Natural Heritage Information System. Although this application can cover the whole USA or the world, the data are currently available only for Oregon. The second application provides a map interface for a local database containing DiGIR records harvested from DiGIR providers, DiGIR (Distributed Generic Information Retrieval) is an XML-based communication protocol for a *federation* of databases managed by natural history museums**.** The third application provides a prototype map interface for flora occurrences in California. The fourth one is a Web-based mapping application for a plant germplasm collection maintained at Western Regional Plant Introduction Station (USDA-ARS) [4, 5]. The fifth one allows the soil information at the location where a mouse click occurs on the map interface to be retrieved. The soil map displayed is for Yolo County in California.

The WebGD framework is built on Minnesota Mapserver [6] with MapScript [7] and PostGIS [8].

One salient feature of the current WebGD framework is *dynamic switching of spatial references* [9]. Typically, different geographic regions and localities have preferred *map projections* in order to avoid distortions in the maps created . The framework allows the whole world to be covered with multiple-levels of maps, e.g., the world map, continent maps, and regional maps. The map interface then automatically selects the most suitable projection for the region whose portion is displayed. For example, the world can use the *geographical coordinate system*, the United States the *Albers equal-area projection*, and Oregon the *Lambert conformal conic projection*. Thus, spatial analysis can be performed with the most appropriate projection for a particular area. The *dynamic switching* of the *spatial reference*, the *map file*, the *legend*, and the *quick view menu* supported by the current WebGD framework allows any part of the world to be covered with its own scale and spatial reference, including regions with one-meter resolution aerial images. Providing aerial images is a very important feature, especially now that the cost of storing aerial images for the entire US has dropped to affordable levels. Furthermore, many states are putting aerial images in the public domain.

2 WebGD-GEN Application Generator

Several tools have been developed to augment the WebGD framework and simplify application development. The WebGD Web-site generator (WebGD-Gen) can create an entire WebGD application, including a web-based mapping interface. WebGD-Gen *automatically* generates a consistent set of Web scripts from *configuration files*, which are again automatically generated from a relational database schema. Since form generation is automatic, the cost of application development is greatly reduced. For a database such as Biotics that contains approximately 700 tables, programming all the required 3,500 (700 x 5) forms manually can be very costly, even infeasible.

WebGD-Gen is implemented as a collection of *templates*. Each template, combined with a corresponding *configuration file*, generates one of the following six types of Web scripts: *search, select, edit, information, action,* and *treeview* scripts. Templates and configuration files are written in PHP. The Web scripts generated by them are also in PHP. The generated scripts are executed on a Web sever by a PHP interpreter. Each script, except for an action script, creates a Web form that is displayed on a client computer by a Web browser.

Furthermore, WebGD-Gen can automatically generate the statements for inserting, searching, and deleting *geographical features* if the following lines, e.g., are added to a configuration file:

```
$web_gd = 'MULTIPOLYGON';
$layer_name = 'grp_eo_py'
$geometry_column = 'the_geom';
$gid_column = 'gid';   // primary key
$db_table_srid = 6010; // spatial ref
```

The forms generated for geographical features can perform the following additional functions compared to those for ordinary database tables

[1] A search form can be activated from a map interface. In this case, the extent of a search box specified on the map is passed as additional search parameters.

[2] A select form includes additional JavaScript code for highlighting geographical features retrieved or selected by the user.

[3] An edit form can insert a record for a geographical feature, after transforming the coordinate values from the spatial reference used by the current map interface to the one used by the geometry column for the record.

The forms related are automatically linked each other. For example, the *edit* form for a Student table. From this edit form, the user can open the forms for the department and courses related to the student. The information needed to create the links are extracted from the *primary-key/foreign-key relationships* among the tables in the database.

3. WebGD Development History

The WebGD framwork and WebGD-GEN were developed *incrementally* and *iteratively* during the last four years. We first implemented in 2000 an application that allowed point features to be inserted on a map by using ASP with ArcIMS and ArcSDE. In 2001, we re-implemented this application with ASP.NET, as ASP.NET provides Web controls, which are better building blocks for Web pages [10]. Based on this application, the first version of WebGD framework was created in 2002 in order to support multiple applications [11].

In early 2003, we re-implemented an application called Motels Oregon with MapServer, PostGIS, and PostgreSQL [12]. This version on Linux was more reliable and faster than the old one, as welll as being built with free software. While implementing the next MapServer application, which was a germplasm resource management system (GEM-GIS), we created the first version of WebGD framework for MapServer. This framework was then enhanced so that it can handle line and polygon features as well as point features.

The two major enhancements made to the WebGD framework in 2004 were *dynamic switching* of *spatial references* for different regions [9] and *automatic generation* of Web forms that can be used to insert, query, and delete geographical features. The form generator is based on our earlier work [13].

4. Conclusions and Future Work

We have been developing the WebGD framework and the WebGD-GEN application generator for Web-based GIS/database applications. A Web-based GIS application generated by them are unique in the following respects.

[1] The geographical features can be inserted, queried, and deleted from the map interface displayed on a standard Web browser.

[2] Web-forms that manage data on geographical features and data in ordinary database tables can be automatically generated.

[3] Dynamic switching of spatial references allows an application to cover different regions with different map files, map legends, and quick-view lists. This is an important feature needed for an application that covers the entire USA or the world.

The cost of running our applications is extremely low. We could put copies of such large databases as Biotics, Fishbase, and a part of National Germplasm Resource Information System on a $800 PC. The software tools we use, such as the University of Minnesota MapServer, PostgreSQL DBMS, PostGIS, Apache, and PHP are all available for free. The GIS data used, such as those from USGS, TIGER/LINE, and Digital Chart of the World (DCW), are also in the public domain. Automatic code generation of a WebGD application will save a great deal of effort in the development of a spatial decision-support system. Although some manual customization is required, the time needed to customize can be lowered to weeks or months compared to the years required to build a *spatial decision-support system* from scratch.

References

[1] NatureServe (February 2002). Element Occurrence Data Standard. Retrieved January 4, 2004, from http://whiteoak.natureserve.org/eodraft/all.pdf.

[2] Fogelsong, C. (December 2002). Biotics 4.0 data model version 1.0. Retrieved January 5, 2004, from http://whiteoak.natureserve.org/hdms/HDMS-DataModel.shtml.

[3] Wuttiwat, T., Minoura, T., and Steiner, J. (May 2003). Using Digital Orthographic Aerial Images as User Interfaces. In *Proc. of ASPRS Annual Conference*, Anchorage, Alaska.

[4] USDA-ARS. Western Regional Plant Introduction Station, USDA - Agricultural Research Service, Pullman, Washington, http://www.ars-grin.gov/ars/PacWest/Pullman/

[5] Sharma, A. (December 2003). Web-based analysis module for a germplasm collection. M.S. report, School of Electrical Engineering and Computer Science, Oregon State University

[6] University of Minnesota. *MapServer*, http://mapserver.gis.umn.edu.

[7] DM Solutions Goup Inc. *PHP MapScript*, http://www.maptools.org.

[8] Ramsey, Paul. *PostGIS Manual*, Refractions Research Inc, http://www.refractions.net.

[9] Dileep Arur and Toshimi Minoura. Dynamically Switching Projections in a Web-Based GIS/Database (WebGD) Application, Geotec Event 2005.

[10] Wangmutitakul, P., Li, L., and Minoura, T. User Participatory Web-Based GIS/Database Application. In. *Proc. of Geotec Event Conference*, March 2003.

[11] Wangmutitakul, Paphun, et al. WebGD: Framework for Web-based GIS/database Applications, *Journal of Object Technology 3*, 4, 209-225, 2004.

[12] Sano, J., Wanalertlak, N., Maki, A., & Minoura, T. (July 2003). Benefits of web-based GIS/database applications. In *Prosc. of 2nd Annual Public Participation GIS Conference*, Portland, Oregon.

[13] Eum, D. and Minoura, T. (June 2003). Web-based database application generator. *IEICE Transactions on Information and Systems, Vol. E86-D, No. 6.*

A Robust, Low-Cost Virtual Archive for Science Data

Christopher Lynnes
Bruce Vollmer
NASA/Goddard Space Flight Center
Christopher.S.Lynnes@nasa.gov, Bruce.E.Vollmer@nasa.gov

Abstract

Archiving data online, as opposed to in tape silos, is attractive for its reduced complexity and access latency. Low-cost mass-market disk drives help make this affordable but are hard to scale to the petabyte range, with unknown reliability in archival use. However, a "virtual" science archive, i.e., producing data on demand when requested by users, needs only a fraction of the data online. Radiance data from the satellite-borne Moderate Resolution Imaging Spectroradiometer instrument provide an opportunity for a virtual archive at the Goddard Earth Sciences Distributed Active Archive Center (GES DAAC): the raw data are only one-fourth the size of the derived radiances. However, virtual science archives face special challenges in securing the raw data and assuring the quality of the derived products. The GES DAAC is prototyping a virtual archive to tackle these challenges and demonstrate feasibility.

1. Introduction

Despite their expense, tape silos are sometimes the only affordable option for petabyte-scale science data archives. Though disks have been dropping fast in price per unit storage, they still have not surpassed tapes, and they have many logistical issues, such as data reliability, floor space, power and cooling load. On the other hand, tape silos have their own disadvantages: multiple layers of software and firmware needed to manage tape silos introduce a daunting complexity; continuous mounting, winding and dismounting of tapes are vulnerable to mechanical problems; and access latency to requested files can extend from minutes to hours in very active archives. Thus, migrating data from tape to disk can be attractive to archives like the Goddard Earth Sciences Distributed Active Archive Center (GES DAAC), a 2-PB archive of earth science remote sensing data.

Drastic reductions in low-cost mass-market PC hard disks (~$1/GB in January of 2005) would seem to make such a migration more affordable. However, the cost is still relatively high, and it is challenging to scale assemblages of these disks to the petabyte range. Also, their use as archive devices is untested.

2. Solving the Cost Problem of Online Archives

One approach to this dilemma is to store only the raw data online, and "virtualize" derived data products: that is, offer the derived products to users but produce them only when requested by a user. There are several such systems that employ "lazy" processing for satellite data, *e.g.*, "BigSur"[1] and the Earth Science System Workbench[2]. This solution is especially attractive when derived products represent much greater data volumes than raw data. At the GES DAAC, the raw data flow from the Moderate Resolution Imaging Spectroradiometer (MODIS) instruments on the Aqua and Terra satellites is 140 GB/day, about one-fourth the volume of its derived radiance products (550 GB/day). Thus virtualizing the MODIS radiance products reduces the amount of disk needed, making an online solution affordable.

Still, we are faced with two immediate challenges. Storing all the MODIS raw data to date requires 185 TB of disk: deploying low-cost mass-market disks in a disk subsystem (*e.g.*, in a Storage Area Network) requires significant engineering, taking into account floor space, power, cooling load, interconnections, and management software, all of which threaten to eat up the savings from the disks themselves. Secondly, we need significant processing power to produce the derived products on demand. For example, in March of 2005, the GES DAAC shipped about 300 GB of MODIS radiance and geolocation data per day to end users, or about 7000 data files/day. At the GES DAAC, we are prototyping a single solution to solve both problems: attaching the disks to the 90-odd personal computers used by GES DAAC staff for terminals and office automation. This has the effect of dispersing the floor space, power and cooling load requirements, while at the same time serving as a processing cluster for computing the derived products. Office automation and terminal applications typically use small fractions of the CPU in today's computers, leaving substantial power available for processing data.

This low-cost archive/processing cluster is integrated in an end-to-end system from user interface to data delivery for the on-demand derived products (Fig 1).

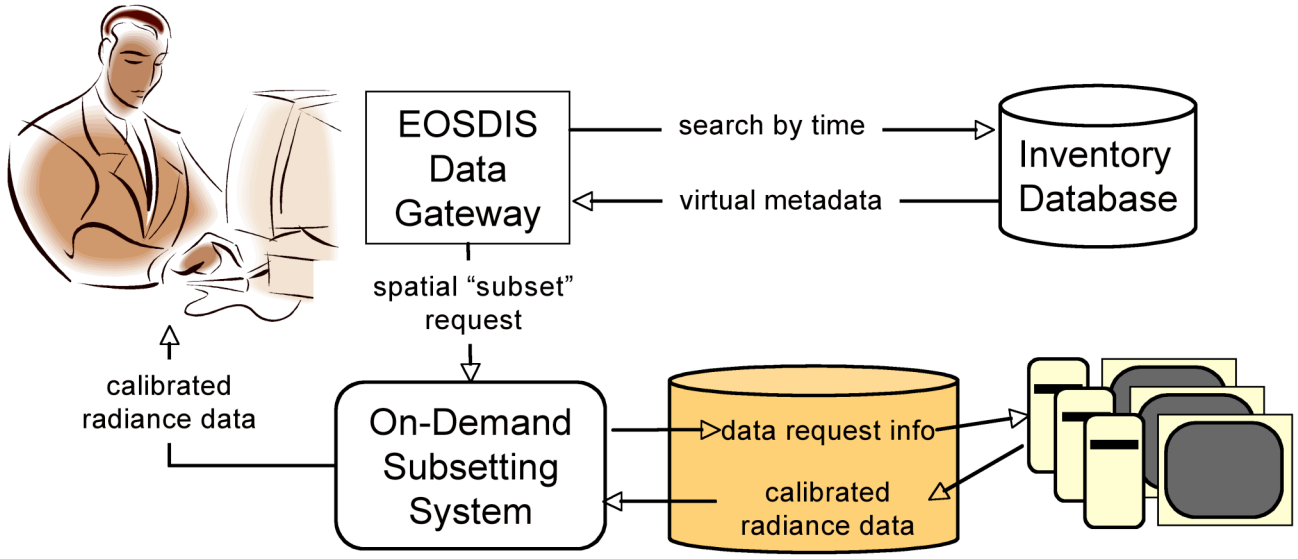


Figure 1. Implementation of a virtual product system within the framework of an existing on-demand subsetting system.

The GES DAAC currently operates an on-demand data subsetting system that is integrated with the overall EOSDIS search and order system, including the EOSDIS Data Gateway. It is possible to represent a specific scene of a virtual product as a spatial subset of a virtual daily product. Thus, the user searches for time periods of interest at a daily granularity and then specifies the spatial area for the results he/she wants to order. The resulting subset request is sent to a central subsetting system running on a Sun server, where it is processed almost like any other subset request. However, instead of subsetting the stub file that represents the virtual daily product, the "subsetting algorithm" places the request information in a predetermined directory for the remote PC that hosts the necessary raw data, and then waits for a response. The PC polls this directory regularly for these "work orders"; when it detects one, it processes the raw data using the MODIS science processing software [3][4][5] to produce the requested products and returns the output to the central subsetting system, which handles the subsequent distribution to the user.

The remote PC runs the processing job at a lower than normal priority so as not to interfere with the user's main applications. However, those applications use significant CPU only at rare intervals (e.g. startup, saving documents, printing), so this does not slow down the processing significantly. This PC "cluster" is actually quite heterogeneous, consisting of Windows XP, Linux (Intel) and Macintosh (OS X) workstations. As a result, the science processing algorithms had to be ported to a variety of platforms. The Windows XP port is actually accomplished using the Cygwin environment, which serves as a kind of "Linux emulator" for Windows.

3. Incorporating Robustness

While implementing the end-to-end virtual product system is a simple matter of coding, incorporating robustness is more challenging. In virtualizing a science archive, robustness takes on some unique aspects: integrity of the raw data; quality and reproducibility of the on-demand derived products; and availability of the derived products. Each of these aspects requires a different approach.

3.1 Raw Data Integrity

For a science data archive, the integrity of the raw data is the most important factor. In tape archives, the most serious threat is usually the loss of individual tapes. Similarly, loss of individual disks or disk packs represents a risk to disk archives. This can be mitigated partially by using RAID, or almost completely by keeping a second copy of data on a disk that is physically distant. However, the quick random access that makes online archiving so attractive for users also presents the most insidious threat, that of a virus or worm rapidly multiplying and corrupting huge numbers of data files. Thus, it is essential that backup copies of disk archives be physically protected from possible corruption, such as by a hardware read-only switch, air-gap or even complete disconnection from any computer or network. On the other hand, one of the

advantages of the disk based archive is that it is far easier and more feasible to conduct continual checking of the online data, now that we need no longer treat tape mounts as a precious resource.

This requirement for offline disk backup affects the organization of data on disk. Since user orders tend to favor data over the continents, it might be desirable to put such input data on the fastest workstations, with adjacent spatial areas on different machines to enable parallelism for orders that cover large spatial areas. On the other hand, the data arrive continuously from the satellite; as backup disks are filled up, they are taken offline to protect the raw data. Thus, data for a given time range are backed up on a small number of disks. Were we to load up one primary disk with say, just North American continental scenes, the time range covered would be relatively large, so its backup images would be spread over many backup disks, which would all need to be physically reconnected to reconstruct the failed disk. Also, the science algorithm requires data from adjacent input scenes within an orbit to execute correctly, which could force cross-disk transfers. As a compromise, we place one full orbit on a disk, with a one-scene overlap, moving to another machine for the next orbit. With this scheme, time-sequential backups are supported, while requests with large horizontal extents are spread across several machines.

3.2 Data Quality and Reproducibility

A unique challenge faced by science data archives is acceptance of the concept by the end users. When derived products are stored in a tape archive, the users have a certain degree of confidence in them as standard products, produced under controlled conditions. Producing such data on the fly does not always produce such confidence. The science processing algorithms themselves record the input data files and algorithm version (*i.e.*, the pedigree) of the derived products within their metadata, a crucial first step. Beyond that, however, the calibrated radiance data are still produced routinely as the raw data are first received, so that gridded geophysical parameter products can be produced. These routinely produced radiance products are made available on disk for a short period of time (~2 weeks) for immediate download by users (thus reducing the amount of ad hoc on-demand requests). At the time of initial processing, subsampled versions of these products are also stored online permanently for use by scientists that do not need high spatial resolution (Fig 2). These subsampled products provide an opportunity to check the on-demand products to see if the data points are the same, so that we can assure that the latter are of equal quality to the routinely produced standard products. This provides additional quality assurance information[6].

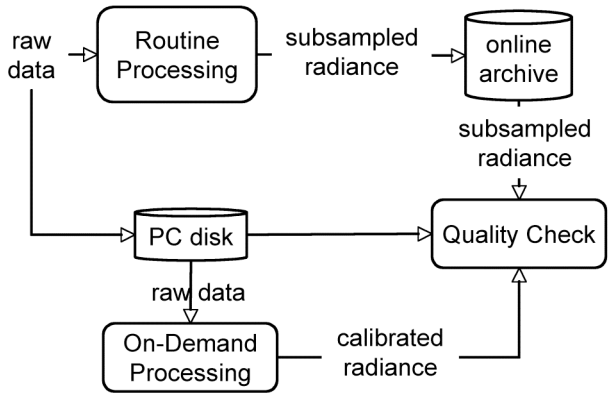


Figure 2. Quality control of on-demand products using an online archive of subsampled radiances.

3.3 Availability of Derived Products

The most complex aspect of the robustness problem is the availability of the derived products. This is a function of the availability of the individual computers, the correctness of the algorithm software and configuration on each computer, the availability of each data disk, and in rare cases, the load on the computers.

This is an area where some of the initiatives in autonomic computing may be of use. One relatively straightforward approach to part of the problem is to have each PC transmit a small signal file at regular intervals to the central subsetting Sun server. While the presence of the file itself represents a useful heartbeat, we can add a small amount of information to the signal file itself, such as CPU load or data processing performance, *i.e.*, a kind of pulse [7]. The heartbeat / pulse of each station can then be integrated into the overall monitoring interface for the GES DAAC, where it can be monitored by the operations staff on a 24x7 basis.

However, it is also important to know if the science processing software is properly configured and working. One way to ensure this is for the PC to process small slices of raw data when no on-demand processing is happening. (In a sense, this is analogous to the Folstein Mini-Mental Status Exam used by physicians to screen for cognitive disorders[8].) The results of the mini-tests are included in the pulse sent back to the central subsetting server, with the useful side effect of continually checking the health and integrity of the raw data.

4. Current Status and Future Plans

A proof-of-concept system with end-to-end functionality of virtual product creation has been constructed on a small number of PCs, including

representatives from Windows, Linux and Macintosh OS X platforms. Work is currently underway to incorporate the robustness features outlined above.

In order to keep initial costs as low as possible, the current implementation reuses existing code at the GES DAAC, with a few minor modifications. However, some aspects of the system mirror those in grid technology, suggesting possible enhancements based on grid standards. For example, virtual product systems have been implemented in such grid projects as GriPhyN [9] using Chimera and Pegasus to handle workflow. Also, distributing the data amongst the various PCs might be managed more easily using a data grid technology such as the Storage Resource Broker [10].

5. Advantages of Virtual Archives

Ultimately, the most difficult test for virtual archives may be acceptance by the end user community, who are often more comfortable with pre-constructed products retrieved from archive. In order to overcome this, we must first demonstrate that the products produced on-demand are of equal quality as pre-constructed products. Beyond that, however, virtual archives offer some potential advantages to the users. One advantage of this particular implementation is that the users get only the spatial area they are interested in, whereas the pre-constructed products cover fixed areas (5 minutes of MODIS data in each scene), resulting in large files. A second important advantage is that when a processing algorithm is updated, the users need not wait for all the data to be reprocessed to get the particular time and spatial period they are interested in. Instead, they can request the data for that space-time region as soon as the algorithm update is installed, which may be as much as a year (or more) earlier than the data would be available in the course of routine reprocessing.

In addition to the user advantages, the archive benefits as well. The complexity of the hardware-firmware-software interactions associated with tapes and robotics is largely replaced by science processing software, where problems are easier to diagnose (and fix). Perhaps more importantly, however, the problem of migrating data to new technologies is drastically reduced in size. Instead of physically moving large quantities of data from older tape drives to newer, more advanced tape drives, the data migration problem is replaced by the issue of migrating the science processing software to new versions of the operating system and libraries, a task which is much faster and easier.

6. Acknowledgements

We would like to thank three reviewers for very helpful comments, M. Theobald and the S4PMViP team for work on the virtual processing system, and M. Clausen, D. Isaac, H. Ramapriyan, G. McConaughy, M. Hinchey and R. Sterritt for useful and stimulating discussions of virtual product systems.

7. References

[1] P. Brown and M. Stonebraker, "Big Sur: A system for the management of Earth science data," presented at 21st International Conference of Very Large Data Bases, Zurich, Switzerland, 1995.

[2] J. Frew and R. Bose, Earth System Science Workbench: A Data Management Infrastructure for Earth Science Products, Proceedings of the 13th International Conference on Scientific and Statistical Database Management (SSDBM 2001), Fairfax, VA, 2001, pp. 180-189.

[3] M. Nishihama, R. Wolfe, D. Solomon, F. Patt, J. Blanchette, A. Fleig, and E. Masuoka, MODIS level 1A Earth location: Algorithm theoretical basis document version 3.0. SDST-092, MODIS Science Data Support Team, 1997.

[4] X. Xiong, J. Sun, J. Esposito, B. Guenther, and W. Barnes, "MODIS Reflective Solar Bands Calibration Algorithm and On-orbit Performance," Proceedings of SPIE – Optical Remote Sensing of the Atmosphere and Clouds III, 4891, 2002.

[5] X. Xiong, K. Chiang, B. Guenther, and W. L. Barnes, MODIS Thermal Emissive Bands Calibration Algorithm and On-orbit Performance, Proceedings of SPIE — Optical Remote Sensing of the Atmosphere and Clouds III, 4891, 2002.

[6] C. Lynnes and M. Clausen. Virtual Data Products in an Intelligent Archive, White Paper prepared for the Intelligent Data Understanding program, 15 p., http://disc.gsfc.nasa.gov/IDA/VirtualDataProducts_v1_0.pdf, 2003.

[7] R. Sterritt, Pulse monitoring: extending the health-check for the autonomic grid, Proc. IEEE International Conf. on Industrial Informatics, 2003, p. 433-440.

[8] M. Folstein, S. Folstein, P. R. McHugh, Mini-Mental State: a practical method for grading the cognitive state of patients for the clinician, J. Psych. Research, 12, 1975, p. 189-198.

[9] E. Deelman, K. Blackburn, P. Ehrens, C. Kesselman, S. Koranda, A. Lazzarini, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn and R. Williams., "GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists," 11th Intl Symposium on High Performance Distributed Computing, 2002.

[10] Moore, R., C. Baru, "Virtualization Services for Data Grids", in *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons Ltd, 2003.

Session 3: Workflow and Lineage

Efficient Scheduling and Execution of Scientific Workflow Tasks

Laura Bright David Maier

Department of Computer Science
Portland State University
Portland, Oregon
{bright,maier}@cs.pdx.edu

Abstract

Large-scale scientific workflows are often characterized by tasks that produce or consume large amounts of data (frequently both) and generate large volumes of derived data products. Minimizing the end-to-end running time of a set of workflow tasks is important to deliver data products in a timely manner and free up processors to accomodate additional workflows. A single workflow task may perform the same computations on multiple files, presenting many opportunities for concurrent execution on multiple nodes of a Grid. In addition, many different tasks may operate on the same large input files. An important challenge to efficient workflow execution on multiple nodes is determining an assignment of tasks to nodes. Processor and network speeds may vary at different times, workflow tasks may be modified, and new workflows may be added. In this paper we examine algorithms for scheduling tasks concurrently on nodes of a dedicated Grid to address these challenges. We use real workflow tasks from the CORIE Environmental Observation and Forecasting System. We propose a hybrid scheduling approach that exploits knowledge of task running times and locations of input files to assign some tasks to nodes statically, while others are assigned dynamically to adapt to variations in task execution times. We show the effectiveness of our approach using both simulations and our prototype implementation.

1 Introduction

Large-scale scientific workflows are characterized by tasks that may produce or consume large amounts of data and generate large volumes of derived data products. Applications include experimental physics, earth sciences, life sciences, and environmental forecasting. Many workflows in these domains operate on large input files that may be 100s of megabytes or larger. The same workflow may run at regular intervals, e.g., daily, and sizes of the input and output files are often known a priori. Executing an entire workflow on a single machine may take several hours or longer. Several tasks may process the same input file. In addition, a single task may perform the same computations independently on several such files. Minimizing the end-to-end running time (makespan) of a workflow may be important to free up processors for other workflows or to generate high-priority data products. There may be many opportunities to speed up the makespan of a set of tasks by executing independent tasks concurrently on separate machines connected by a local network. We refer to such a set of machines as a *dedicated Grid*.

There are several challenges to effective execution of scientific workflow tasks in a dedicated Grid environment. First, different nodes of the Grid may have different speeds, and the execution time of a given task on the same node may vary depending on factors such as time of day and workload on the node, and task execution times may depend on the input data. Even with accurate statistics from previous executions, it is impossible to know exactly how long a task will run on a given node. An assignment of tasks to nodes that appears to be optimal may take noticeably longer than expected to complete. Task reassignment techniques have been proposed [9, 22], but these may have high overhead.

Another set of challenges relates to the dynamic nature of both the workflows and nodes. In a dynamic Grid environment nodes may be added or removed at any time, so statistics on the speed of a new node may not be available and the number of nodes may change. Tasks may be modified to generate new data products or process different files, which may affect their running times. Finally, different tasks may have different priorities, so we may need to minimize the makespan of the highest priority tasks.

A third challenge relates to precedence constraints. In a typical workflow, some tasks will generate data that serves as input to one or more subsequent tasks. Workflow tasks whose outputs serve as inputs to many tasks should be executed as early as possible, and any assignment of tasks to nodes must take these constraints into account.

A final challenge relates to data transfer overhead. All needed input files must be transferred to the node where a task is executed. Even on a local network, this transfer may take several minutes in the worst case. Since many tasks might process a file, this overhead may be reduced by assigning tasks that process the same file to the same node.

In this paper we consider algorithms for scheduling and executing scientific workflow tasks on a dedicated Grid that address these challenges. We consider both static (assigning tasks a priori) and dynamic (assigning tasks as they become available) scheduling approaches, and discuss the limitations of each. We propose a *hybrid* scheduling approach that combines the benefits of both the static and dynamic approaches. Our hybrid approach is well-suited to handling the challenges posed by scientific workflows, including reducing data transfer overhead and adapting to variations in network speeds and execution times at nodes. We present heuristics to determine which tasks to assign to nodes statically and which to assign dynamically. We present a fast algorithm to assign static tasks to nodes that approximates the optimal makespan of a set of tasks. The approximation algorithm allows us to scale up to increasingly large workflows, quickly adapt to changes in the number of available processors, and accomodate custom workflows.

We motivate and evaluate our work using examples from the CORIE Environmental Observation and Forecasting System [4]. CORIE measures and simulates physical properties of the Columbia River Estuary. CORIE includes daily forecast workflows (described in Section 2) as well as hindcast and custom workflows that are run as needed. Many workflow tasks operate independently on multiple input files, and the sizes of these input files are typically the same from run to run. We note that many scientific workflows share similar properties, thus, we believe our results will apply to other domains.

We evaluate our hybrid scheduling approach using our simulator, which can model a scientific workflow execution, and using our prototype implementation built on Thetus [19]. Thetus is a commercial product that enables scientific workflow execution. Our simulations model existing CORIE workflow tasks, and our implementation executes existing CORIE workflow tasks concurrently at separate nodes to improve performance [5]. Our results indicate that the hybrid approach can indeed reduce the makespan of a set of workflow tasks, and identify areas of further research.

There are several key differences between scheduling scientific workflow tasks on a dedicated Grid and other work in Grid and multiprocessor scheduling. First, many of the workflows we consider are run regularly, e.g., daily, and we therefore have statistics on past executions. Unlike a globally distributed Grid environment where many different users may submit workflows for execution, we consider a set of dedicated nodes where scientists may regularly run one or more standard workflows in addition to occasional custom workflows. Further, a dedicated Grid may include an ad-hoc collection of machines of varying speeds connected by a local network and might not have the computation or network speed of large-scale Grid computing systems. Second, the number of workflows run and data products generated is limited only by the available resources, and scientists will run additional workflows and generate additional data products if additional resources become available.

This paper is organized as follows: Section 2 gives an overview of the CORIE Environmental Observation and Forecasting System and presents details of its workflows and challenges posed by these workflows. Section 3 presents algorithms for scheduling workflow task execution on a Grid, including our Hybrid scheduling algorithm. We present experimental results from our simulation and our prototype implementation in Section 4. We survey related work in Section 5 and conclude in Section 6.

2 CORIE Overview

CORIE [4] is an environmental observation and forecasting system that measures and simulates the physical properties of the Columbia River Estuary and surrounding coastal regions. Applications address issues including salmon habitat and passage, hydropower management, navigation improvements and habitat restoration. The system includes both *forecast* and *hindcast* simulations. *Forecasts* are used to predict near-term conditions in the river, while *hindcasts* are run retrospectively for specific time periods using observed values. Hindcasts typically consist of an extended set of simulations covering a year. They may also include calibration runs to test the sensitivity of the model to empirical parameters. CORIE scientists are continually adding new workflows and data products. They plan to include forecasts of 30 different coastal regions in the near future, thus flexible and scalable scheduling algorithms will become increasingly important. In addition, existing workflows may be modified, for example, by changes to models or parameters, or may be used to process a new dataset.

We first discuss details of several tasks in the existing forecast workflow that illustrate some challenges to efficient execution of tasks and that will be used throughout this paper. We then describe our execution environment and outline the specific challenges we address.

2.1 Workflow Tasks

We consider a set of tasks that process several large input files generated by the ELCIRC simulation [27]. These files contain values of different physical variables, such as salinity and velocity, computed over a 3D finite-element grid at

File	Size
hvel.64	655MB
salt.63, vert.63, temp.63	334MB
wind.62	35MB
*.61	23MB

Table 1. Input Files for CORIE Workflow Tasks

regular time steps covering multiple days of simulated time. These files are used to generate many images such as transects (vertical slices of the river) and isoline plots (horizontal slices). A single forecast run generates two files of each type, i.e., 1_salt.63, 2_salt.63, for both the current day and the next day, and a hindcast over a week generates seven of each file. Thus, the entire set of hindcast and forecast runs consists of a large number of tasks and data files. For simplicity, we focus on a 1-day subset of a forecast workflow in the remainder of this paper.

Table 1 summarizes the sizes of each of these files for the Columbia River forecast. Note that this table does not include the times to transfer each file, which is typically about 10MB/sec in our network. We next describe the details of three tasks, *Isolines*, *Transects*, and *Model Stations*, that each operate on some of these files. Tables 2 and 3 summarize the details of these tasks.

Task	Running Time(sec)	Input Files
isolines {airt,fllu, fllu,flsu,pres, radd,radu,srad}	25	{airt,fllu,flsu pres,radd radu,srad}.61
isolines salt	70	salt.63, wind.62
isolines vert	33	vert.63, wind.62
isolines temp	10	temp.63, wind.62
isolines hvel	73	hvel.64, wind.62
transects salt	202	salt.63
transects vert	15	vert.63
transects temp	197	temp.63
transects hvel	185	hvel.64

Table 2. Approximate running times of Isolines and Transects tasks

Task	Time (sec) Time (sec)	Inputs
do station extraction	60	salt.63,vert.63,temp.63 hvel.64, elev.61
plot model stations (146 tasks)	30	xxxxx.dat (146 files)

Table 3. Approximate running times of Model Stations workflow tasks

2.1.1 Isolines and Transects

The Isolines task generates animations of horizontal slices of the Columbia River illustrating different conditions such as temperature, salinity, and velocity at varying depths and locations. The task uses data from the wind.62 file and operates on the four largest files (hvel.64, salt.63, temp.63, and vert.63) as well as seven smaller 23MB files. The animations generated from each file may differ. For example, the workflow currently generates 13 different isolines animations from the salt.63 files, but only four from the vert.63 files, so Isolines salt has a longer running time than Isolines vert. We note that CORIE scientists may modify which animations are generated for each file, which could affect the running times of each subtask. The Transects task generates animations of vertical slices along different paths up the Columbia River, for the four largest files only. As in Isolines, different animations may be generated for different files, so some subtasks may run longer than others.

To exploit opportunities for concurrency we divide Isolines into 11 subtasks, one for each file, and similarly divide Transects into four subtasks. Since both tasks process four of the same large files, assigning subtasks that process the same files to the same nodes can reduce data transfer overhead and improve performance. We refer to these subtasks as *tasks* in the remainder of the paper. We discuss task granularity issues in another paper [5]. We note that scientists may modify the granularity of some tasks, which is another way a workflow and its running time can change.

2.1.2 Model Stations

The *Model Stations* segment of the CORIE forecast workflow consists of two tasks. First, the task do_station_extraction processes hvel.64, salt.63, temp.63,vert.63, and elev.61 and generates 146 inputs to the task plot_modelstations, which generates plots for 146 different stations. Thus, this workflow segment includes a precedence constraint, and do_station_extraction should be executed as soon as possible to enable the execution of the 146 plot_modelstations tasks. While a single invocation of plot_modelstations is relatively short (under 30 seconds), running the task on all 146 stations requires intelligent scheduling to minimize the end-to-end running time. As we will see in Section 4, poor scheduling choices can significantly increase the running time.

2.2 Existing Implementation

The existing CORIE workflows consist of a collection of executable programs written in Fortran, C, and Perl. The

current workflow implementation runs on a shared filesystem, which provides convenient access to files across multiple nodes but misses many opportunities for concurrent execution. There is no easy way to coordinate workflow execution among multiple machines.

The existing machines are dedicated nodes shared over a local network. While they are not shared by outside users or jobs, there may be contention for the nodes among different CORIE workflows and tasks. For example, during a daily forecast run, there may also be hindcasts or calibrations running. As mentioned earlier, CORIE scientists would like to generate as many data products as possible and will use all available capacity on the nodes.

2.3 Experimental Prototype

To address some limitations of the existing CORIE workflow implementation, we have implemented an experimental prototype that allows independent tasks to execute concurrently on separate nodes. Our prototype is built on Thetus [19], a commercial product for management of non-text data common in many scientific domains. We chose Thetus because it provides facilities for automatically launching workflow tasks and enables experimentation with scheduling algorithms. In contrast, other scientific workflow packages that we are aware of focus on other challenges such as mapping abstract workflows to nodes [10, 13] or workflow specification [3, 23, 24]. (We discuss these efforts in greater detail in Section 5.) A previous paper [5] gives a detailed description of our prototype. We briefly describe the Thetus features that are useful for efficiently executing scientific workflows.

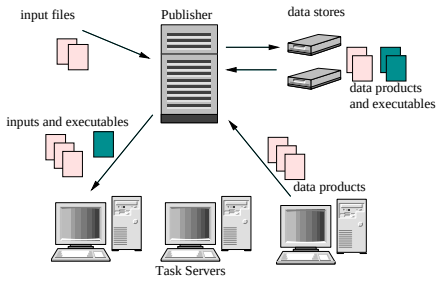


Figure 1. System Architecture

Thetus consists of a *Publisher* and one or more *Task Servers*. The Publisher manages the storage of all data files and executables along with configurable metadata, while the Task Servers execute workflow tasks and upload the resulting data products to the Publisher. We present the architecture in Figure 1. When all the required input files for a workflow task have been uploaded to the Publisher, the task will be launched automatically on one of the available Task Server nodes. This feature facilitates the specification of workflows and ensures the timely execution of workflow tasks. All required input files and executables are downloaded to the node where the task is executing. After a task completes, all data products generated by the task are uploaded to the Publisher, which may launch one or more subsequent tasks to process these products. The Thetus Publisher includes a scheduler that by default assigns tasks to nodes in a round-robin manner as soon as the tasks become available. We experimented with several other scheduling algorithms, as described in Section 3.

We implemented several extensions to Thetus in our prototype. First, we provided data stores on each node to reduce data transfer overhead. Second, we used Thetus file storage facilities to store executables. Any executables that are not stored at a node can be downloaded from the Publisher, enabling the seamless addition of new nodes. We note that executables are typically much smaller than input files and data products, so their data transfer overhead is minimal.

2.4 Challenges

We now consider specific challenges posed by the CORIE workflows.

Concurrent Execution One set of challenges relates to executing independent tasks concurrently on separate nodes. We must efficiently determine an assignment of tasks to nodes that will minimize the makespan of the workflow.

Data Transfer Overhead Transferring large input files to nodes may add considerable overhead to the cost of executing a task. Since many tasks operate on the same files, temporarily storing files locally at nodes and assigning tasks to nodes based on file location can reduce some of this overhead. However, determining when it is beneficial to assign tasks to nodes based on file location, and determining an appropriate assignment, is non-trivial. While this problem may be alleviated in some cases by upgrading the local network or using protocols such as GridFTP [11] or FAST TCP [16], these solutions may not be feasible in all cases.

Varying Processor Speeds and Task Times Another challenge relates to variations in the speeds of processor nodes. While we have statistics on past execution times of workflow tasks, we may not know the exact running time of a task on a given node, especially as nodes are added or removed from the system. Even the speed of a single node may vary depending on external factors. Also, some task running times may be slightly dependent on the inputs. For example, the Isolines task execution time may depend on the range of values in the input data. Variations in the running times of individual tasks may noticeably increase the end-to-end running time of an entire workflow.

Precedence Constraints Another challenge relates to

precedence constraints. Many CORIE workflow tasks generate outputs that serve as inputs to subsequent tasks. Thus, to minimize the end-to-end running time of a set of tasks, it is important to consider these precedence constraints in addition to task inputs and running times. While a detailed study of these challenges awaits future work, we consider the constraints of the Model Stations tasks in this paper as an example. We note that many existing scheduling algorithms [10, 22] consider precedence constraints, typically by representing a workflow as a DAG and computing the earliest starting time of each task. In our ongoing work we plan to investigate similar techniques and consider the challenges of data transfer overhead and task prioritization in the presence of precedence constraints.

Adding or Removing Nodes Our prototype implementation consists of a central Publisher and several remote Task Server nodes. Our eventual goal is to have scientists use this system to run their actual workflows. In that setting, new hardware is continually being added, and existing nodes may temporarily become unavailable. The dynamic nature of the system means that both the number and types of nodes available at any time may change unexpectedly, so a good assignment of tasks to nodes may need to be recomputed on the fly before executing a workflow.

Adding and Modifying Workflows Finally, new workflows are continually being added to the CORIE system. In addition, existing workflows may be modified to change parameters or add new data products, which may affect the running times of tasks. For example, when the scientists are at sea on observation runs, they may run additional tasks to generate higher-resolution forecast data products near the location of the ship. To ensure timely execution of all tasks, we may need to change the assignments of workflow tasks to nodes. We do not currently consider pre-emption of tasks due to the memory-intensive nature of many of them.

3 Scheduling Algorithms

We present several scheduling algorithms to assign a set of workflow tasks to available nodes. We first present two baseline algorithms that can be used in the absence of any prior knowledge of task running times. We then present our *Offline* algorithm, which exploits statistics from previous executions, and our *Hybrid* algorithm, which combines the benefits of both static (offline) and dynamic (online) scheduling techniques. The motivation behind these algorithms was to minimize the makespan of a set of tasks by building on algorithms for the well known bin-packing problem. In the following discussion we use the term *task-server queue* to refer to the set of tasks that are ready to execute and are assigned to a given task server, and the term *common queue* to refer to the queue of tasks that are ready to execute but have not been assigned to a task server.

3.1 Round Robin

Round Robin (RR) scheduling is the default scheduling algorithm used by the Thetus Publisher. It works by immediately assigning each available task to a task-server queue for one of the nodes in a round-robin manner, without considering the current load on any of the nodes. Once assigned, the task waits in a task-server queue on the assigned node until all earlier tasks in the queue have completed. An advantage of this approach is its ease of implementation. In addition, since it immediately assigns tasks to nodes as soon as they are available, it eliminates the need to wait for tasks to complete at nodes. A disadvantage is that it does not consider the inputs to the tasks or the expected running times, so it may assign several long-running tasks to the same node.

3.2 Dynamic

The *Dynamic* algorithm adds tasks to a common queue as they become ready to execute, and assigns the first task in the common queue to a node whenever the node becomes idle. Thus, tasks are assigned to the nodes in the order they become ready to run. In the absence of statistics on the expected running times of tasks, dynamic scheduling is the best approach. While dynamic scheduling overcomes some limitations of RR scheduling, it does not exploit knowledge of task inputs and running times when assigning tasks to nodes and might therefore copy large files unnecessarily or assign long tasks to slow nodes.

3.3 Offline

The *Offline* algorithm improves on RR and Dynamic by exploiting knowledge of task inputs and running times. The goal is to assign a set of tasks to a fixed number of nodes to minimize their makespan. This problem reduces to the well-known bin-packing problem, which is known to be NP-complete [14]. For relatively small workflows that are run repeatedly, we can solve for an optimal solution using brute force. However, this approach will not scale to increasingly large workflows, and does not easily adapt to changes in the running times of workflow tasks or the number of available nodes. Therefore, a fast approximation algorithm that can assign tasks to nodes on the fly is needed.

We adapt the MULTIFIT algorithm [8], which provides a good approximation to the optimal solution to the bin-packing problem but does not consider the potential benefits of assigning tasks with the same input files to the same nodes. We therefore add a heuristic to this algorithm that aims to assign tasks to nodes that already have some or all of the needed input files, as described below.

The Offline algorithm is shown in Figure 2. In MULTIFIT, the algorithm works by sorting tasks in decreasing size order and assigning each task in this order to the "bin" where it fits with the smallest (positive) remaining space. We modify this algorithm to compute for each task both the bin where it fits with the smallest remaining space and the bin that has the largest input overlap (fraction of the task input data stored locally). Note that the "size" of a task is the time to download all inputs not already present at the node plus the task's expected running time. Each task is assigned to the bin with the largest input overlap that has the smallest remaining space. Initially the bin capacity is set to be `mincapacity`, which is either the total running time (of all offline tasks) divided by the number of processors or the running time of the longest task, whichever is larger. As in MULTIFIT, if Offline does not find a solution that satisfies `mincapacity`, we perform binary search in the range [`mincapacity`, 2*`mincapacity`], and repeat the Offline algorithm until a good solution is found.

The subroutine COMPUTE_REMAINDER*(t, p)* in Figure 2 calculates the amount of remaining space if task t is assigned to processor p. The subroutine COMPUTE_OVERLAP*(t, p)* computes the number of bytes of input data to task t that are are also inputs to other tasks already assigned to processor p.

The Offline algorithm is expected to give a good solution assuming that estimates of running times are close to the actual running times. It also assigns tasks with the same input files to the same processor in many cases, so, unlike the Dynamic algorithm, it can reduce data transfer overhead. However, in practice the estimated running times are only averages that may vary considerably in practice, so the actual makespan of a set of tasks may be far from optimal.

```
OFFLINE_ASSIGNMENT (Vector TASKS)
{TASKS is sorted in order of decreasing running time}

foreach t in Tasks {
    foreach processor p {
        remainder = COMPUTE_REMAINDER(t, p)
        overlap = COMPUTE_OVERLAP(t, p)
        if remainder < min_remainder
            min_remainder = remainder
            rem_proc = p
        end if
        if overlap > max_overlap and remainder ≥0
            max_overlap = overlap
            overlap_proc = p
        end if
    }
    /* if overlap on some node is nonzero, add task to node
      with maximum overlap */
    if max_overlap > 0
        overlap_proc.add(t)
    /*otherwise add task to node with smallest remainder */
    else
        rem_proc.add(t)
    end if
}
```

Figure 2. Offline Algorithm

3.4 Hybrid

The *Hybrid* algorithm combines the flexibility and robustness of the Dynamic algorithm with the Offline algorithm's ability to incorporate task inputs and running times into node assignments. Intuitively, the idea is to assign "large" tasks to nodes offline, since these tasks can benefit the most from the Offline algorithm. "Small" tasks can be assigned to nodes dynamically whenever nodes are idle, allowing the workflow to automatically adjust to varying processor speeds. Intuitively, small tasks can fill in gaps around larger tasks that finish early, and executing them will not unduly delay a larger task that becomes ready later.

We formally define small and large as follows. The *Hybrid* algorithm includes a $Threshold$ parameter that may have any value between 0 and 1. For each task, we compute its total running time (including estimated time needed to download and upload and inputs and outputs). We divide the total running time of each task by the maximum total running time, and if the value exceeds the $Threshold$, then the task is considered "large" and assigned offline. Note that when $Threshold$ = 0, Hybrid is equivalent to Offline, and when $Threshold$ = 1, it is equivalent to Dynamic.

We present the Hybrid algorithm in Figure 3. This algorithm first determines which tasks should be assigned offline and which should be assigned dynamically, then assigns the offline tasks to nodes using the Offline algorithm described above. In our current implementation, offline tasks are given priority and online tasks are assigned to nodes whenever a node becomes idle and there is no offline task ready to execute on the node. In our ongoing work, we are considering prioritization of tasks.

4 Experiments

We now present experimental results that evaluate the performance of the algorithms above. We first present results from simulations that model the performance of the *Isolines*, *Transects*, and *Model Stations* workflows using the statistics presented in Tables 2 and 3. We then present preliminary experimental results that evaluate the performance of the scheduling algorithms on our prototype implementa-

HYBRID (Vector TASKS)

task $longest\_task$
int $longest\_length$ = 0
Vector $Offline\_Tasks$
foreach t in $Tasks$ {
  if ($t.length > longest\_length$)
    $longest\_length$= $t.length$
    $longest\_task$= t
  end if
}
foreach t in $Tasks$ {
  if ($t.length/longest\_length > Threshold$)
    $Offline\_Tasks$.add(t)
  end if
}
  OFFLINE_ASSIGNMENT($Offline\_Tasks$)

Figure 3. Hybrid Algorithm

tion for the same *Isolines* and *Transects* workflows. Recall that these tasks represent only a subset of all the CORIE forecast workflow tasks. Thus, the improvements in the makespan of an entire workflow is expected to be much larger than what is shown in this section.

4.1 Simulation

We implemented our simulation using JavaSim [15], a discrete-event process-based simulation tool. We report on experiments with three nodes using our 1-day subset of a forecast workflow, which models our current implementation. We plan experiments with multiple workflows and additional nodes in future work. We consider both the case where the cost estimates are accurate and all processors have equal speeds, and the effects of variations in processor speeds or task execution times as described in Section 4.1.2. We varied the order that the 16 input files became available to model a real-world scenario where we do not know exactly when each file becomes available and therefore do not know when each task will be ready for execution.

While we report on the use of our simulation tool to evaluate the performance of different scheduling algorithms, a simulation tool has other uses as well. For example, the simulation tool allows us to easily test the effects of adding processors and determine when additional processing power is beneficial, without the cost of adding hardware to a system. A simulation tool also allows us to evaluate the choice of scheduling algorithm and parameter settings for a given workflow. For example, simulations can help us quickly determine a good *Threshold* value for a given workflow, which can aid configuring the actual workflow implementation.

We report on the end-to-end latency of a set of tasks, i.e., the longest running time on any of the three nodes. This metric is appropriate when the goal is to complete a set of related tasks as soon as possible, as is common in many scientific workflows. It also frees up processors as early as possible to allow other data products to be generated. We note that other metrics, e.g., average latency or amount of data transferred, may be useful in some cases, and we are currently investigating these as part of our ongoing work.

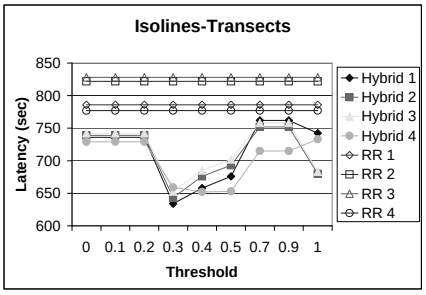
4.1.1 Equal Processor Speeds

We first consider the case where estimated running times are close to actual running times and all three processors have equal speeds. Figure 4(a) shows the performance of Hybrid and Round Robin for four different random orderings of the input files. Recall that when Threshold = 0, Hybrid is equivalent to Offline, and when Threshold = 1, Hybrid is equivalent to Dynamic. As we varied the Threshold from 0 to 1, there were six different assignments of tasks to nodes. In all cases, a threshold between 0.25-0.5 reduces the end-to-end running time compared to Offline (far left) and Dynamic (far right), suggesting that a threshold in this range gives good performance in most cases. For all four random orderings, Round Robin performs worse than either Offline, Dynamic, or Hybrid, in some cases by nearly 200 seconds (over 3 minutes).

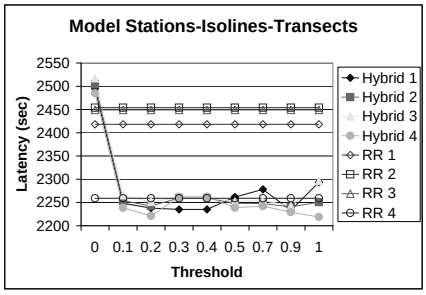
Figure 4(b) shows the performance of Hybrid and Round Robin for Model Stations, Isolines, and Transects. As we varied the *Threshold* from 0 to 1, there were eight different assignments of tasks to nodes. For these tasks Round Robin does significantly worse than Hybrid for most *Threshold* values because Round Robin divides the 146 Model Stations tasks evenly among all three nodes, but their execution is delayed at some nodes due to previously assigned larger tasks. In contrast, Hybrid and Dynamic (*Threshold*=1) can adapt to the varying load at each node and assign the model stations tasks to nodes as they become idle. The Offline algorithm *Threshold* = 0 does even worse than Round Robin because it does not consider precedence constraints. Offline assigns many of the 146 Model Stations tasks to the same node. However, these tasks cannot execute until the `do_station_extraction` task uploads the needed input files, so there is over 300 seconds of idle time at the node before the Model Stations tasks begin executing. This example illustrates the importance of incorporating precedence constraints into node assignments.

4.1.2 Effect of varying task execution speeds

We next consider the effects of varying node speeds. These experiments model likely scenarios where we do not have statistics on new nodes, or where the load on a node at a

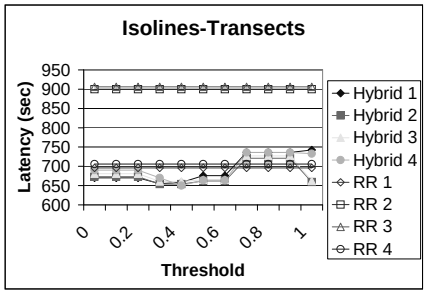


(a)

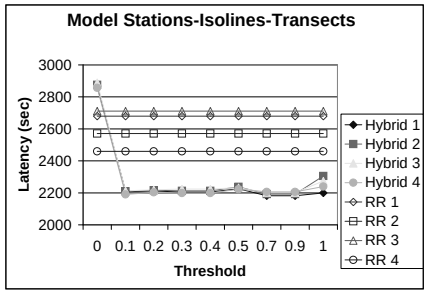


(b)

Figure 4. Effect of varying the Threshold value for the Isolines, Transects and Model Stations tasks with equal processor speeds



(a)



(b)

Figure 5. Effect of varying the Threshold value for the Isolines, Transects and Model Stations tasks with varying processor speeds

given time causes it to run slower than usual. In these experiments, one of the three nodes executed tasks with a factor of 1 (relative to the numbers in Tables 2 and 3), one used a factor of 0.8, and the third used a factor of 1.2.

Figure 5(a) plots the latencies for the *Isolines* and *Transects* workloads. In this case, some Round Robin assignments have low end-to-end latency because longer tasks are assigned to faster nodes. However, in other cases longer tasks may be assigned to slower nodes. Thus, the behavior of RR is unpredictable and may do poorly in the worst case.

Figure 5(b) plots the latencies of ModelStations-Isolines-Transects with varying processor speeds. In this case Round Robin does even worse than when processor speeds are equal. Round Robin does worse because the 146 tasks in Model Stations are divided evenly among the three nodes, but take considerably longer to run on the 1.2 node. Hybrid provides a slight improvement over Dynamic since it exploits the shared input files of *Isolines* and *Transects*.

4.2 Implementation

We now present preliminary experimental results running the *Isolines* and *Transects* tasks on our protype implementation. Our implementation consists of three task-server nodes running on machines connected to the Thetus Publisher by a local network (see Section 2). We summarize the properties of each task server node in Table 4.

Node	OS	Memory	Speed
1	Redhat Linux 9	1GB	2.60GHz
2	Redhat Linux 8	3GB	2.40GHz
3	Fedora Core 1	1GB	2.80GHz

Table 4. Node Properties

We ran experiments using Dynamic, Round Robin, and Hybrid, and we uploaded files using three different random orderings for each of these algorithms (Figure 6). Using three different orderings allows us to evaluate how each algorithm will perform under a variety of circumstances. In our implementation of Hybrid we used a *Threshold* of 0.25, which was empirically validated by our simulations.

The first observation is that Round Robin scheduling may assign many long-running tasks to the same node. In two of the three random orderings the longest end-to-end running time on a node is over 1000 seconds. In contrast, Dynamic and Hybrid assign a more even load to all three

nodes, showing their flexibility. Further, the end-to-end running times are fairly consistent across all three random orderings, suggesting that these approaches do better in the worst case. A second observation is that Hybrid benefits from reduced data transfer overhead compared to Dynamic. Thus, assigning some tasks offline can significantly reduce running times.

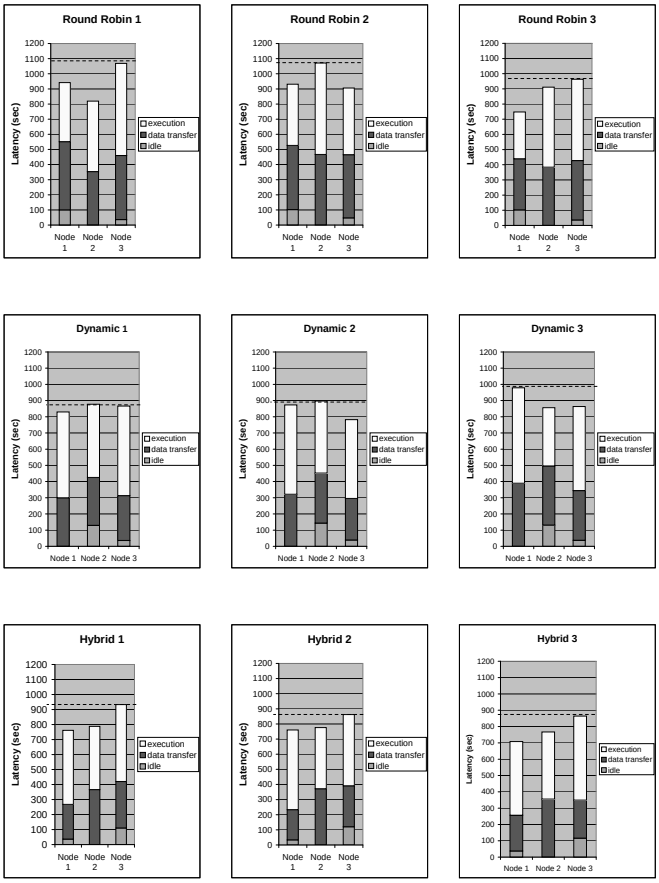


Figure 6. Round Robin, Dynamic, and Hybrid Scheduling for Isolines-Transects for three different file orderings

5 Related Work

There is significant interest in using Grid computing resources to execute large-scale scientific workflows. Systems such as Globus [12] and Condor [17] provide facilities to submit jobs for execution on a set of distributed nodes, but do not consider workflow management. Further, they are designed for a large number of globally distributed users and nodes, and are not well suited to executing a single workflow on a dedicated Grid.

JOSH [25] provides load-aware and data-aware multi-site scheduling, and shares our goal of efficiently executing a set of tasks on a Grid. Clients can submit tasks for execution on one or more nodes. The system chooses a node based on both the cost of transferring the required files to the node and the current load at the node. However, JOSH does not consider storing files locally or global workflow information, so it may make poor scheduling decisions.

There has also been considerable work in replica management for Grid environments [2, 7, 26]. This research aims to improve global data availability, but does not consider storing data products at nodes. Research on replica selection for Grid applications [26] does not consider storing local data copies to reduce workflow execution time or incorporate replica location into scheduling decisions.

Ranganathan and Foster [21] consider the challenges of transferring data files and scheduling tasks for Grid execution. The authors consider executing a set of independent jobs to optimize one or more metrics such as average response time, and also consider placement and maintenance of data copies locally at nodes. However, this work does not consider optimal scheduling of a set of related independent tasks to minimize end to end execution, or adapting to variations in processor and network speeds.

Several tools [1, 3, 13, 18, 23, 24] provide facilities for the specification and execution of scientific workflows. Chimera [13] provides a virtual data system that stores procedures to derive data products. The system schedules these procedures to be executed as needed. The scheduler considers the costs of transferring data files from a remote site compared to the costs of regenerating files locally, but does not consider assigning a set of tasks to nodes to minimize end to end execution time as we do.

Zoo [1] facilitates scientific workflow execution by defining workflows as object-oriented database schemas. The emphasis is on easily specifying workflows by exploiting DBMS functionality, and this work does not consider the challenges of executing tasks on a Grid. More recently, GridDB [18] provides a data-centric overlay for Grid data analysis, and supports task execution on a Grid. However, GridDB does not determine where to execute tasks, instead relying on existing middleware [12, 17], and does not explicitly address the challenges of managing large data files.

Several tools, including Kepler [3], Triana [23], and Taverna [24], provide interfaces and tools to specify and execute scientific workflows. The emphasis of these tools is on formalizing and constructing workflows and providing access to heterogeneous data and distributed web services. In contrast, we focus on the efficient concurrent execution of predefined workflow tasks.

Research in assigning workflow tasks to heterogeneous

nodes [6, 9, 10, 20, 22] shares many of our goals. Pegasus [10] delays assignments of some tasks to cope with changes in node availability, but does not explicitly aim to minimize makespan. GrADS [9] and Sakellariou et al. [22] propose efficient rescheduling algorithms to adapt to inaccurate cost estimates. However, these algorithms have a higher overhead than our hybrid algorithm and do not consider the overhead of transferring data to nodes. Casanova et al. [6] present static scheduling heuristics that consider data location and data transfer overhead, but may perform poorly when there is a wide variation in task execution times.

6 Conclusions and Future Work

The problem of efficiently executing a set of scientific workflow tasks on a local dedicated Grid is important for scientists who need to generate important data products quickly and want to maximize resource utilization. In this paper we have studied algorithms for scheduling and executing scientific workflow tasks to minimize their end-to-end running time using real environmental modeling workflows from the CORIE system. We have evaluated our algorithms using simulations, and have presented preliminary implementation results that validate the simulation results.

We are considering several areas of future work:

Precedence Constraints: We plan to extend our Offline algorithm to handle precedence constraints among tasks.

Priorities: We will consider assigning priorities to sets of data products to meet the needs of scientists who need some products generated as quickly as possible.

Incremental Computation: We plan to develop support for incremental computation of data products to present partial results before an entire data product is computed.

Multiple Workflows: Finally, we plan to consider how to accomodate mutliple workflows running simultaneously on the same set of nodes. Challenges include determining how many nodes to assign to each workflow to minimize the makespan of each and supporting priorities of different workflows.

7 Acknowledgments

This research is supported by NSF grant CCF-0121475.

References

[1] A. Ailamaki, Y. Ioannidis, and M. Livny. Scientific workflow management by database management. *Proc. SSDBM*, 1998.

[2] B. Allcock et al. Data management and transfer in high-performance computational grid environments. *Parallel Computing Journal*, 28(5), 2002.

[3] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludaescher, and S. Mock. Kepler: Towards a grid-enabled system for scientific workflows. *Proc. GGF10 Workshop on Workflow in Grid Systems*, 2004.

[4] A. Baptista, M. Wilkin, P. Pearson, P. Turner, and P. Barrett. Coastal and estuarine forecast systems: A multi-purpose infrastructure for the Columbia River. *Earth System Monitor, NOAA*, 9(3), 1999.

[5] L. Bright and D. Maier. Deriving and managing data products in an environmental observation and forecasting system. *Proc. of the Conference on Innovative Data Systems Research (CIDR)*, 2005.

[6] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. *Heterogeneous Computing Workshop*, 2000.

[7] A. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and scalability of a replica location service. *Proc. IEEE Symposium on High Performance Distributed Computing (HPDC-13)*, 2004.

[8] E. G. Coffman, M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1), 1978.

[9] K. Cooper et al. New grid scheduling and rescheduling methods in the GrADS project. *Proc. IPDPS*, 2004.

[10] E. Deelman et al. Pegasus: Mapping scientific workflows onto the grid. *Proc. 2nd AcrossGrids Conf.*, 2004.

[11] W. Allcock ed. GridFTP protocol specification. *Global Grid Forum Recommendation GFD.20*, 2003.

[12] I. Foster et al. Globus: A metacomputing infrastructure toolkit. *Int'l Journal of Supercomputer Applications and High Performance Computing*, 11(2), 1997.

[13] I. Foster, J. Vockler, Michael Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. *Proc. SSDBM*, 2002.

[14] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.

[15] JavaSim. *http://javasim.ncl.ac.uk*.

[16] C. Jin, D. X. Wei, and S. H. Low. FAST TCP: Motivation, architecture, algorithms, performance. *Proc. IEEE INFOCOM*, 2004.

[17] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. *Proc. ICDCS*, 1988.

[18] D. Liu and M. Franklin. GridDB: A data-centric overlay for scientific grids. *Proc. VLDB*, 2004.

[19] Thetus Publisher. *http://www.thetuscorp.com*.

[20] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. *Proc. IEEE High Performance Distributed Computing*, 1998.

[21] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. *Proc. IEEE High Performance Distributed Computing (HPDC-11)*, 2002.

[22] R. Sakellariou and H. Zhao. A low-cost rescheduling policy for efficient mapping of workflows on grid systems. *Scientific Programming*, 12(4), 2004.

[23] M. Shields and I. Taylor. Programming scientific and distributed workflow with triana services. *Proc. GGF10 Workshop on Workflow in Grid Systems*, 2004.

[24] Taverna project. *http://taverna.sourceforge.net*.

[25] JOSH V1.1. *http://gridengine.sunsource.net/project/gridengine/josh.html*.

[26] S. Vazhkudai, S. Tuecke, and I. Foster. Replica selection in the Globus data grid. *Proc IEEE/ACM Int'l Conference on Cluster Computing and the Grid (CCGRID)*, 2001.

[27] Y. Zhang, A. Baptista, and E. Myers. A cross-scale model for 3d baroclinic circulation in estuary-plume-shelf systems: I. formulation and skill assessment. *Cont. Shelf Res.*, (accepted for publication).

Incorporating Semantics in Scientific Workflow Authoring

Chad Berkley, Shawn Bowers, Matthew B. Jones, Bertram Ludäscher, Mark Schildhauer, Jing Tao
{berkley, jones, schild, tao}@nceas.ucsb.edu, {bowers, ludaescher}@ucdavis.edu

Abstract. *The tools used to analyze scientific data are often distinct from those used to archive, retrieve, and query data. A scientific workflow environment, however, allows one to seamlessly combine these functions within the same application. This increase in capability is accompanied by an increase in complexity, especially in workflow tools like Kepler, which target multiple science domains including ecology, geology, oceanography, physics, and biology. To overcome this complexity, we have developed semantically-driven user-interface components that are customized at run-time using domain-specific ontologies. One such subsystem in Kepler uses domain-specific ontologies to customize the presentation of analytical components and data for use by scientists building workflows. Kepler also provides for semantically-enabled queries for components, which can significantly increase efficiency in workflow authoring tasks. In this demonstration, we show how ontologies can be used for user-interface customization and more. In particular, we show our recent ontology-driven extensions for workflow authoring in Kepler. These extensions include our advances in: (1) automating data-integration and service-composition tasks, (2) the use of semantic annotations to verify that workflows are semantically meaningful, and (3) the ability to search for contextually relevant components and data sets* in situ, *i.e., as a user is designing a scientific workflow.*

1. Introduction

Scientific workflow systems have traditionally been stand-alone applications designed for a specific domain. For example, physicists, geologists, ecologists, and oceanographers typically use their own applications (e.g., a set of "MATLAB" scripts) for creating and executing scientific workflows. The Science Environment for Ecological Knowledge (SEEK) [SEEK] project is developing a powerful, cross-domain scientific-workflow authoring environment that allows scientists to design and execute novel workflows. The need for such a tool has been recognized in other scientific domains, and so SEEK has teamed up with several other projects, including GEON [GEON], SDM [SDM], EOL [EOL] and ROADNet [ROADNET] to produce *Kepler* [KEPLER].

Scientific workflow systems such as Kepler provide scientists with a number of benefits. In particular, they provide an integrated environment in which scientists can design, communicate, and execute their analytical processes. They typically incorporate a variety of functions for end-to-end workflow execution and management, including data query, retrieval, and archiving tools. And, they provide a mechanism to help scientists recreate previous analyses (thus allowing workflows to serve as a form of metadata) and provide an opportunity for workflows (and data) to be reused to form novel and extended analyses.

A major challenge for Kepler is to effectively support users from different scientific disciplines, while maintaining both generic support for scientific workflows and enabling cross-domain data and workflow reuse. Instead of creating complex interfaces and tools for each domain, we desire the capability to provide domain-specific customization. We believe that ontologies can be used not only to formalize domain knowledge, but also to support creation of customized user interfaces, thus facilitating cross-domain interaction.

As part of SEEK (and in collaboration with the other projects previously noted), we are actively engaging scientists to develop ontologies, with the goal of having a rich repository of domain-specific terminologies and cross-linkages among them. Along with this effort, we are also developing a suite of ontology-based tools [BLL04, BL04, BTWL04] to allow scientists to more easily browse, query, integrate, and compose relevant cross-discipline datasets and services. This demonstration will highlight these ontology-enabled tools and their implementation within Kepler.

2. Scientific Workflows and Kepler

A scientific workflow is an executable description of a scientific process. In particular, a scientific workflow records each inline process required to take input data and produce a meaningful output product. Scientific workflows are similar to business-process workflows but have several properties uncommon to the business environment. For example, scientific workflows often operate on large, complex, and heterogeneous data. They can be computationally intensive, and can produce complex derived data products that may be archived for use in re-parameterized runs or other workflows. Moreover, unlike business workflows that are often event-flow driven, scientific workflows are generally data-flow driven (i.e., execution is based on the flow of data as opposed to triggered events).

In Kepler, scientific workflows bring together data and services, possibly created by groups or individuals unknown to each other. Moreover, the workflow applications written in Kepler encompass a wide variety of scientific domains, sub-domains, and specialties. By making these data and services broadly accessible and comprehensible way, Kepler facilitates cross-domain investigations and interdisciplinary research.

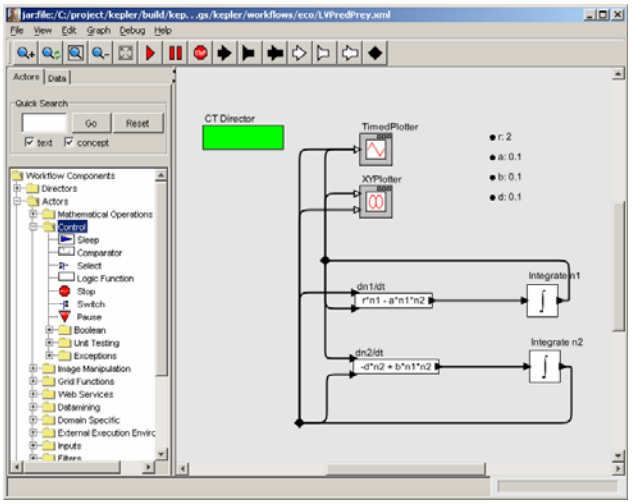


Figure 1. The Kepler scientific workflow environment.

Within Kepler, scientific workflows are authored in a graphical, drag-and-drop manner. Services contain typed ports that can be connected to other services or data sources. Ports can have simple atomic types such as *integer* and *string* as well as more complex structures, including arbitrarily nested array and record types. As a workflow is executing, data passes between ports via tokens that can be readily manipulated to meet the differing syntactic needs of other services. Data produced by a scientific workflow can be displayed graphically at run time, or written to disk for later use.

An example scientific workflow within Kepler is shown in Figure 1. The panel on the left is the "library" where components are categorized and can be searched by a user. When a component is needed on the canvas (the panel on the right), it is dragged from the library onto the canvas where it can then be configured and have its ports connected to other components. The green box controls the timing and flow of the model and can also be selected and drug from the library.

3. Conceptual Challenges in Scientific Workflows

Because Kepler is a powerful and flexible workflow system with a diverse set of users, a number of conceptual-modeling challenges arise. Our goal is to allow users with different backgrounds and varying levels of computing expertise to create new scientific workflows with a minimum amount of difficulty. We highlight below the main difficulties we wish to address.

Supporting high-level conceptual models. Most scientists have a high-level conceptual model of their workflows. If asked, a scientist can typically write down the steps involved in taking raw data and producing their desired output fairly quickly. However, when this conceptual model becomes formalized into an executable scientific workflow, a large number of low-level technical details arise. Details such as file access, network protocols, dataset schemas, service input and output typing, execution models (e.g., tuple-at-a-time versus table-at-a-time dataflow), and configuration parameters tend to obscure the high-level conceptual model of the workflow, making it hard to compare it with existing workflows and reuse it in new settings. We would like to effectively capture the high-level aspects of a workflow, while also preserving but often hiding the underlying technical details.

Basic contextual metadata. A general lack of contextual metadata with respect to data and services is problematic for users (e.g., those who are trying to find new and relevant datasets and services). As an example, a service titled "interpolator" might give one the impression that it provides a generic interpolatation operation over arbitrary datasets when in fact, the service was written to interpolate spatial grid data. Additionally, the same component could simply have been named "int," obscuring the functionality of the service even though those familiar with the particular workflow know that "int" means "spatial data grid interpolate". We face the challenge of making these services generically comprehensible and accessible.

Schema and service-type semantics. Scientific data integration can be a complex and time-consuming process. Scientific data is highly heterogeneous, laden with structural, schematic, and semantic differences. Today, scientific-data integration is typically performed by hand and requires significant "meta" information. Service composition similarly requires considerable contextual information describing structure (to manage heterogeneity in input and output types) and semantics (the kind of objects consumed or produced by a service).

4. Using Semantics in Workflow Authoring

Robust metadata is required to meet the challenges involved in enabling domain scientists to create, run, and share scientific workflows. Several communities continue to have grass-roots organizations that deal with the collection and storage of syntactic metadata. The Knowledge Network for Biocomplexity [KNB] serves the ecological community with the Ecological Metadata Language (EML) [EML] and associated metadata repositories [JBBS01]. Other relevant metadata standards for Kepler include FGDC [FGDC] and Dublin Core [DC], to name a few.

While standards such as EML may provide some support for semantic metadata (e.g., using a "keyword" field), this information is typically not sufficiently formalized for general use in an automated environment. Most current metadata standards for services also fail to include such formal semantics, including the Web-Service

Description Language (WSDL) [WSDL] and the Modeling Markup Language (MoML) [MOML].[1]

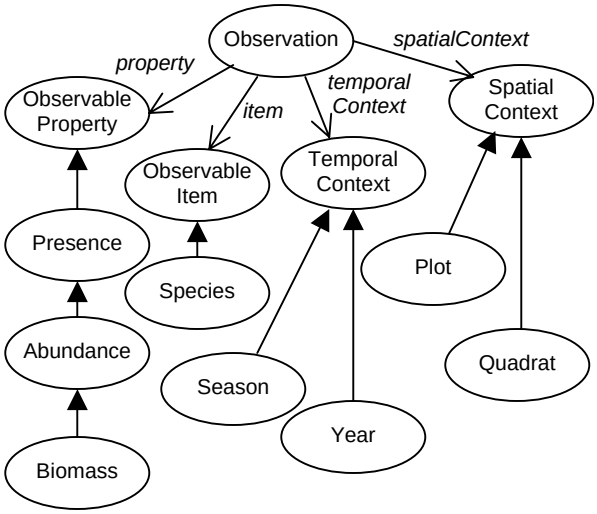


Figure 2: A simplified SEEK ontology.

Kepler has adopted the OWL web ontology language [OWL] (more specifically, OWL-DL) as the primary language for capturing domain-specific terminologies used in semantic metadata. Our approach is to leverage OWL-DL ontologies and *semantic annotations* (described below) of data and services within Kepler to capture rich and possibly complex semantic metadata. A fragment of the SEEK "measurement" ontology is shown graphically in Figure 2.

To address the conceptual challenges discussed in the previous section, we have developed the following features, which we propose to demonstrate. Each of these features leverages the domain ontologies being developed within SEEK and the other Kepler projects.

Support for detailed semantic annotations. Kepler is designed to provide users with the ability to semantically register [BLL04] their dataset schemas and services (and their corresponding input and output types) using *semantic annotations.* Figure 3 gives a set of semantic annotations for the *biom* dataset containing species biomass observations. A semantic annotation defines a relationship between a service or dataset and terms in an ontology. Intuitively, semantic annotations define the "semantic type" of the resource (shown by the statements on the left of the arrows in Figure 3), and link portions of the semantic type to portions of the resource (shown on the right of the arrow in Figure 2). For example, the first annotation in Figure 3 states that tuples in the *biom* dataset denote *Observation* instances from the ontology (in Figure 2). Similarly, the second annotation states that a year value within a tuple denotes the corresponding observation's temporal context and is an instance of the *Year* ontology concept. The semantic annotation language is designed for use at different "granularities," e.g., from selecting a single concept and assigning it to a service, to prescribing a complex ontology instantiation and assigning individual structures within it to particular data values within a dataset (such as in figure 3).[2]

Dataset Schema:
biom(yr, seas, plt, qd, spp, bm)

Semantic Annotations:

x:biom	==>	*x*:Observation
x:biom[yr=*y*]	==>	*x*[*temporalContext*=y:Year]
x:biom[seas=*s*], *s*='W'	==>	*x*[*temporalContext*=s:Winter]
x:biom[seas=*s*], *s*='S'	==>	*x*[*temporalContext*=s:Spring]
x:biom[seas=*s*], *s*='F'	==>	*x*[*temporalContext*=s:Fall]
x:biom[plt=*p*]	==>	*x*[*spatialContext*=*p*:Plot]
x:biom[qd=*q*]	==>	*x*[*spatialContext*=*q*:Quadrat]
x:biom[spp=*s*]	==>	*x*[*item*=*s*:Species]
x:biom[bm=*b*]	==>	*x*[*property*=*b*:Biomass]

Figure 3: Example semantic annotations.

Workflow-component classification and browsing. Kepler leverages semantic annotations to provide customizable access to datasets and services. As shown in Figure 1, the panel on the left displays hierarchically arranged concepts taken from a user-selected ontology, and automatically places services within the hierarchy. This feature provides Kepler users the ability to: 1) select and configure the classification ontology, 2) view the hierarchically arranged ontology (which is computed using a description-logic classifier), and 3) see services classified according to the concept hierarchy (by matching these up through their semantic annotations). In this way, users can easily customize Kepler service presentation (similarly for datasets), and provide ontology-based browsing of data and services.

Semantic scientific-workflow analysis. Given a workflow of interconnected actors, Kepler statically checks (i.e., at design time) whether two connected services (or data sources) are "semantically compatible" based on their semantic annotations, and notifies the user when a connection is not considered semantically well typed. This capability directly assists a user with the workflow creation process.

Ontology-directed scientific-workflow design. As large repositories of workflow components become available,

[1] An exception is the proposed OWL-S [OWL], which provides a "heavy-weight" language for defining the semantics of services.

[2] Semantic annotations in Kepler differ from other approaches by providing rich semantic descriptions that can be "superimposed" over structural types and schemas, allowing explicit connections between substructures and semantic types.

finding relevant resources becomes more difficult. Given a workflow service on the Kepler canvas (the right panel of Figure 1), a user can search for all "semantically compatible" resources (either datasets or services) that can be connected to the input (or output) of the service. This search can also be restricted to return resources that are both semantically and structurally compatible (using Kepler's type system).

(Semi-)Automated Integration and Composition. Scientists often reuse existing workflow components to construct new models. Such components are more often than not structurally incompatible, even though they may be semantically compatible. Our goal is to exploit semantic annotations to derive structural correspondences between input and output data types [BL04]. These correspondences often contain enough information to derive the desired data transformations, allowing scientists to state the desired component connection instead of the low-level details of how those connections should be made. Similarly, multiple datasets must often be combined (i.e., merged or integrated) to be useful as input to a workflow. In this demonstration, we will also show our recent developments for assisting Kepler users in the process of data integration [BTWL04] and service composition, leveraging semantic annotations and domain-specific ontologies.

5. Conclusions and Future Work

In our poster we will show our recent ontology-driven extensions to Kepler for workflow authoring. These extensions include (1) our advances in automating data-integration and service-composition tasks, (2) the use of semantic annotations to verify that workflows are semantically meaningful, and (3) the ability to search for contextually relevant components and data sets in situ, i.e., as a user is designing a scientific workflow. The utility of these extensions will be shown within the context of developing species biodiversity analyses within Kepler.

Acknowledgments

Kepler is based upon the Ptolemy II code base. Kepler includes contributors from SEEK [SEEK], SDM Center-SPA [SPA], Ptolemy II [PTOLEMY] and Geon [GEON]. Work supported in part by NSF ITRs 0225676 (SEEK), 0225673 (GEON) and DOE Grant DE-FC02-01ER25486 (SDM)..

References

[BLL04] Bowers S., K. Lin, and B. Ludäscher, 2004. On integrating scientific resources through semantic registration, In Proc. of SSDBM, pp. 349-352.

[BL04] Bowers S. and B. Ludäscher, 2004. An ontology-driven framework for data transformation in scientific workflows, In Proc. of Workshop on Data Integration in the Life Sciences (DILS), LNCS, vol. 2994, pp. 1-16.

[BTWL04] Bowers S., D. Thau, R. Williams, and B. Ludascher, 2004. Data procurement for enabling scientific workflows: On exploring inter-ant parasitism, In Proc. of Workshop on Semantic Web and Databases (SWDB), LNCS, vol. 3372.

[DC] Dublin Core Metadata Initiative. http://dublincore.org.

[EML] Ecological Metadata Language. http://knb.ecoinformatics.org/software/eml/.

[EOL] Encyclopedia of Life. http://eol.sdsc.edu/.

[FGDC] Federal Geospatial Data Committee. http://www.fgdc.gov.

[SP99] Stockwell D.R.B. and D. Peters 1999. The GARP Modeling System: problems and solutions to automated spatial prediction. International Journal of Geographical Information Science 13 (2): 143-158.

[GEON] The Geosciences Network. http://www.geongrid.org

[JBBS01] Jones, M.B., C. Berkley, J. Bojilova, and M. Schildhauer, 2001. Managing Scientific Metadata, IEEE Internet Computing 5(5): 59-68.

[KEPLER] Kepler: An Extensible System for Scientific Workflows, http://kepler.ecoinformatics.org

[KNB] Knowledge Network for Biocomplexity. http://knb.ecoinformatics.org.

[LL00] E. A. Lee and S. Neuendorffer. 2000. "MoML - A Modeling Markup Language in XML, Version 0.4," Technical Memorandum UCB/ERL M00/12, University of California, Berkeley, CA 94720, March 14, 2000. http://ptolemy.eecs.berkeley.edu/publications/papers/00/moml/.

[OWL] Web Ontology Language. http://www.w3.org/TR/owl-features/.

[PTOLEMY] Ptolemy II, http://ptolemy.eecs.berkeley.edu/ptolemyII/.

[ROADNET] Real-time Observatories, Applications and Data Management Network. http://roadnet.ucsd.edu.

[SDM] Scientific Data Management Center. http://sdm.lbl.gov/sdmcenter/.

[SEEK] Science Environment for Ecological Knowledge. http://seek.ecoinformatics.org.

[WSDL] Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl.

Loose Coupling of Distributed Data and Models through Web Services

Peter McCartney, Robin Schroeder, Corinna Gries
International Institute for Sustainability, Arizona State University
peter.mccartney@asu.edu, robin.schroeder@asu.edu, corinna@asu.edu

Abstract

Arizona State University is developing a local interoperability framework to enable the university and government collaborators to share data and modeling applications. The project focuses on establishing documented interfaces to data and models using XML metadata; defining workflows that transform the outputs of one process into compatible input to another; and implementing these workflows across a distributed network using web services based on Apache SOAP. Point-of-presence (POP) servers at three participating institutions (ASU, Maricopa Association of Governments, and Arizona Department of Water Resources) upon which services for querying data, accessing applications, and transferring data from another POP are configured and manipulated by a remote workflow processor.

1. Introduction

The pace of urban development has presented science, government, and culture with significant challenges to understand, predict and manage the ecological consequences of this global process. It is widely recognized that the questions posed by urban systems exceed the scope of any one agency or discipline, requiring new levels of collaboration. Extensive monitoring datasets and sophisticated models of urban systems have been developed by scientists working for federal, state, and local agencies, often in cooperation with university researchers. Each is designed to collect data and explore a specific urban/environmental system for issues related to the mandate of the sponsoring agency. These models are of value to science both for their insight into social and economic process as well as potential media for ecological monitoring data to be represented in environmental planning and decision-making.

The goal of this project has been the development of an information infrastructure that will promote and facilitate the sharing of data and models among the diverse members of the urban ecological research community in a given region. In a prior project, ASU developed the Southwest Environmental Information Network (SEINet) to promote the use of environmental data archives in research, education, and decision-making [1]. In our current project, we seek to build upon this infrastructure by first establishing a multi-agency network of metadata, data, and application services that can be invoked through an open, platform-independent messaging format. We propose to then use this system to enable the creation and execution of scenarios, or workflows, that loosely couple models by "piping" outputs of one model or data source to the input of another. There are, of course, existing efforts to develop model integration platforms such as the Earth Systems Integration Framework [2]. However, the approach taken in this project presumes that modification of the original model code to conform to a common development framework and to accommodate the necessary data transformations to match their outputs to a priori format requirements is not always an option. Many of the datasets and models maintained by agencies are designed for specific management applications and there are few resources or incentives to make extensive alterations in response to external constraints. Therefore, a goal for this project was to devise an approach to integration that could wrap the model applications and query data archives in their original form even when they are running in different locations, on different operation systems, or in different programming languages.

As a test of this system, we defined a workflow that links output from a global climate model (CCSM2) running at National Center for Atmospheric Research to a hydrology model (MODFLOW2000) running at the Arizona Department of Water Resources and then to a land-use change model (SAM-IM) developed by the Maricopa Association of Governments as an ArcView GIS model (Figure 1). Coupling these models in an integrated workflow involves accessing source data in several formats, executing models in different locations, capturing outputs and transforming them to overcome scalar and semantic differences between the outputs of one model and inputs to another, and moving the result to the appropriate location to be used as input to the next analytic process.

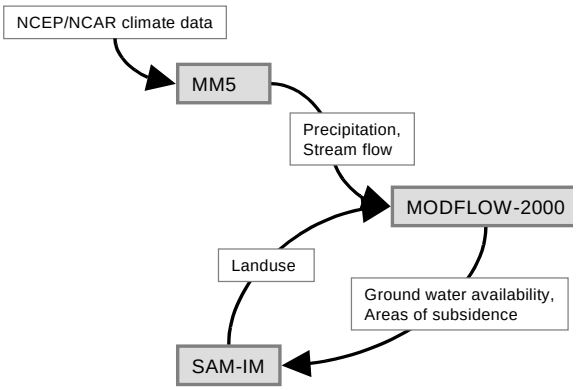


Figure 1. Data flow and coupling of evaluation models.

As depicted in Figure 2, the processing of model outputs can be potentially complex, involving multiple computational steps to reclassify, scale, clip, and merge additional datasets to produce the correct input stream for the coupled model. In the example here, we wish to use raster output from a land-use change model to determine changes in the pump-out rates that are fed into the groundwater model, requiring us to make several changes in the format and semantics of the data. The goal of this project is to expose these functions to a common workflow processing environment that script these steps and generates the specific calls to retrieve data, transform data, execute models, and forward the results to an end-user report or visualization.

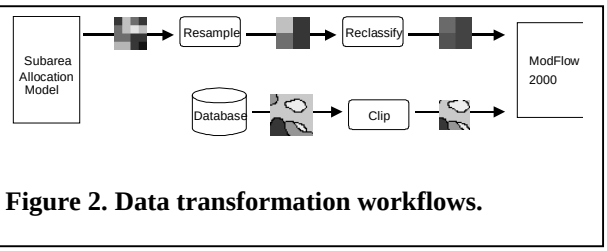


Figure 2. Data transformation workflows.

2. Metadata-mediated interface to data and models

Datasets and models are documented using an XML based metadata schema defined by an open-source, multi-institutional project called Ecological Metadata Language (EML) [3]. EML uses an extensible content model that draws heavily from the Content Standard for Geospatial Metadata [4], the ISO Geospatial Metadata Standard [5], and the National Biological Profile for FGDC Metadata [6]. EML adequately describes the logical structure and physical formatting of tabular, GIS and image data, providing details allowing an application to connect to, query and parse a dataset in most DBMS and GIS archive formats. In this project, we are extending EML's application to models both as a means of documenting a model's outputs as new data, as well as defining the data input requirements of a model. Several existing efforts to define model metadata have focused on indexing and descriptive attributes [7] [8]. In our project we draw from the approach taken by the Earley Suite project [9] to provide metadata that supports the usability of model outputs by defining the settings, inputs, and attributes. With extensions to EML that permit the description of modeling applications, we are able to describe a model's interface both in terms of the data inputs and parameters it requires, and the data file(s) it produces.

Metadata for published data and modeling applications are indexed in a searchable registry using an XML-based query service called *Xanthoria* [10], developed under a previous project at ASU. A client-server based system similar to Z39.50, *Xanthoria* allows multiple metadata catalogs to be searched from a single embedded query client. One feature of the Xanthoria system is that the server component has several connectors that can query existing data sources such as relational databases, raw XML files, and native XML databases such as Exist [11]. A significant goal of the project is to recruit local agencies to participate in a metadata registry network by creating catalogs or connecting existing ones to the *Xanthoria* protocol.

3. Web services

The underlying architecture of the integration network is based on each agency participating as a point of presence (POP) within a local network environment by hosting a compatible set of web services to provide metadata search, data query and processing, and application access (Figure 3). This general architecture was patterned after the National Earthquake Engineering Simulation Grid [12], a grid environment for coupling various models of building stress. The *Xylopia* web services project was created to define some standards for all web services to be located within a pop. Common features include shared registry of network device mounts, a common temporary workspace, code for writing out metadata of the service's processing, and a standardized message structure to create a uniform interface to all application components. A typical workflow begins with a query performed against some dataset in an archive at one of the POP locations. A generic data query service is designed to read the dataset's EML description, resolve a query expressed in an XML message, and execute the query. It returns a package with the resulting data file and a new EML document, which is written to a user workspace at that POP. Subsequent processing, modeling or visualization services are designed to read and write data and generate EML to describe its output. In most cases, the data processing services are actually wrappers to existing software. Open-source packages such as the GRASS GIS [13] and the R Statistical software [14] package are

chosen for the actual computations for several reasons. They are open source and are widely known and trusted. . They work well under Linux, the operating system used for most Xylopia services, and they can be wrapped by Java classes using command- line interfaces.

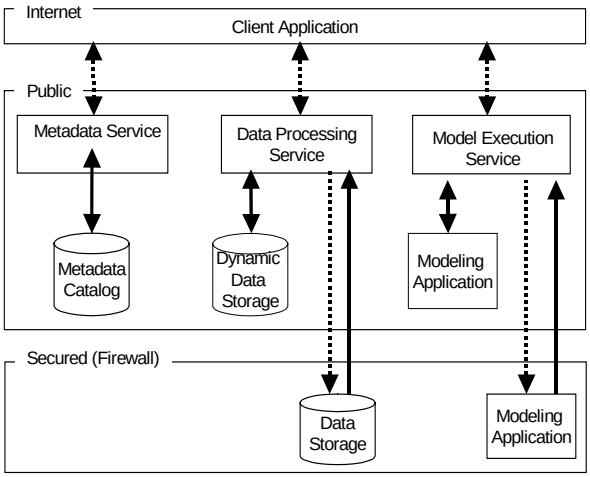


Figure 3. Software and data components of a network Point of Presence (POP). Solid lines represent restricted communication; dashed lines represent SOAP messaging.

When the workflow requires that data be physically moved to another POP location, a transport service packages the data and metadata into a zipped file. A call to the unpack method from the same service at the destination POP causes the file to be retrieved via scp (secure copy) and unzipped into a user workspace on that machine. By distributing the computing services to the remote locations, it is possible to generate workflow scripts that take advantage of the co-location of data and processing code, minimizing the size and frequency of data transfers between two different nodes.

Wrapping the ecological models themselves has proven to be a challenge, for practical rather than technical reasons. The two primary models for this test bed (one a public-domain FORTRAN model, the other a proprietary ArcView GIS overlay model) are both configured and run at their respective agencies by individuals using custom GUI's (one in ArcView, the other in MS Access). Neither system was designed to enable direct access via code, much less from outside the local network firewall. Therefore, our goals for this pilot study have been simply to define and implement web service interfaces for the models following the *Xylopia* specification, with each simply packaging the input parameters and data files and notifying an operator of an execution request. An html interface provides the operator with forms to generate a notification message and post the resulting output so that it may re-enter the workflow when it is completed. Thus, an integrated workflow under this system will have many steps where processing stops and waits for an asynchronous task to complete and post a notification. We are now seeking to mitigate the effect of this asynchronous flow by introducing more indexing and storage services that can manage multiple outputs and formulate batch requests for the subsequent model using outputs from multiple runs of the prior segments of the workflows.

Two important issues affecting an agency's participation in an open data and application sharing system are 1) security and 2) increased load due to external access traffic. To mitigate these two issues during prototype network development, servers were acquired by the project and configured with metadata catalogs and web service components. These were then installed at the two participating government agencies as independent servers from their normal computing resources. Placement of these systems outside firewalls eases limitations on network access and allows agencies to control the communication between the node server and their internal resources.

4. Workflow processing

In developing the prototype for this project, we are making use of workflow processing software based on the Ptolemy II application [15]. An open-source software project called Kepler has been formed by several major NSF-funded projects (including the Science Environment for Ecological Knowledge (SEEK), Geosciences Network (GEON), Biomedical Information Research Network (BIRN), and SCIDAC to develop extensions to Ptolemy for integration with web and grid service components, as well as a broad range of ecological modeling and data functions [16].

Ptolemy provides an XML scripting language (MOML) and a graphical interface (Vergil) in which components are wrapped with small Java classes called actors that define input and output ports for each function. While the original *Ptolemy II* application (and much of the ensuing *Kepler* development) focused on actors that perform or wrap code that is executed locally, ASU has begun the *Encelia* project, creating *Kepler* actors that are specifically designed to manipulate the *Xylopia* web services deployed at the remote POP locations. *Encelia* actors contain SOAP clients for invoking remote services. The input and output ports of *Encelia* actors do not send and receive data directly to and from other actors. Instead, only the metadata describing the output of the prior operation is returned and passed to the next actor (Figure 4). Other parameters take constants at design-time to provide user-specified information as to how to read the data and set any options for the actor's request method. Each remote web service we are developing provides support for distributed workflows though secure shell transfers of data between node drop-

boxes on the network and structured metadata to pass

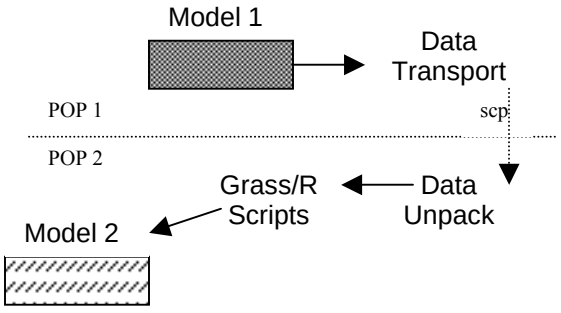


Figure 4. Example Xylopia web service workflow

relative addresses of data to the remote processing service.

5. Future directions

The significance of this approach to integration is that it leverages existing investments in models and data management systems. While we expect to add many other features such as stronger security, job priorities, and logging before the system can be put in production, our test-bed is demonstrating the value of an open, interoperability framework for inter-agency collaboration that we hope will lead to greater incentives for online data publication and sharing. We expect this prototype to lead to an infrastructure that will support more rapid and flexible collaborations between academic and government agencies by reducing the need to move copies of large datasets to new locations or recode existing models to comparable languages.

Acknowledgements. This material is based upon work supported by the National Science Foundation under Grant Nos 9983132 and 0219310. Any opinions, findings and conclusions or recommendation expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

[1] McCartney, P. 2003. SEINet: metadata-mediated access to distributed ecological data. *Databits* Spring 2003. (http://intranet.lternet.edu/archives/documents/Newsletters/DataBits/03spring)

[2] Hill, C., C. DeLuca, V. Balaji, M. Suarez, A. da Silva, and the ESMF Joint Specification Team, "The Architecture of the Earth System Modeling Framework", Computing in Science and Engineering, Volume 6 Number 1, January/February, 2004

[3] McCartney, P.H.; Jones, M.B., "Using XML-encoded metadata as a basis for advanced information systems for ecological research." Information Systems Development II. Eds: Callaos, N., Porter, J., and Rishe, N.. International Institute of Informatics and Systemics.VII, 2002, pp: 379-384.

[4] FGDC 1998. Content Standard for Digital Geospatial Metadata. Federal Geographic Data Committee.

[5] International Standards Organization, 1999. CD 19115, Geographic information – Metadata. Norwegian Technology Standards Institution, Oslo, Norway.

[6] Frondorf, A., M.B. Jones, and S. Stitt, 1999. Linking the FGDC geospatial metadata content standard to the biological/ecological sciences, Proceedings of the Third IEEE Computer Society Metadata Conference. Bethesda, MD. April 6-7, 1999

[7] J. Benz, "What is ECOBAS?", http://dino.wiz.uni-kassel.de/ecobas.html, 2004.

[8] Hill, L.L., S.J. Crozier, T.R. Smith, M. Goodchild, "A Content Standard for Computational Models", D-Lib Magazine, volume 7 number 6, June 2001.

[9] Bouton, K.A., L. Steenman-Clark, "The Earley Suite Numerical Model Metadata Project", http://ugamp.nerc.ac.uk/bouton/model_metadata/index.php

[10] Schoeninger, R.; C. Gries, and P. McCartney. Xanthoria: A SOAP-Based Distributed Query System, International Symposium on Distributed Objects and Applications October 28 - November 1, 2002.

[11]W. Meier. "eXist: An Open Source Native XML Database", Web, Web-Services, and Database Systems A. B. Chaudri, M. Jeckle, E. Rahm, R. Unland (Eds.), NODe 2002 Web- and Database-Related Workshops, Springer LNCS Series, 2593, Erfurt, Germany, October 2002..

[12] R Butler, I. Foster, C. Kesselman, "NEES-POP Concept and Overview", NEESgrid Technical Report 2001-04, September 27, 2001.

[13] M. Neteler, H. Mitasova, Open Source GIS: A GRASS GIS Approach. Second Edition, Kluwer Academic Publishers, Boston, Dordrecht, ISBN 1-4020-8064-6 (Also published as eBook, ISBN 1-4020-8065-4). , 2004, 424 pages

[14] R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, Volume 5, Number 3, 1996, pp:299-314

[15] Lee, E. A., "What's Ahead for Embedded Software?", IEEE Computer, September, 2000, pp. 18-26

[16] Science Environment for Ecological Knowledge, http://seek.ecoinformatics.org

GENESIS SciFlo: Scientific Knowledge Creation on the Grid Using a Semantically-Enabled Dataflow Execution Environment

Brian Wilson, Benyang Tang, Gerald Manipon, Dominic Mazzoni, Eric Fetzer,
Annmarie Eldering, Amy Braverman, Elaine Dobinson, and Tom Yunck

Jet Propulsion Laboratory
Pasadena, CA 91109
Brian.Wilson@jpl.nasa.gov

Abstract

SciFlo is a system for Scientific Knowledge Creation on the Grid using a Semantically-Enabled Dataflow Execution Environment. SciFlo leverages Simple Object Access Protocol (SOAP) Web Services and the Grid Computing standards (WS- & Globus Alliance toolkits), and enables scientists to do multi-instrument Earth Science by assembling reusable Web Services and native executables into a distributed computing flow (operator graph). SciFlo's XML dataflow documents can be a mixture of concrete operators (fully bound operations) and abstract template operators (late binding via semantic lookup). All data objects and operators can be both simply typed (simple and complex types in XML schema) and semantically typed (linked to OWL ontologies). We will demonstrate an early prototype of the SciFlo client and server software and an example dataflow in which atmospheric temperature and water vapor profiles from three Earth Observing System (EOS) instruments are retrieved using SOAP (geo-location query & data access) services, co-registered, and visually & statistically compared on demand.*

1. Introduction

The General Earth Science Investigation Suite (GENESIS) project is a NASA-sponsored partnership between the Jet Propulsion Laboratory, academia, and NASA data centers to develop a new suite of Web Services tools to facilitate multi-sensor investigations in Earth System Science. The goal of GENESIS is to enable large-scale, multi-instrument atmospheric science using combined datasets from multiple Earth Observing System (EOS) sensors on the three EOS satellites: the Atmospheric Infrared Sounder on Aqua (AIRS) [Aumann], the Moderate Resolution Imaging Spectrometer on Terra and Aqua (MODIS) [King], the Multi-angle Imaging Spectrometer on Terra (MISR), and Global Positioning Satellite (GPS) limb sounding [Hajj; GENESIS]. Investigations include cross-comparison of spaceborne climate sensors, cloud spectral analysis, study of upper troposphere-stratosphere water transport, study of the aerosol indirect cloud effect, and global climate model validation. The challenges are to bring together very large datasets, reformat and understand the individual instrument retrievals, co-register or re-grid the retrieved physical parameters, perform computationally-intensive data fusion and data mining operations, and accumulate complex statistics over months to years of data. To meet these challenges, we are developing a Grid computing and dataflow framework, named SciFlo, in which we are deploying a set of versatile and reusable operators for data access, subsetting, registration, mining, fusion, compression, and statistical analysis.

SciFlo is a system for Scientific Knowledge Creation on the Grid using a Semantically-Enabled Dataflow Execution Environment. SciFlo leverages Simple Object Access Protocol (SOAP) Web Services and the Grid Computing standards (WS-* & Globus Alliance toolkits), and enables scientists to do multi-instrument Earth Science by assembling reusable Web Services and native executable into a distributed computing flow (operator graph). The SciFlo client & server engines optimize the execution of such distributed data flows and allow the user to transparently find and use datasets and operators without worrying about the actual location of the Grid resources. The scientist injects a distributed computation into the Grid by simply filling out an HTML form or directly authoring the underlying XML dataflow document, and results are returned directly to the scientist's desktop. Once an analysis has been specified for a granule or day of data, it can be easily repeated with different control parameters and over months or years of data.

In the sections following, we will highlight some of the design issues and solutions adopted in the implementation of SciFlo, including XML dataflow description documents, use of XML datatyping & semantic web ontologies, a parallel dataflow execution engine, data access services using SOAP, and distributed catalog lookup of operator bundles.

2. Technology Obstacles

With present data systems large-scale studies are burdensome and in many cases impractical. Each requires swift access to, selection from, fusion of, and operation upon volumes of incommensurate products from multiple sensors and archives. This invariably requires custom code and deep expertise to accomplish properly.

The goal of GENESIS is to create new tools to facilitate multi-sensor science; exercise them in real scientific investigations; deliver them in an open source toolkit that automates many steps in the process; adopt uniform and flexible standards; and provide a model archive of multi-

sensor data, co-registered and cross-validated. Here we describe several of today's persistent IT bottlenecks and briefly outline our approach to address them.

Data Volume and Access. A single-instrument dataset may contain thousands of files (granules) and terabytes of data. A typical AIRS, MODIS, or MISR swath is broken into many granules, each containing tens or hundreds of parameters. Assembling a regional subset for a specified interval involves subsetting the granules by time and location, retrieving the desired parameters, and aggregating the results.

Time/Geo-Location and Parameter Subsetting. To find overlaps between multiple instrument retrievals, one needs flexible query and subsetting services for time, geolocation, and desired physical variables. Since each EOS granule (data object) has a unique ID, one can separate the data query & retrieval process into several steps: query by time & location for a list of matching granule ID's, subset the parameters in each granule to reduce the size of the files, retrieve the customized granules over the Internet from a remote data center. A key goal of SciFlo is to enable a scientist to develop & push a custom subsetting or data mining operator from her Sciflo server into a server that has local access to a large data archive. SciFlo provides default SOAP services for data query, subsetting, and access:

- GeoLocationQuery – return granule ID's that are near a latitude/longitude point with locations,
- GeoRegionQuery – return granule ID's that intersect a lat/lon region with location metadata,
- FullMetadataQuery – query by all metadata fields (constraints for a SQL where clause),
- FindDataById – return one or more replicas for each granule as a list of pointers (URL's).

The semantics of these SOAP interfaces are general enough to be reused for AIRS, MODIS, MISR, and GPS datasets. The actual retrieval and movement of data files is performed as necessary by the engine, using GridFTP and other protocols.

Coincidence searching. Co-registration operators have been developed to find time & space overlaps between two point-like data objects (e.g., GPS profiles and radiosondes) and point & swath (curvilinear scan grid) objects (e.g., GPS profiles and AIRS swaths). The overlaps are found using the location metadata, without retrieving the actual data. To our knowledge, there are no other services that solve this random access lookup-and-intersect problem in a robust way for large EOS datasets.

3. Features of SciFlo

Here we describe the SciFlo design and its key technologies. (The current prototype does not yet implement all of the planned features.) The design anticipates a large and growing network of SciFlo nodes, at first federated and later connected in a peer-to-peer (P2P) topology. Each SciFlo node is both a client and a server; distributed data queries and searches for operator bundles will be scaled up using P2P protocols (i.e., probabilistic flooding [Banaei-Kashani] or Gnutella [Paolucci]). SciFlo will both move data to the operators and move operators to the data.

Dataflow Execution Engine. At the core of Sciflo is a parallel dataflow execution engine (for other work see Kepler [Altintas] and Chimera [Foster]). The SciFlo engine parses the XML description of the process flow, creates an optimized execution plan, distributes and load balances the computation, parallelizes if possible, and coordinates the execution of each operator. Operators can be local executables or scripts, or remote entities that require the engine to invoke a remote SOAP service. The engine creates the code to move data inputs and outputs between nodes, sequence operations, execute operators in parallel, update a status log, and deliver results.

Data and Operator Types. SciFlo *names* objects and operators hierarchically for simple catalog lookup (e.g., sciflo.data.airs.l2std.granule and sciflo.op.coreg.point2swath), uses XML schemas to describe simple and complex *types*, and uses OWL ontologies to provide higher-level *semantic* typing (kind information via one or more subClassOf properties). We are using and adding to Ontology Web Language [OWL] work in describing space & time and the Earth science domain by [Raskin]. Each data object and operator can be labeled with both a *type* and a (semantic) *kind*. However, all type information beyond the simplest "string versus float" typing is optional. If type and/or kind information are present, then the engine can do type and/or kind checking and missing operator resolution using the types and/or kinds. If type information is missing, execution proceeds at the risk of run-time errors due to dynamic type conversions.

Streaming Binary Data Between Operators. For loosely-coupled computing, small objects can be transferred in binary or XML formats, while larger objects are kept in binary containers (netCDF or HDF files) and referenced by URL's. The Open Data Access Protocol [OpenDAP] is particularly useful in this context. An OpenDAP URL allows one to retrieve over the web a slice of any named matrix in a netCDF or HDF file. Using OpenDAP, SciFlo can efficiently move fine-grained data slices to in-memory operators, while retaining the convenience of self-documenting composite containers.

The SciFlo Document. All flows are specified in SciFlo description language, an XML schema that reuses WSDL to describe each operator. A typical SciFlo document consists of a SOAP envelope, header, text description, authentication data, imports, inputs, outputs, expected resource needs, a list of processes; and a postamble describing how outputs are to be collected and delivered. The document contains the minimum declarative information needed to specify the flow, and it can be simply authored in an XML outline editor. (A visual programming tool will be available but not required.)

Each process in a flow can be another flow (recursive), a normal operator, a source operator to provide input data to initiate the flow, or a conversion operator inserted to

fill a gap. SciFlo has built in operator "binding" support for remote SOAP calls, http GET or POST calls (backward web compatibility); local executables; or python methods. SciFlo is novel in that a single dataflow document seamlessly integrates the disparate realms of remote service calls, executables operating on files, and in-memory python objects and methods (entering or leaving the python realm at will). A referenced WSDL document provides the operator description, with some additional information to handle the translation between XML inputs and the executable's command-line arguments or python method's arguments.

Parallel Computing. The SciFlo execution engine can apply parallelism at many levels:

- Within a single specialized operator
- Create multiple processes or threads on a single node
- Create multiple processes by launching operators or sub-flows on local (slave) nodes
- Invoke a remote operator via a SOAP or http GET call and wait for the results
- Redirect a flow or sub-flow to a server that can access a data source locally or that can efficiently execute a CPU-intensive operator
- Partially evaluate a flow and then redirect
- Invoke a flow multiple times if the source operator(s) yield multiple objects/files that need to be processed (implicit loops over lists).

Flow Execution. Based on the expected execution time and computing resources described in the flow specification, flows are executed in either immediate or queued mode. The execution process consists of:

- *arrival* – flow execution request arrives at server
- *syntax check* – see if document is well-formed
- *queue* – push the flow onto an execution queue
- *dequeue* – move the flow to the execution phase
- *parse* – validate document against XML schema
- *type check* – verify that operator input/output data will be of the correct types/kinds
- *elaborate* – insert implicit unit or format conversions to fill in missing steps
- *plan* – determine the execution plan and annotate the flow document accordingly
- *execute* – execute parallel flow according to plan
- *deliver* – deliver results to the requester.

At first, 'elaborate' will supply minor missing steps using unit and format conversion operators. Later, it will be possible to leave out more substantial steps and have them automatically filled with the "best" operator that can be discovered in a distributed catalog or P2P search [Paolucci; Thaden]. Each SciFlo user can also specify preferences for particular conversion operators.

Server Operation. The execution engine comprises separate processes executing on one node or many, that talk via SOAP; an XML file specifies the node configuration to employ. The server is a SOAP service and can register itself in a UDDI or P2P-distributed catalog; one can then use service discovery to find SciFlo nodes to combine. Thus, SciFlo will function as an adaptive parallel server that can be reconfigured by editing XML files.

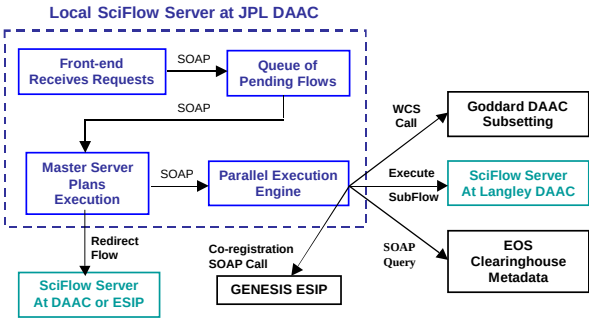


Fig. 1. *Flowchart of SciFlo Distributed Computing.*

Figure 1 depicts a master server and its links to other servers. Front-end nodes field requests, check them, and push them to a queue. A master node will later retrieve the flow and process it through the planning phase. If the plan calls for flow redirection, the annotated flow document is sent to the proper server. If the flow starts locally, parallel execution begins on a master server and may tap multiple CPU's. The figure shows several remote operators executing, starting at JPL. A SOAP query on the EOS Clearinghouse locates datasets and three parallel processes are launched: a subsetting operation on granules from Goddard via a Web Coverage Server call; a custom geolocation and subsetting operation at Langley; and a coincidence search for GPS profiles from GENESIS via a SOAP call. When the operators complete, the datasets are brought together and various data fusion operators complete the analysis.

Server Implementation. Since the server is implemented as a set of interacting SOAP services, each module can be written independently in the programming language of choice. A prototype server has been rapidly developed using python as a "glue" language and for a first implementation of many of the modules. High performance operators are developed by binding C, C++, & Fortran libraries into python, rather than using an all-Java solution. SOAP endpoints are established by using an embedded web server or cooperating with an Apache web server. The primary interface for both the client & server is provided by controlling a web browser.

Parallel Scheduler. The parallel scheduling module is the core of the execution engine. In its full generality the planning problem is quite difficult and involves parallelism within a local computing cluster, and between potentially unavailable clusters or Grid resources. We have begun with fairly hard-wired parallelism for a single-node or small-cluster server. We query loading at each node, use simple rules to assign operators and sub-flows to idle nodes, and use expected operator execution times to predict when nodes will become idle. The scheduler is a SOAP plug-in that could be replaced by existing Grid solutions such as Pegasus [Deelman; Foster].

Implicit Conversions. SciFlo supports several automatic (implicit) conversions: units, formats, and automatic extraction (aggregation) of array slices from (to) composite data containers (HDF files or XML representations). Such conversions are the simplest "missing" steps that can be inserted automatically by constraint resolution on types and tractable semantic inference. Format conversion will support translation of geo-registered 4D data arrays between HDF, XML, and other representations.

Automated Installation of Operators. Existing SOAP methods allow a SciFlo server to locate, request, and install new types, kinds, and operators. If a server passes execution forward, the downstream server can transfer and install needed operators on demand. Thus, a new operator can be installed on the master server to be propagated as needed. Automated transfer of foreign, native executables raises serious security issues, but will be feasible once the WS-Security & WS-Authentication standards are finalized and implementations are robust.

Data Provenance. The traceability of custom SciFlo products is completely specified by the unique ID's of the input data objects, the versions of all of the operators, the elaborated SciFlo document, and the execution log. Thus, data provenance can be maintained by saving snapshots of the entire flow package. Custom products can also be assigned unique ID's by computing a digest from that text package. By organizing data analysis into SciFlo documents, a scientist can use SciFlo as a workbench to track ongoing analyses.

Scalability. Several SciFlo features are designed to foster *automatic* scalability: the server modules are SOAP services so they can be distributed across multiple CPU's or nodes; redundant front-end servers can queue up external requests; and redundant master servers can utilize clusters of slave servers. Large networks of SciFlo nodes will scale by using P2P mechanisms.

Adoption. Every new technology must vie for acceptance. SciFlo confronts this challenge with a lightweight, language-independent framework; loosely-coupled distributed computing; simple declarative programming; processing flows expressed in XML; reuse of WS and Grid Computing standards; an execution engine using parallelism at many levels; standard operator and data types specified by XML schemas and OWL ontologies; open source software; and one-step installation on Linux or Windows clusters. Ease of use and authoring is also promoted by several novel aspects: both concrete and abstract dataflow specification are integrated into a single document schema; almost all "typing" information is optional; more type and (semantic) kind information can be added later as ontologies and rule bases are developed.

4. Future Work

Once the first implementation of the SciFlo core engine is complete and the architecture of the server (SOAP interfaces between modules) is stable, further research questions can be addressed, such as how to: plug in a more sophisticated parallel scheduler; dynamically combine information from multiple ontologies into the semantic inference process; and scale the network of SciFlo nodes using P2P mechanisms.

Acknowledgements. The work described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Altintas, I., C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, S. Mock, Kepler: An Extensible System for Design and Execution of Scientific Workflows, *Proceedings of Scientific and Statistical Database Management*, 2004.

Aumann, H. H., M. T. Chahine, C. Gautier, M. D. Goldberg, E. Kalnay, L. M. McMillin, H. Revercomb, P. W. Rosenkranz, W. L. Smith, D. H. Staelin, L. L. Strow, and J. Susskind, 2003: AIRS/AMSU/HSB on the Aqua Mission: Design, Science Objectives, Data Products, and Processing Systems, *IEEE Transactions on Geoscience and Remote Sensing*, **41**(2).

Banaei-Kashani, F. and C. Shahabi, Criticality-based Analysis and Design of Unstructured Peer-to-peer Networks as Complex Systems, *Third International Scientific Workshop on GLOBAL and PEER-TO-PEER Computing* (GP2PC), 2003.

Deelman, E., J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh and S. Koranda, Mapping Abstract Complex Workflows onto Grid Environments, *Journal of Grid Computing*, vol. 1, 2003.

Foster, I., J. Voeckler, M. Wilde, and Y. Zhao, Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation, *Scientific and Statistical Database Management*, 2002.

[GENESIS] http://genesis.jpl.nasa.gov.

Hajj, G. A., C. O. Ao, B. A. Iijima, D. Kuang, E. R. Kursinski, A. J. Mannucci, T. K. Meehan, L. J. Romans, M. de la Torre Juarez, T. P. Yunck, CHAMP and SAC-C Atmospheric Occultation Results and Intercomparisons, *J. Geophys. Res*., Vol. 109, doi:10.1029/2003JD003909, 2004.

King, M. D., Y. J. Kaufman, W. P. Menzel, and D. Tanr6, Remote Sensing of Cloud, Aerosol, and Water Vapor Properties from the Moderate Resolution Imaging Spectrometer (MODIS), *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING*, VOL. 30(1), Jan 1992.

[OpenDAP] http://www.opendap.org.

Paolucci, M., K. Sycara, T. Nishimura, N. Srinivasan, Using DAML-S for P2P Discovery, *International Conference on Web Services (ISWS)*, 2003.

Raskin, R.G. and M.J. Pan, A Semantic Web for Earth and Environmental Terminology (SWEET), *Semantic Web Technologies for Searching and Retrieving Scientific Data*, Sanibel Island, Florida, Oct 2003.

Thaden, U., W. Siberski, and W. Nejdl, A Semantic Web based Peer-to-Peer Service Registry Network, Technical Report, Learning Lab Lower Saxony, 2003.

A Scientific Workflow Approach to Distributed Geospatial Data Processing using Web Services

Efrat Jäger[†], Ilkay Altintas[†], Jianting Zhang[‡], Bertram Ludäscher[†,§], Deana Pennington[‡], William Michener[‡]
[†]San Diego Supercomputer Center, [‡]University of New Mexico, [§]University of California Davis
{*efrat,altintas*}*@sdsc.edu*, {*jzhang,dpennington,wmichener*}*@lternet.edu, ludaesch@ucdavis.edu*

Abstract

Geospatial analyses of distributed data from surveys and sensors are often stored and managed in diverse regional, national and global repositories. The nature of scientific processes requires composition of these resources in a meaningful order to solve a specific geoscience problem. These tasks can be viewed as scientific workflows. Web based interfaces allow access to remote data and tools, and enable running computational experiments using different online resources. However, it requires manual processing to combine multiple resources in pipelines and scientists still need IT experts to automate their large-scale scientific workflows. The challenging problem is how to enable the scientists to harvest online data and models for designing and executing experiments in a seamless manner. A solution becomes feasible by the introduction of Web Services in a variety of scientific domains. These services can be discovered and composed through generic visual interfaces and scientific workflow tools. For this purpose, we present a complete framework for registering, discovering, composing and executing Web Services to support online science.

1. Introduction

Geospatial analytical functionality is essential to environmental modeling. As large scale distributed geospatial data is available, software reuse and sharing becomes more and more important in integrated environmental modeling, experimenting and analysis. The Service Oriented Architecture (SOA) allows cooperation of data and process components among different organizational units and supports reusability and interoperability of components on the Web, thus increasing the efficiency of assembly and decreasing the cost of development.

In recent years, the need for adaptable interfaces and tools for accessing scientific data and executing complex analyses on the retrieved data has risen in a variety of disciplines (e.g., geology, biology, ecology). Such analyses can be modeled as scientific workflows in which the flow of data from one analytical step to another is described in a formal workflow language. While traditional business workflows are oriented towards document processing, task management and control-flow, scientific workflows typically are data- and/or compute-intensive, dataflow-oriented, and often involve data transformations, analysis, and simulations. Kepler [3, 14] is a system for design, discovery, execution and deployment of scientific workflows from different scientific domains. In this paper, we propose to use the Kepler scientific workflow system to compose geospatial services for environmental modeling. To the best of our knowledge, there exists no previous work on utilizing scientific workflows for geospatial analysis and environmental modeling by composing and executing Web Services in a systematic manner.

We present an approach to provide uniform access to the vast amount of highly heterogeneous services. These services and the related metadata are archived using extensive storage capabilities through registries, and the services are discovered using a metadata description of their operation. We use the Kepler workflow system to demonstrate the registration and discovery process of services within repositories. We further describe how within our framework, services can be composed into scientific workflows and executed to perform scientific tasks. Workflows can also be stored in repositories and shared between scientists. The workflow execution can be monitored to detect and recover from failures, to capture intermediate and end results of the process for data provenance, and to log process information and save execution logs. A challenging issue is to provide a web based access for viewing, executing and sharing scientific workflows and deploying scientific workflows as a new service that can be applicable to other applications.

The rest of the paper is organized as follows. In section 2, we provide a brief overview of distributed geospatial data processing and propose to use Web Services as the building blocks of distributed geospatial data processing within a scientific workflow system. Section 3 introduces the Kepler scientific workflow system's Web Services framework

and the overall architecture. Section 4 presents a running example to illustrate using Kepler to support environmental modeling. Finally, in Section 5, we provide a summary and future work directions. Although the motivating example of this paper is on geospatial data processing, the solutions are applicable to other domains as well.

2. Distributed Geospatial Data Processing

In the early days of computer-aided environmental modeling, geospatial data and process sharing between different machines was available only through manually copying it using mediums, such as floppy disks or CDs. With the development of computer networks, much of this work can be automated using tools and scripts. However, this approach is still inherently labor intensive and requires considerable human interactions which is both inefficient and error prone.

During the past few years, major Geographical Information System (GIS) software vendors (e.g., ESRI [1] and Oracle [6]) expanded their software functionality to provide distributed geospatial data management using mainstream Database Management Systems (DBMS) to support computation remotely in a computer network environment. However, there are several limitations to using a pure vendor-specific DBMS approach to distributed geospatial data processing for environmental modeling, such as cost of ownership, technology complexity and interoperability. More importantly, though database systems are naturally suited for querying/filtering using spatial indexing, they provide poor support for spatial transformations. Spatial transformations are required for transforming between data types or geospatial data values.

Nowadays, some major commercial database systems are beginning to offer some spatial transformations capabilities. However, the underlying ORDBMS model still makes it difficult to apply them to environmental data, since the data is usually unstructured or semi-structured. Therefore, a pure distributed spatial DBMS approach based on SQL-like queries for environmental modeling is undesirable if not infeasible. On the other hand, although Web GIS [12] has put major efforts towards distributed geospatial data processing, adopting the client/server architecture, they are mostly restricted to visualization capabilities, offering little support for complex queryies and analysis.

Furthermore, rendering geospatial data at client side in the form of images or Java/COM object makes the integration and reuse of geospatial data very inefficient. Although using XML as the communication protocol has been proposed for geospatial data integration purposes [15], distributed geospatial data processing that involves data transformation has hardly been investigated.

SOA provides a publishing interface to data and tools using the platform independent Web Service Definition Language (WSDL) [10]. SOA can be used for exposing geospatial data processing methods to the Web.

Within the efforts for standardization of geospatial data formats (i.e. [11, 13]), Geographical Markup Language (GML) [13] is expected to bridge between various data formats. Thus, we propose to use the GML data format and the SOA in distributed geospatial data processing.

We envision that an open architecture is vital for newly emerging integrated and distributed environmental modeling. The architecture should support (1) both flat data (such as operation system files), semi-structured and structured data (such as databases), (2) legacy models written in traditional languages or scripts, and (3) interactive and automatic executions of environmental models in the form of scientific workflows. We believe that using Web Services as the building blocks for geospatial data processing within a scientific workflow system fulfills the above requirements.

In this paper, we propose publishing geospatial data and processes as Web Services and composing them using a scientific workflow approach. In order to efficiently access distributed data and process services, we use web service registries that are accessible through the Kepler system. In the next section we present the Kepler scientific workflow system as our Web Service composition and execution framework to achieve distributed environmental modeling.

3. Kepler Web Service Framework

Kepler [3] is a system for the design and execution of scientific workflows. It is built on top of the PtolemyII system, a modeling and design tool for assembling concurrent components by means of various models of computation [7]. Kepler is an extensible open source scientific workflow system that provides scientists with a graphical user interface to register and discover resources, and to interactively design and execute scientific workflows using emerging Web and Grid-based technologies to distributed computations.

Kepler is unique in that it seamlessly combines high-level workflow design with execution and runtime interaction, access to local and remote data, and local and remote service invocation along with a built-in concurrency control and job scheduling mechanism. Other unique features are inherited from the underlying PtolemyII system, e.g., the ability to combine different models of computations in a single scientific workflow.

Computational units in Kepler are called *actors*, which are reusable components that communicate with each other via input and output ports. Actors are linked to each other to compose a *scientific workflow*. The workflow execution is orchestrated by a *director* that provides the model of computation, that is, scheduling components interaction. In this paper we explain how the Kepler environment can be utilized to discover, compose and execute geospatial data pro-

cessing workflows for environmental modeling, most essentially using a generic Web Service invocation component.

Several generic Web services actors have been implemented in Kepler that serve as clients for accessing distributed resources within Kepler workflows. Specifically, the WebService actor provides a simple plug-in mechanism to execute any WSDL-defined Web Service. An instantiation of the actor acts as a proxy for the Web Service being executed and links to the other actors through its ports. Using this component, any application that can be deployed as a remote service, can be used as a Kepler component.

Other features of the Kepler framework to support Web Service execution are shown in Figure 1 . The figure depicts the Kepler overall architecture for facilitating web service based scientific experiments. The sequence of events involved in performing and analyzing a scientific experiment are as follows. A service in our framework can be a *process service* to perform an analysis operation, or a *data service* to query over a dataset. The geospatial analytical functions that are wrapped as Web Services are process services, whereas services that query different formats of geospatial data are data services. The user or provider publishes scientific datasets and processes. The purpose of registering services is to facilitate their discovery and provide methods for their execution. In the Kepler system, services are registered using domain ontologies and can be discovered by querying over concepts in the related domain ontologies. The user can then access distributed scientific resources by searching and harvesting them. A search can be either syntactic, that is, a text based search by the services names, or semantic by issuing a query against semantic information stored in the registries. Harvesting is facilitated by a *Web Service harvester* to conveniently plug in a whole set of (possibly related) services. Discovered components can be composed to a scientific workflow and may also be registered within the system either as a local component or within domain repositories. The system provides several features for monitoring workflows execution, such as, failure recovery, data and process provenance and post execution processing. Another functionality that is currently under research and development is the deployment of a scientific workflow as a new remote service.

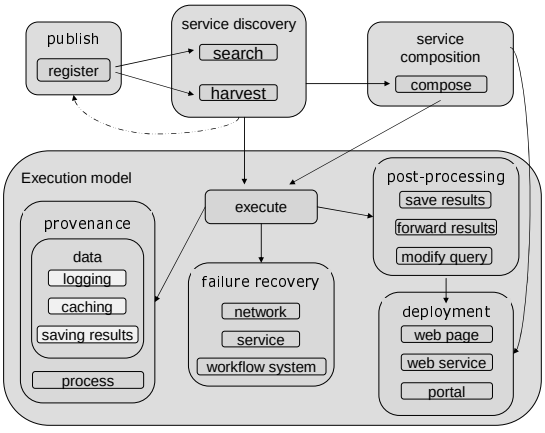


Figure 1. Life-cycle of Kepler Web Services.

4. Geospatial Data Processing Example

The following example in species occurrences analysis is used to illustrate geospatial data processing within the Kepler system. The goal is to find all the occurrences of species A that are within the intersection of the convex hulls of the occurrences of species B and species C. We assume that the occurrences datasets (point data) are stored in three different formats: species A data is stored in a flat file, species B data is stored and managed by SQL Server 2000 and species C data is stored and managed by Oracle 10g with spatial capabilities. We further assume that these three datasets are accessible through Web Services which return a GML representation of the data. Three geospatial functions are used to process the query: GRASS' Convex Hull, Oracle Spatial's Polygons Intersection and a Java Point in Polygon algorithm. These functions are wrapped as Web Services to provide a uniform, domain independent access. Finally, a visualization component is used to display GML documents. This component wraps GeoTools' [2] GML displayer as a Kepler actor. All of these components are registered within a geological repository and are discovered, composed and executed within the Kepler scientific workflow system.

Figure 2 provides a snapshot of the geospatial data processing workflow. During the workflow composition, the user first searches for the desired data and process services. Discovered services appear in top left panel of the Kepler graphical user interface and can be dragged and dropped onto the workflow canvas to perform within a scientific workflow. The services are linked to one another from their visual ports using the GUI. The semantic data transformations in this example are transparent to the user. The datasets are automatically transformed into a GML format while being accessed, using the discovered data services, therefore, no additional intermediate components are required. As for the process services, those were initially designed to consume and produce GML document strings, and thus require no further format integration processing.

As shown in Figure 2, accessing the datasets and the two Convex Hull operations can be done concurrently while the Point in Polygon and Polygon Intersection execute sequentially. Such an execution is feasible in Kepler through a *Process Network director*, (PN) [9], which schedules the workflow execution to a parallel mode (when possible) by creating a separate execution thread for each actor.

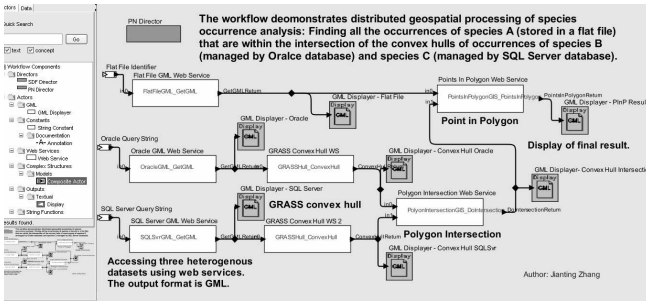


Figure 2. Modeling species occurrences.

GML display actors are used to provide a visualized display of the resulting datasets of each processing step. This actor may also be replaced with a string display actor to display the resulting GML content, or a file writer to store the computation result. Kepler also provides support (as mentioned above) for execution monitoring, such as, a time-out mechanism for connecting to a Web Service, exceptions handling, and enabling to continue halted executions from a certain checkpoint by caching components' computations. The interactive workflow system essentially provides users with a visual programming environment that can effectively harvest, compose, execute and monitor distributed geospatial processes for environmental modeling.

5. Discussion and Future Work

In this paper, we describe a framework for service-based workflows and illustrated the application of it to distributed geospatial data processing for environmental modeling. The Kepler scientific workflow system provides a convenient interface for discovering and harvesting services, and interactively composing and executing them with support for execution monitoring and data provenance.

We demonstrate the feasibility of the proposed approach for geospatial data processing with a simple running example within the Kepler system. The example emphasizes Kepler's strength to facilitate environmental modeling. By publishing geospatial data and processes as Web Services, a unified framework is achieved for accessing various formats and systems. To the best of our knowledge, there exists no previous work on experimenting with geospatial data through scientific workflows, using Web Services as building blocks. Although the focus of this example is towards environmental modeling, the proposed solutions have already been utilized within other scientific domains that uses the Kepler system's web service framework.

As for future work, from a geospatial perspective, we would like to develop a client application to facilitate publishing of datasets as GMLs, including solving efficiency issues when processing very large datasets.

We are also interested in extending Kepler's web service framework in a direction that allows efficient third-party data transfers, has specialized directors (computation engines) for optimized distributed execution for service-based scientific workflows, and provides a role-based Kepler access framework that allows users to create ad-hoc usage groups in order to enable them share their data and processing services through a unified infrastructure.

Acknowledgments

Kepler includes contributors from SEEK [4], SDM Center-SPA [8], Ptolemy II [7] and Geon [5]. Work supported by NSF ITRs 022567 (SEEK), 0225673 (GEON), DOE DE-FC02-01ER25486 (SciDAC/SDM), DARPA F33615-00-C-1703 (Ptolemy), and DARPA N00017-03-1-0090. Special thanks to Kai Lin and the rest of the Kepler developers for their support.

References

[1] ESRI, http://www.esri.com.

[2] Geotools Lite, http://sourceforge.net/projects/geotools/.

[3] Kepler: An Extensible System for Scientific Workflows, http://kepler.ecoinformatics.org.

[4] NSF/ITR: Enabling the Science Environment for Ecological Knowledge (SEEK), www.seek.ecoinformatics.org.

[5] NSF/ITR: GEON: A Research Project to Creat Cyberinfrastructure for the Geosciences, www.geongrid.org.

[6] ORACLE Spatial, http://www.oracle.com/technology/products/spatial/index.html.

[7] PtolemyII, http://ptolemy.eecs.berkeley.edu/ptolemyII/.

[8] Scientific Data Managment Center (SDM), http://sdm.lbl.gov.

[9] The Process Networks (PN) Domain, http://ptolemy.eecs.berkeley.edu/publications/papers/99/HMAD/html/pn.html.

[10] Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl.

[11] D. Green and T. Bossomaier. *Online GIS and Spatial Metadata*. Taylor & Francis, 2002.

[12] A. R. Hardie. The Development and Present State of Web-GIS. *Cartography*, 27(2):11–26, 1998.

[13] W. Kresse and K. Fadaie. *ISO Standards for Geographic Information*. Springer, 2004.

[14] B. Ludaescher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows*, to appear 2005.

[15] J. Zhang, M. Javed, A. Shaheen, and L. Gruenwald. Prototype for wrapping and visualizing geo-referenced data in a distributed environment using XML technology. *ACM-GIS*, pages 27–32, 2000.

Session 4: Grids/Trees

Efficiently Supporting Structure Queries on Phylogenetic Trees *

Susan B. Davidson Junhyong Kim Yifeng Zheng
University of Pennsylvania
susan@cis.upenn.edu, junhyong@sas.upenn.edu, yifeng@cis.upenn.edu

Abstract

With phylogenetics becoming increasingly important in biomedical research, the number of phylogenetic studies is increasing rapidly and huge mount of phylogenetic data has been generated and stored in databases. How to efficiently extract information from the data has become an important research problem.

In this paper, we focus on a class of important queries on phylogenetic trees: structure queries which include least common ancestor, minimal spanning clade, tree pattern match and tree projection. After analyzing the characteristics of the phylogenetic tree as well as structure queries, we propose a storage system based on labeling using RDBMS and design algorithms for query evaluation. We implement these algorithms and compare them with existing techniques. Performance studies prove the efficiency of our strategy.

1 Introduction

Phylogenetics – the science of identifying and understanding evolutionary relationship between different species – has become increasingly important in biomedical research. For example, within epidemiology it has been used to trace contact histories of infectious diseases [9], to identify the geographic origins of outbreaks (as in the case of West Nile Virus [13]), and to predict the timing of new introductions [17].

In response to the demand for phylogenetic trees, the number of phylogenetic studies is increasing rapidly and a variety of phylogenetic tree generation algorithms [12, 6, 10, 26, 29, 19, 18] have been proposed. The number of trees published is doubling every 5 years, and the number of sequences in GenBank that might be used to build trees is doubling even faster, roughly every year [1]. The size and scope of individual trees are also increasing rapidly, as recent publications of trees with hundreds to thousands of species demonstrate.

The growth of phylogenetic information and the need for on-line archival storage and retrieval has led to the establishment of several database systems, most notably TreeBase[23, 22] and Tree of Life[14] (ToL). ToL contains a single tree, and although it is still far from complete it is quite large; the current tree represented in XML format is almost 30MB [14]. TreeBase[23] currently contains more than 3000 trees.

To extract data of interest from these databases, various specialized search tools have been developed. TreeBase provides a keyword-based search tool which allows a user to enter a tree ID, the name of a taxon, or other identifying features to search the database of trees. ToL employs visualization techniques that allow the user to view a section of the tree, expand or contract portions of the tree, and to link to supporting literatures. However, neither of these systems allow users to search the structure of a phylogenetic tree. Since the structure of a phylogenetic tree models the important information about the taxa contained within the tree, structure search is very important.

Recent research efforts have therefore begun to consider structure queries on phylogenetic trees [27, 20]. [27] focuses on pattern match queries: given a query tree (sample phylogeny), find all trees that contain the query structure. Their technique is to decompose the pattern into a set of paths, and try to score the trees in the database with the number of matched paths. However, the method cannot be extended to structure queries whose input does not contain structural information, such as least common ancestor queries: given two species, find their least common ancestor. [20] focuses on least common ancestor and minimal spanning clade queries. By storing each tree edge as a tuple in a relational database, they can translate these queries into SQL expressions using transitive closure (provided by many relational systems, such as Oracle). A major shortcoming of this approach is that transitive closure can be very expen-

* This work was funded by NSF ITR EF 03-31654 entitled "BUILDING THE TREE OF LIFE: A National Resource for Phyloinformatics and Computational Phylogenetics".

sive for large data sets [28].

This paper presents a storage scheme and optimization techniques for efficiently supporting structure queries on phylogenetic trees. The structure queries supported include *least common ancestor*, *minimal spanning clade*, *tree pattern match*, and *tree projection*. Our method is based on a Dewey numbering scheme [30] which encodes the information of the path from the root to a node. Experimental results show that our approach performs well and scales to large data sets.

The outline and contributions of this paper are:

1. In Section 2, phylogenetic trees and structure queries are defined.
2. A labeling scheme based on structure information is presented in Section 3; a database schema based on this labeling scheme is then designed to store phylogenetic tree information.
3. Efficient algorithms for evaluating structure queries are presented in Section 3 using the proposed database schema.
4. Section 4 details the experimental results that demonstrate the efficiency of our strategy.

We close the paper by discussing related and future work.

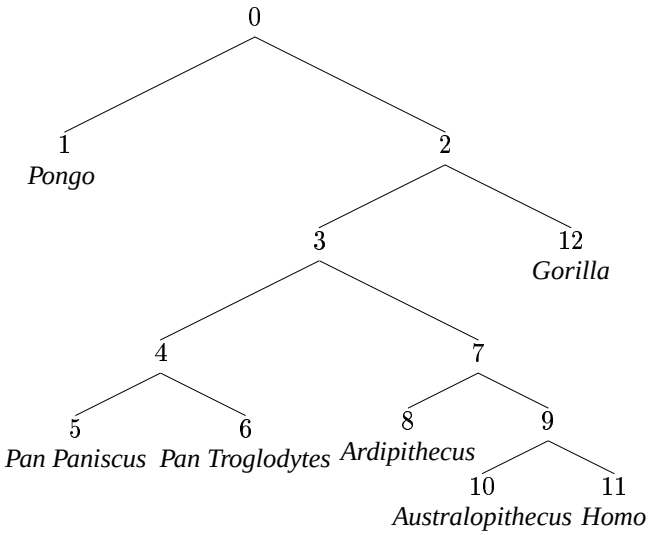
2 Data Model and Queries

A common data model for representing evolutionary relationships between species is a tree. Phylogenetic trees have the following special characteristics and requirements:

1) In theory, phylogenetic trees are unordered binary trees since it is almost impossible for a species to evolve into more than two species at the same time. Occasionally, trees with slightly larger fanout will be built if insufficient information is available; however, this is rare and the fanout is always small. Although it is important to determine if two nodes (species) have the same parent or ancestor in phylogenetic research, there is no obvious biological reason to sort the siblings. Trees are therefore considered to be unordered.

2) The leaves in a phylogenetic trees are tagged. The tag attached to a leaf is always unique, and typically denotes a species name.

3) The information attached to nodes is typically large, representing either sequence information (several million characters) or information about the model used to build this node in the tree.



Each nodes are given a unique identification to facilitate the discussion.

Figure 1: Phylogenetic tree for hominidae[14]

Formally, we can define a phylogenetic tree as follows:

Definition 2.1: A phylogenetic tree T can be defined as a tuple(V Σ, tag, $parent$, root), where

• $V = V_I \bigcup V_L$ is a finite set of nodes where V_I is the set of internal nodes and V_L is the set of leaf nodes

• Σ is a finite alphabet of node tags.

• tag:$V_L \rightarrow \Sigma$ is the *tag function*; $tag(n)$ returns the tag name of n, which can be either a tag or ϵ.

• $parent$:$V \rightarrow \{\epsilon\} \cup V_I$ is the *parent function*; $\rho(n)$ returns the parent node of n if it exists and ϵ otherwise.

• $root \in V$ is root of the tree. ■

Phylogenetic trees may also have more information associated with nodes or edges. For example, the edges may be weighted to represent evolutionary time. Here, we give a very basic model to simplify the presentation.

For example, Figure 1 shows the phylogenetic tree for Hominidae [14] where *tag*($node_{11}$) = "Homo" and *parent*($node_3$) = $node_2$.

Biologists frequently exchange and store phylogenetic trees using the NEXUS [15] format. In the NEXUS format, a pair of parentheses is used to rep-

resent an interior node, a string to identify a leaf node, a comma to separate two sibling, and an optional real number preceded by a colon to denote the weight of the incoming edge of the node. For example, the subtree rooted at $node_7$ in Figure 1 can be represented as (*Ardipithecus*, (*Australopithecus*, *Homo*))). Sometimes, biologists also want to identify the internal nodes and use an extended NEXUS format in which strings are also used to identify internal nodes. For example, if we use *id* to identify an internal node, the subtree rooted at $node_7$ in Figure 1 can be represented as (*Ardipithecus*, (*Australopithecus*, *Homo*)$node_9$)$node_7$.

We also define the ancestor relationship as follows:

Definition 2.2: Given a phylogenetic tree (V, Σ, tag, $parent$) and a node $n \in V$, $ancestor(n) = \{m | \exists l_1 \ldots l_k \in V \ (l_1 = n \wedge l_k = m \wedge \forall 1 \leq i \leq k-1 \ parent(l_i) = l_{i+1})\}$ The function $isancestor(m, n)$ returns true if and only if $m \in ancestor(n)$. ■

Queries.

Structure queries are used to determine relationships between species or to check if a given pattern exists in a given tree.

Least Common Ancestor: Least common ancestor is an important structure query on phylogenetic trees. Although it is not frequently used by biologist directly, it is the basic component for other structure queries. Least common ancestor finds the common ancestor of a set of nodes which is farthest one from the root.

Definition 2.3: Given a phylogenetic tree $T(V, \Sigma, tag, parent)$, the *least common ancestor* of a set of nodes $n_1, \ldots, n_k \in V$, denoted as $lca(n_1, \ldots, n_k) = \{m \mid isancestor(m, n_1) \wedge \ldots \wedge isancestor(m, n_k) \wedge \nexists l \ (isancestor(l, n_1) \wedge \ldots \wedge isancestor(l, n_k) \wedge isancestor(m, l))\}$ ■

For example, we may ask the following query:
Q_1 : Find the least common ancestor of *Homo* and *Gorilla.*
Using the tree in Figure 1, the result would be $node_2$.

Minimal Spanning Clade: Minimal spanning clade is often used when biologists want to find all species which are closely related to the species they are working on. The minimal spanning clade is defined as follows.

Definition 2.4: Given a phylogenetic tree $T(V, \Sigma, tag, parent)$ and a set of nodes $n_1, \ldots, n_k \in V$, the *minimal spanning clade*, denoted as $msc(n_1, \ldots, n_k)$, is the subtree rooted at $lca(n_1, \ldots, n_k)$. ■

For example, Q_2 : Find the minimal spanning clade of species *Homo*, *Gorilla* and *Pan Troglodytes.*
The result is (((*Pan Paniscus*, *Pan Troglodytes*),(*Ardipithecus*, (*Australopithecus*, *Homo*))), *Gorilla*).

Tree Pattern Match: Tree pattern match is used when a biologist knows the relationships (a phylogenetic tree) between a set of species, and wants to find related research on this set of species. We define the tree pattern match as follows:

Definition 2.5: Given a query tree $Q(V', \Sigma', tag', parent')$ and a data tree $T(V, \Sigma, tag, parent)$, *tree pattern match*, denoted as $tpm(Q, T)$, return true if and only if there is a homomorphism from V' to V. That is, there is a function $h : V' \rightarrow V$ such that:

- $\forall v' \in V' \ \exists v \in V, v = h(v')$
- $tag(v) = tag(h(v'))$
- $\forall w', v' \in V' \ w' = parent(v') \rightarrow isancestor(h(w'), h(v'))$

■

For example, we may ask the following query:
Q_3 : Is the pattern ((*Gorilla*, *Ardipithecus*), *Homo*) in the tree of Figure 1?
The result would be false.

Tree projection: Sometimes, biologists are interested in a set of species, but may not know the relationships between them. In this case, they may go to a well known phylogenetic tree, such as TOL, and extract the subtree which only contains the relationship among the species they are interested in. We call this *tree projection* and define it as follows:

Definition 2.6: Given a data tree $T(V = V_I \bigcup V_L, \Sigma, tag, parent)$ and a set of nodes $S \subset V_L$, *tree projection*, denoted as $projection(T, S)$, returns a tree $(V' = V'_I \bigcup V'_L, \Sigma', tag', parent')$ such that $V'_L = S$ and there is a homomorphism $h : V' \rightarrow V$' such that:

- $\forall v' \in V' \ \exists v \in V, v = h(v')$
- $tag(v) = tag(h(v'))$
- $\forall w', v' \in V' \ w' = parent(v') \rightarrow isancestor(h(w'), h(v'))$

■

Note that this differs from minimal spanning clade. For example, Q_4 : find the tree projection with given species *Homo*, *Gorilla* and *Pan Troglodytes.* The result will be ((*Homo*, *Pan Troglodytes*), *Gorilla*) which is different from the result of query Q_2.

Since biologists are typically interested in a small set of species, the result of a structure query is usually relatively small.

3 Evaluating Structure Queries

Based on the properties of phylogenetic trees– in particular the fact that phylogenetic trees are unordered with small fanout – we use the Dewey numbering system [30] which is widely used in library book classification to label nodes and speed up structure queries [1].

The abstraction to phylogenetic trees is as follows: For each node n, we randomly order the outgoing edges and use the order as the label of the edge. Since there is a unique path p from the root to a given node n, we concatenate the labels of edges appearing in p and using the result string as the label for node n.

In this paper, we focus on a binary phylogenetic tree; however, the algorithms also hold for a fixed fanout tree. To clarify the following discussion, we introduce some terminology: Given a node id n, *label*(n) denotes its label, *leftchild*(n) denotes its left child and *rightchild*(n) denotes its right child . Given a label l, *node*(l) denotes the node id. Given two labels l_1 and l_2, *lcp*(l_1,l_2) denotes their longest common prefix.

Ancestor/descendant relationship as well as common ancestors can be determined by comparing node labels as follows:

Ancestor/Descendant Node m is a descendant of node n if and only if *label*(n) is a prefix of *label*(m). Note that n is the parent of m if and only if its label is the string obtained by deleting the last character of $label(m)$.

Common ancestor A common ancestor of m and n has a label which is a common prefix of *label*(m) and *label*(n).

Note that a node label explicitly gives information of the path from the root to this node and therefore uniquely identifies this node.

Example 3.1: Figure 2 shows label information for the sample phylogenetic tree of Figure 1, where each tuple in the table corresponds to a node in the tree in Figure 1, the *tag* attribute represents the node tag information, the *id* attribute is the unique node identification and the *label* attribute in the table stores the label generated for this node. Consider species *Homo*, for which *label(Homo)*=010111. Since *label(Node$_3$)*= 010 which is a prefix of 010111, *Node*$_3$ is an ancestor of *Homo*, but not the parent. Furthermore, since *label(Pan Paniscus)*=01000, *Node*$_3$ is the common ancestor of species *Homo* and *Pan Paniscus* since 010 is a common prefix of 010111 and 01000. ■

[1] A similar scheme is also used in [21].

label	*tag*	*id*
0		0
00	Pongo	1
01		2
010		3
0100		4
01000	Pan Paniscus	5
01001	Pan Troglodytes	6
0101		7
01010	Ardipithecus	8
01011		9
010110	Australopithecus	10
010111	Homo	11
011	Gorilla	12

Figure 2: Relational Representation of Hominidae Phylogeny

Algorithm 1 Tree: labeling(Tree: r, String local)

```
1: if r is null then
2:     return
3: end if
4: if parent(r) exists then
5:     label(r) = concat(label(parent(r)), local) // concat is the function to concatenate two string
6: else
7:     label(r) = local;
8: end if
9: labeling(leftchild(r), "0")
10: labeling(rightchild(r), "1")
```

The labels can be constructed in a single-pass using depth-first traversal of the input phylogenetic tree as presented in Algorithm 1.

Next we will discuss how to evaluate structure queries using this labeling scheme.

Least Common Ancestor.

The least common ancestor of two nodes m and n can be answered by finding the node whose label is the

Algorithm 2 Node: lca(Node: n, Node:m)

1: l_n= *label*(n), l_m= *label*(m)
2: Compute the longest common prefix l of l_n and l_m
3: Return *node*(l)

longest common prefix of *label*(m) and *label*(n). Details can be found in Algorithm 2. Note that *lca(n, m)* can be computed in time proportional to the size of the labels of the input nodes, which are bounded by the height of the tree.

Example 3.2: To answer query Q_1, we will get the labels of *Homo* and *Gorilla*, which are 010111 and 011. The longest common prefix of these two labels is 01. We then determine that $node_2$ has label 01, so *Homo* and *Gorilla* shared the least common ancestor $node_2$. ■

Tree projection.

To find a tree projection from a set of nodes, we first get labels of these nodes and sort the nodes by their labels in alphabetical order. Algorithm 3 can then be used to projection the tree. The algorithm works as follows: Starting with an empty tree T, we insert nodes into the tree in order. Since the order of labels represents the left-right order of the leaves in the data tree, at each point the node being inserted will become the rightmost leaf node in T after insertion. To determine the parent of the new node n in T, we find the first node m on the path from the current rightmost leaf node r to the root such that *label*(m) is a prefix of *label*(n).

Proposition 3.3: Let k be the number of nodes in the input set S. Then the total number of comparison performed by Algorithm 3 is bounded by $3k$.

Proof: Observe that each time we insert a node, the number of comparisons is just one more than the number of nodes removed from the rightmost path. So the total number of comparison = $1 + c_1 + 1 + c_2 \ldots + 1 + c_k$, where c_i is the number of nodes removed from the rightmost path when we insert the ith node. Once a node is removed from the rightmost path, it will be never considered again. So $c_1 + c_2 \ldots + c_k$ is bounded by the size of the result tree which is at most $2k$. Thus, the total number of comparison performed by Algorithm 3 is bounded by $3k$. ■

Example 3.4: To answer query Q_4, we first retrieve the labels of the input species *Homo*, *Gorilla* and *Pan Troglodytes*, which are 010111, 011 and 01001, respectively. We sort this label set and get the list 01001, 010111, 011. We first insert 01001 into an empty tree. To insert 010111, we compute the least common ancestor of 010111 and the rightmost leaf in the current tree (01001) and get the result 010. Since 01001 has no parent, we use 010 as the root of the new tree and get the tree (01001, 010111)010. We then insert 011, and compute the least common ancestor of 011 and the current rightmost leaf (010111) obtaining 01. Since the parent of node 010111 has label 010 which is not a prefix of 01, we must continue up the path to the root of the current tree. However, node 010 *is* the root, so we must create a new root 01, finally, obtaining the tree ((01001, 010111)010, 011)01. Using tags to represent nodes, this is the tree ((*Pan Troglodytes*, *Homo*),*Gorilla*). ■

Algorithm 3 Tree: projection(Node list: $S = (n_1, \ldots, n_s)$)

1: T = null
2: **for** $i = 1, i \leq s, i++$ **do**
3: **if** T is null **then**
4: T = the tree with only one node n_i
5: $r = n$ // use r to record the rightmost leaf
6: **else**
7: lca = lca(r,n)
8: m = *parent*(r)
9: **while** m is not null and *label*(m) is not a prefix of *label*(lca) **do**
10: m = *parent*(m)
11: **end while**
12: **if** m is null **then**
13: *leftchild*(lca) = T
14: *rightchild*(lca) = n
15: $T = lca$
16: **else**
17: *leftchild*(lca) = *rightchild*(m)
18: *rightchild*(lca) = n
19: *rightchild*(m) = lca
20: **end if**
21: $r = n$
22: **end if**
23: **end for**

Algorithm 4 Tree: msc(Node: n, Node:m)

1: l = lca(n, m)
2: Get leaves of the tree rooted by l, $Leaves$, and order them by label
3: projection($Leaves$)

Minimal Spanning Clade.

Using the least common ancestor algorithm, we can find the minimal spanning clade as follows. Given two nodes m and n, we find their least common ancestor l. We then find all nodes a in the tree for which *label(l)* is a prefix of *label(a)*, obtaining as a result a set of nodes.

If the user wishes a tree instead of a set of nodes, we retrieve the leaves for which *label(l)* is a prefix and sort them by their labels. We then projection over them (Algorithm 3) to obtain the tree (see Algorithm 4).

In our implementation (see Section 3), we cluster nodes in a tree by their label and index labels so that matching nodes can be found efficiently. The number of comparisons performed is therefore proportional to the number of matching nodes plus an index scan. Since the result is already sorted, Algorithm 3 can be directly applied to the result to obtain the tree.

Tree Pattern Match.

To answer a tree pattern match query, we also use the projection algorithm (Algorithm 3). Given a tree pattern p, we extract the set of leaves in p. Using the set of leaves as input, we projection a subtree s from the given phylogenetic tree t. We then check whether or not p and s are equal (in the case of an exact match) or compute the difference between p and s as a measure of similarity in the case of approximate match. Algorithm 6 shows the exact pattern match algorithm.

To check if two phylogenetic trees rooted at m and n respectively are the same, we use the property that the leaf tags are unique. The idea is that we perform a depth first traversal of each tree and tag each internal node with the smallest tag of its descendant leaves; this can be done in linear time. Then we compare the tags of the two trees starting at the roots: we first check if the tags of m and n are the same; if not, return false. If they are the same, we recursively check that for each child of m there is a child of n with the same tag (and vice versa). Since the tree is unordered and binary, this entails 3 checks. The total number of comparisons is therefore linear in the number of nodes in the tree. The detailed algorithm is shown in Algorithm 5.

To compute the difference between two trees, we refer readers to [2].

Example 3.5: To answer query Q_3, we first retrieve the leaves of the input tree pattern which are *Gorilla*,

Algorithm 5 boolean: equal(Tree: r_1, Tree: r_2)

```
Function Tree: tagging(Tree: r)
 1: if R is a leaf then
 2:     return
 3: end if
 4: tagging(leftchild(r))
 5: tagging(rightchild(r))
 6: if tag(leftchild(r)) ≤ tag(rightchild(r)) then
 7:     tag(r) = tag(leftchild(r))
 8: else
 9:     tag(r) = tag(rightchild(r))
10: end if
Function boolean: Compare(Tree: r1, Tree: r2)
 1: if r1 is null then
 2:     if r2 is null then
 3:         return true
 4:     else
 5:         return false
 6:     end if
 7: else
 8:     if r2 is null then
 9:         return false
10:     end if
11: end if
12: if tag(r1) != tag(r2) then
13:     return false
14: end if
15: if Compare(leftchild(r1), leftchild(r2)) then
16:     if Compare(rightchild(r1),rightchild(r2))
            then
17:         return true
18:     else
19:         return false
20:     end if
21: else
22:     if Compare(leftchild(r1), rightchild(r2)) then
23:         if Compare(rightchild(r1), leftchild(r2))
                then
24:             return true
25:         else
26:             return false
27:         end if
28:     else
29:         return false
30:     end if
31: end if
Function boolean: equal(Tree: r1, Tree: r2)
 1: tagging(r1)
 2: tagging(r2)
 3: return Compare(r1,r2)
```

Algorithm 6 boolean: pattern-match(Tree: P)

1: get the leaf set of P: S
2: Get the label of element in S and order S by label
3: T=projection(S)
4: **if** equal(T,P) **then**
5: return true
6: **else**
7: return false
8: **end if**

Ardipithecus and *Homo*. Applying the projection operation, we get a subtree ((*Ardipithecus*, *Homo*), *Gorilla*). Applying the tree equality function of Algorithm 5, we find that ((*Ardipithecus*, *Homo*), *Gorilla*) is not the same tree as the input pattern ((*Gorilla*, *Ardipithecus*), *Homo*), therefore, we return false. ■

4 Experimental Results

To evaluate the effectiveness of our method we built a prototype system using C++ and a leading commercial relational database system. We generated phylogenetic trees using r8s [25]. Based on Algorithm 1, a data loader parses the phylogenetic trees, generates a tuple for each node in each tree, and stores them in the database. The schema of the database is <*tid, label, tag*> where *tid* is used to distinguish different trees, *label* is the label of the node, and *tag* records the name of the species which is used to uniquely identify a node. The relation is clustered by {*tid*, *label*}. An index on {*tid*, *tag*} is also built to improve performance. We study the performance of our method and compare it with two related systems. Experimental results show the effectiveness of our approach.

4.1 Experimental Setup

The experiments were performed on a 1.5GHz Pentium 4 machine running Linux, with 512MB memory and one 40GB hard disk (7200rpm). The database is installed on another machine with the same configuration and running Windows 2000. All experiments were repeated 10 times, and the average processing time was calculated disregarding the maximum and minimum values.

We compare the performance of our system with two other systems processing structure queries: [20], which is based on the transitive closure primitive provided by the relational database and denoted as TC; [27], which is based on path match and denoted as PM. We denote our method as LS (labeling scheme).

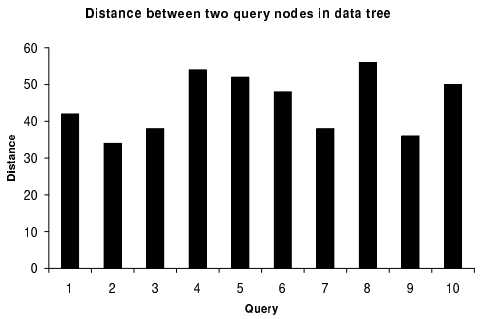


Figure 3: Distance between input pair nodes in the data tree

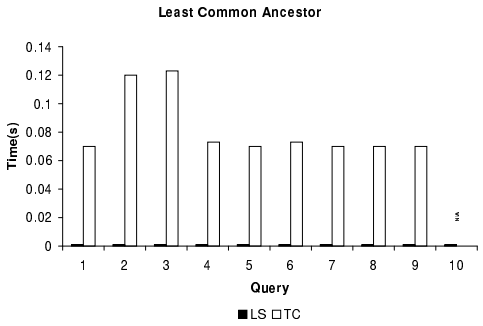


Figure 4: Execution time of LCA queries

Note that PM is a main memory algorithm, while TC and LS need to visit databases.

4.2 Least Common Ancestor (LCA)

The first experiment tests the least common ancestor query. Due to the lack of availability of large phylogenetic trees, in this experiment we use r8s [25] to simulate a phylogenetic tree with 0.5 million nodes according to a Yule stochastic process. The size of data and indexes for TC and LS are 35MB and 50MB respectively. We randomly pick 10 pairs of nodes as input; the distance between the node pairs is show in Figure 3. Figure 4 shows the execution time of least common ancestor queries; the database connection time is not included. Note that the PM method is absent in this test since it cannot support LCA queries. Here as well as throughout the rest of this section, whenever a method cannot support a particular query it will be omitted from the performance graphs.

We can see that both TC and LS work well. LS is based on string comparisons on labels which run very fast and cannot be accurately measured; we use 0.001 seconds as its running time. TC takes less than 0.14 second to run each query except for query 10. It is interesting to see that there is no clear relationship between the running time of the transitive closure

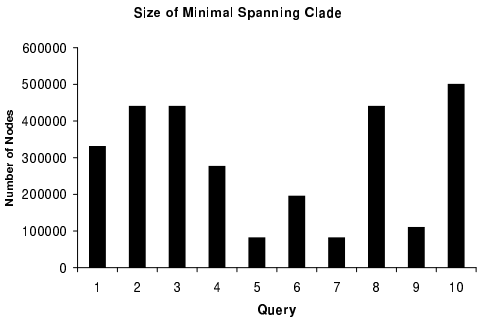


Figure 5: Number of nodes in the result MSC

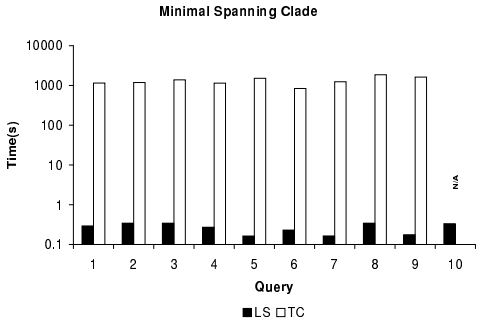


Figure 6: Execution time of MSC queries

method and the distance between two input nodes. For query 10, since the least common ancestor of the two given nodes is the root, and the root has not been stored as a tuple in TC, the result is not available.

4.3 Minimal Spanning Clade (MSC)

The next experiment tests the performance of minimal spanning clade queries. We use the same data and query set as LCA. The number of nodes in the resulting minimal spanning clade is shown in Figure 5. The query execution time is shown in Figure 6.

As we can see, except for query 10 which TC doesn't support, our method outperforms TC by several orders of magnitude: LS takes less than 0.35 second to find the result set while TC takes around 1000 seconds. It is curious as to why TC performs so differently for LCA and MSC queries. The reason is that in a tree, each node has only one parent, so transitive closure for LCA can be implemented as a set of recursive selections. However, since a node can have a set of children, the transitive closure for MSC must be implemented as a set of recursive joins, which are very expensive. Also the execution time of transitive closure has no clear relationship to the size of the query result.

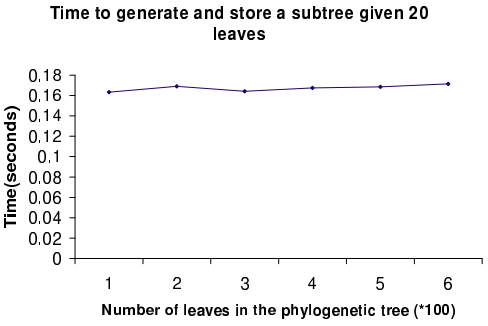


Figure 7: Time to projection a subtree with a given number of leaves from different sized phylogenetic trees

4.4 Tree projection

We also did two experiments to test the performance of our method on tree projection queries. Since TC and PM don't support tree projection, we only have one curve in the experimental results.

Effect of varying the size of the phylogenetic tree

In the first experiment, our target is to understand how our algorithm scales. That is, to determine the effect on the execution time of projection when the number of input leaves is constant, the size of the phylogenetic tree varies, and the leaves are randomly chosen over the phylogenetic tree. To do so, we simulate phylogenetic trees with 100, 200, ..., 600 leaves using HyPhy-II [24].

As shown in Figure 7, the time of projecting a subtree with a given set of leaves is not really affected by the size of the phylogenetic tree.

Effect of varying the number of selected leaves

In the second experiment, we try to understand the effect on the execution time of projection as the number of leaves varies. To measure this, from a fixed phylogenetic tree we randomly select sets of leaf nodes varying the number of nodes. We use HyPhy-II [24] to simulate a phylogenetic tree with 2000 leaves as input and vary the number of leaves selected (10, 20, ... 200 leaves).

In contrast to Figure 7, the time of generating a subtree with a given set of leaves is affected by the number of selected leaves.

4.5 Tree Pattern Match

In the last experiment, we test the performance of tree pattern match queries. We simulate a phylogenetic tree with 200 nodes using HyPhy-II [24] as the data tree. We randomly select 10, 20, ..., 60 leaves and

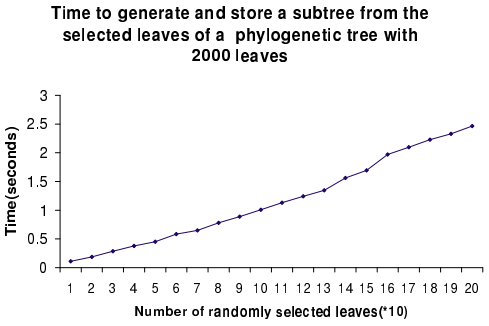


Figure 8: Time to projection a subtree with different number of leaves from a phylogenetic tree with 2000 leaves

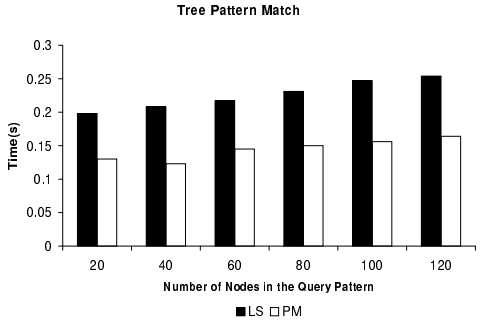


Figure 9: Execution time of pattern match queries

projection a set of subtrees. We use this set of subtrees as query trees. The result is shown in Figure 9. As we can see that, our method based on database engine is comparable to the main memory algorithm PM.

5 Related work

Several database systems [23, 14] have recently been created to store and retrieve phylogenetic trees. The Tree of life [14] is a resource which provides a uniform and linked framework to browse information on phylogenetic relationship as well as various characteristics of organisms, and provides links to related information available on the Internet.

TreeBASE [23, 22] uses a relational database to store phylogenetic trees and the data matrices used to generate them from published research papers. The phylogenetic trees themselves are stored as a BLOB attribute in NEXUS format [15] while other information (for example, the author of the tree) is stored as attributes. Keyword-based queries on attributes other than trees are supported. [27] proposes a method to extend TreeBASE to support structure-based queries (e.g. finding a particular tree pattern) by navigating the NEXUS format phylogenetic tree files using a general-purpose program language.

The next version of TreeBASE will enable structure-based queries by storing the tree structure explictly using the technique of [20]. By storing each edge in the tree explicitly, least common ancestor (LCA) queries can be computed using the transitive closure primitive supported in many commercial relational database systems.

[20, 21] also discuss the requirements of a phylogenetic tree database.

LCA has been well studied in the algorithms literature [8, 4, 3]. [8] describes the first linear preprocessing time, linear space, and constant query time algorithm for LCA. [7] observes that LCA is equivalent Range Minimum Query (RMQ) by giving a linear time algorithm to reduce LCA to RMQ using depth-first search, and a linear time algorithm to reduce RMQ to LCA using a cartesian tree construction. Based on the reduction of LCA to RMQ, [4] gives a linear preprocessing time, linear space, and constant query time algorithm to answer LCA. This algorithm is simpler than the algorithm in [8], and is in turn simplified by [3]. All of these algorithms need to randomly access several different data structures in an interleaved manner, and do not extend well to the database context.

Several techniques have also been developed for manipulating partially ordered sets and ontologies [11, 5, 16]. These techniques are good for processing small graphs (trees) in main memory, and have specialized operations for this application domain.

6 Conclusion and Future work

In this paper, we summarize several important structure queries on phylogenetic trees. Based on an analysis of the characteristics of phylogenetic tree and of structure queries, we proposed a storage system based on a Dewey labeling scheme. We then discuss how to efficiently evaluate structure queries. Our experiments show that this implementation using a relational engine has very good performance and scalability.

In ongoing research, we are investigating structure query operations among multiple phylogenetic trees, such as unions, difference and joins, and how to extend our techniques to support queries on more complex biological data, such as biopathways. In the future, we plan to investigate more general query languages which contain these basic operations. We are also interested in update operations on phylogenetic trees, permitting local rearrangements of phylogenetic trees, and facilitating the curation of phyloge-

netic data.

This work is being performed in the context of the Cyberinfrastructure for Phylogenetic Research (CIPRes) project funded by NSF (http://www.phylo.org/), and will be used for a massive simulation database representing a "gold standard" against which phylogenetic tree reconstruction algorithms can be tested.

7 Acknowledgments

We would like to thank Sampath Kannan for many valuable discussions, and ours reviewers for their constructive comments.

References

[1] Assembling the tree of life. http://research.amnh.org/biodiversity/.

[2] N. Amenta, F. Clarke, and K. S. John, editors. *A Linear-Time Majority Tree Algorithm.*, 2003.

[3] M. A. Bender, G. Pemmasani, P. P. Sumazin, and S. Skiena. Finding least common ancestors in directed acyclic graphs. In *Proceedings of SODA*, 2001.

[4] O. Berkman, D. Breslauer, Z. Galil, B. Schieber, and U. Vishkin. Highly Parallelizable Problems (Extended Abstract). In *Proceedings of STOC*, 1989.

[5] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-chneider, and L. A. Stein. Daml+oil (march 2001) reference description. http://www.w3.org/TR/daml+oil-reference.

[6] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., 2003.

[7] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of STOC*, 1984.

[8] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors . *SIAM Journal on Computing*, 13(2):338–355, 1984.

[9] D. Hillis and J. Huelsenbeck. Support for dental hiv transmission. *Nature*, 369:24–25, 1994.

[10] J. Huelsenbeck and D. Hillis. Success of phylogenetic methods in the four-taxon case. *Syst. Biol*, 42:247–264, 1993.

[11] M. Install. Partially ordered set. http://mathworld.wolfram.com.

[12] J. Kim and T. Warnow. Tutorial on phylogenetic tree estimation. citeseer.nj.nec.com/254275.html.

[13] R. S. Lanciotti, J. T. Roehrig, and V. Deubel. Origin of the west nile virus responsible for an outbreak of encephalitis in the northeastern united states. *Science*, 286:2333–2337, 1999.

[14] D. Maddison and W. Maddison. The tree of life web project. http://tolweb.org/tree/phylogeny.html.

[15] D. Maddison, D. Swofford, and W. Maddison. NEXUS: an extensible file format for systematic information. *Syst. Biol*, 46:590–621, 1997.

[16] D. L. McGuinness and F. van Harmelen. Owl web ontology language. http://www.w3.org/TR/owl-features.

[17] K. McGuire, E. C. Holmes, G. F. Gao, H. W. Reid, and E. A. Gould. Tracing the origins of louping-ill virus by molecular phylogenetic analysis. *Journal of General Virology*, 79:981–988, 1998.

[18] B. Moret, J. Tang, L.-S. Wang, and T. Warnow. Steps toward accurate reconstructions of phylogenies from gene-order data. *J. Comput. Syst. Sci.*, 65:508–525, 2002.

[19] B. Moret, L.-S. Wang, and T. Warnow. Toward new software for computational phylogenetics. *IEEE Computer*, 35:55–64, 2002.

[20] L. Nakhleh, D. Miranker, F. Barbancon, W. H. Piel, and M. Donoghue. Requirements of Phylogenetic Databases. In *Proceedings BIBE 2003*, 2003.

[21] R. D. M. Page. Phyloinformatics: Towards a phylogenetic database. *Data Mining in Bioinformatics(in press)*.

[22] W. H. Piel, M. J. Donoghue, , and M. J. Sanderson. TreeBASE: A database of phylogenetic information. In *In Proceedings of the 2nd International Workshop of Species*, 2000.

[23] W. H. Piel, M. J. Donoghue, , M. J. Sanderson, M. Walsh, T. Eriksson, C. Henze, and K. Rice. Treebase: a database of phylogenetic knowledge. http://www.treebase.org/treebase/.

[24] S. L. K. Pond, S. V. Muse, and S. D. Frost. Hyphy. http://www.hyphy.org/.

[25] M. Sanderson. r8s. http://ginger.ucdavis.edu.

[26] M. Schoniger and A. Von Haeseler. Performance of maximum likelihood, neighbor-joining, and maximum parsimony methods when sequence sites are not independent. *Syst. Biol*, 44(4):533–547, 1995.

[27] H. Shan, K. G. Herbert, W. H. Piel, D. Shasha, and J. Wang. A Structure-Based Search Engine for Phylogenetic Databases. In *Proceedings of SSDBM*, 2002.

[28] S. Shi, E. Stokes, D. Byrne, C. Corn, D. Bachmann, and T. Jones. An enterprise directory solution with db2. *IBM Systems Journal*, 39, 2000.

[29] J. Tang and B. Moret. Scaling up accurate phylogenetic reconstruction from gene-order data. *Bioinformatics*, 19:305–312, 2003.

[30] V. Vesper. Let's do dewey. http://www.mtsu.edu/ vvesper/dewey.html.

Co-Scheduling of Computation and Data on Computer Clusters

Alexandru Romosan, Doron Rotem, Arie Shoshani and Derek Wright *
Lawrence Berkeley National Laboratory,
University of California,
Berkeley, California 94720

Abstract

Scientific investigations have to deal with rapidly growing amounts of data from simulations and experiments. During data analysis, scientists typically want to extract subsets of the data and perform computations on them. In order to speed up the analysis, computations are performed on distributed systems such as computer clusters, or Grid systems. A well-known difficult problem is to build systems that execute the computations and data movement in a coordinated fashion. In this paper, we describe an architecture for executing co-scheduled tasks of computation and data movement on a computer cluster that takes advantage of two technologies currently being used in distributed Grid systems with relatively modest enhancements to these systems. The first is Condor, that manages the scheduling and execution of distributed computation, and the second is Storage Resource Managers (SRMs) that manage the space usage and content of storage systems. The system is capable of dynamically load balancing by replicating popular files on idle nodes. To confirm the feasibility of our approach, a prototype system was built on a computer cluster. Several experiments based on real work logs were performed. We observed that without replication compute nodes are underutilized and job wait times in the scheduler's queue are longer. This architecture can be used in wide-area Grid systems since the basic components are already used for the Grid.

1 Introduction

It is typical of scientific investigations to have two phases: the data generation phase, and the data analysis phase. The data generation phase is usually the result of running large simulations or the collection of data from experiments. Using modern computer systems the amount of data generated in the data generation phase is massive, on the order of terabytes to petabytes. In the data analysis phase, the scientist typically wants to extract a subset of the data based on some criteria. For example, a simulated climate modeling dataset may have data over the entire globe, with multiple height levels for tens of variables, such as temperature, humidity, wind velocity, etc. This large simulation dataset is usually stored on some mass storage system, such as IBM's High Performance Storage System (HPSS). During the analysis phase, a scientist may want to select only temperature over the equator for sea-surface level for 100 years. This requires some way of selecting the files that contain the relevant data, and co-schedule the data movement and computations, by downloading the relevant files to the analysis system before processing can proceed.

Another example of the need for co-scheduling of computation and data movement involves Particle Physics data mining and analysis of detector data. The Data Acquisition System in these detectors records information about collision events between particle beams. The information is stored in multiple files, where each file contains information about thousands of such events. Typical analysis of data involves searching for rare and interesting processes and is performed in multiple phases involving classification and summarization. In addition, the same data files may

*Visiting LBNL from the Computer Sciences Department, University of Wisconsin

be shared simultaneously by several different groups of scientists with different interests.

In general, the problem discussed here is that of effective scheduling of a collection of jobs, each requiring one or more input files to run on a group of servers. Each server in the cluster may have one or more compute slots and a disk cache that can hold some fraction of the data files needed as input for the analysis. A given job can be scheduled on a selected server if: (i) the server has at least one available compute slot; (ii) all the data files needed by the job are available on the disk cache at that server. This introduces the problem of scheduling data movement in coordination with scheduling of computation on a cluster, and the software systems to execute and monitor the schedules. Data movement may be cheap (from one server on the cluster to another) or expensive (from a remote archive).

The above two phases of operations reflect the manner in which most scientific applications run. To speed the data analysis phase, the analysis is partitioned into parallel jobs, and distributed to multiple compute systems. In a Grid environment the compute systems are distributed over the wide area network, and the data sources are usually on remote storage systems. In order to perform the parallel analysis, the data have to be moved to the compute nodes, and the jobs scheduled on these nodes. There are Grid middleware components designed to schedule compute jobs on distributed nodes, and components designed to manage storage and move files between nodes. However, there are currently no components that perform co-scheduling the data and the computation, in part because of the complexity of such middleware systems. Developing a real practical system to perform co-scheduling is one of the most difficult challenges in the Grid domain. We address this challenge in this paper.

We describe a system that was developed to perform co-scheduling of data and computation by taking advantage of two technologies used in distributed Grid systems. The first is Condor [1], that manages the scheduling and execution of distributed computation, and the second is Storage Resource Managers (SRMs) [2] that manage the space usage of storage systems and the dynamic content of the storage. In order to have a controlled experimental environment, the system was developed on a small cluster of workstations that do not share memory, and have their own independent attached disks. In order to achieve co-scheduling, some modifications to Condor and SRMs had to be made, but as will be discussed next, we achieved coordination between these systems with relatively modest enhancements.

1.1 Organization of paper

The rest of the paper is organized as follows. In Section 2 we explain the main contributions of this work emphasizing the ability to build a complex co-scheduling system by using existing mature components. In Section 3 we describe the architecture of the co-scheduling system and the software modules developed for this project. In Section 4 we describe the replication algorithms used for evaluating our system. In Section 5 the data used in our experiments and the test environment are described and the performance results are analyzed. Finally, in Section 6 some conclusions and future work are presented.

1.2 Related Work

The development of the co-schedueling system facilitated tests on a real co-scheduling system with real logs (taken from a high-energy physics experiment). Most of the previous work in this area is based on simulations.

In [3] several replication algorithms are examined and evaluated using simulations. The results in that paper show that data aware scheduling on the grid results in significant improvements in job response time. In [4] the authors describe simulation of a Grid system and evaluation of different file replication algorithms. The authors also examined various cache replacement policies. The research in [5] describes a system for treating data transfer events as real jobs. The system allows checkpointing and monitoring of data transfers. This work is complementary to our work as we can use such a system (called Stork) for scheduling data movement while using SRMs to keep track of cache contents and enforce caching policies. Another approach to perform replication management on the grid uses economic models [6] where some incentive is offered to resource owners for contributing and sharing resources, and motivates resource users

to think about tradeoffs between the processing time (e.g., deadline) and computational cost (e.g., budget), depending on their QoS requirements. In [7] the authors use an auction protocol for selecting the optimal replica of a data file. The work in [8] and [9] deals with prediction functions to make informed decisions about pre-fetching of data. Finally, we note that are various workflow systems that could be used to execute the coordinated scheduling, but modules that manage the content of the disk caches and identify the coordianated scheduling are needed as tasks of the workflow. In this paper, we show how we use Condor and SRMs to provide this functionality.

2 Main Contributions

Job scheduling systems are very complex and take a large effort to implement and support. While there are examples of such systems that are available commercially or as open source products, these systems manage scheduling of compute slots only, not the co-scheduling of data with the compute slots. Examples of such packages include SUN's Grid Engine software, Load Sharing Facility (LSF), Portable Batch System (PBS), and Condor.

Developing a system that can co-schedule compute and data resources is a very large undertaking. It took years to perfect the systems that perform only compute-slot scheduling. We address in this work the possibility of using existing software components to tackle this complex challenge. Our starting point was to select a job scheduling system and a storage management system, and design a combined co-scheduling system without making major changes to these systems. The main contribution of this paper is in the methodology and architectural design that succeeded to bring this co-scheduling system into fruition.

The key to this success was the flexibility of the existing systems we chose to work with. In particular, the Condor system is designed to perform matches between jobs and worker nodes based on an open-ended description of what to match on. Thus, we could easily extend the descriptions to include sets of files that a worker node has at any one time. This was complemented by the flexibility of Storage Resource Managers to manage their content dynamically (i.e. using automatic caching policies), and their ability to keep files for jobs that are scheduled to be matched as well as remove unneeded files after the jobs finish.

We now have a real system where various algorithms that were previously tested only by simulations can now be tested in a realistic environment. Furthermore, while we implemented this architecture on a cluster, it is straight-forward to adapt this design to wide-area Grid systems, because the components are already functional as Grid middleware.

3 Architecture

To achieve co-scheduling, the scheduler must have information on the content of each machine's disk cache, as well as the availability of compute-slots on each machine. The problem is one of matching each job to the machine that has the files needed by the job. This is achieved by providing the Condor scheduler information on the dynamic content in the disk caches in the compute nodes. This information is provided by the SRMs that reside on each computing node. We accomplish this by extending the standard mechanism used by Condor to describe a worker node to also include information on the files available on the node.

The advantages of using Condor and SRMs are numerous. Both of these systems are open-source. Condor provides job scheduling using an extensible "match-making" technology, as well as initiation and monitoring of jobs on the compute nodes. SRMs provides dynamic storage allocation, with the ability to "pin" and "release" files to ensure that they stay on the disk cache when they are needed. SRMs also have their own local policies for removing files that were released and are not needed. The combination of these two technologies is a fast and efficient way to implement and test the co-scheduling setup.

The architecture and the components we used to achieve co-scheduling are shown in Figure 1. As can be seen, the master node has three parts of the Condor system: i) *condor_schedd* (the scheduler daemon) that is responsible for scheduling jobs and keeping their state information, ii) the *condor_collector* that collects and organizes all the information about nodes in the form of *classAds* (classified advertisements); the *classAds* contain information about compute-slots in the nodes, and their hardware and software capabilities, and iii) the *condor_negotiator* whose function is to find

a match for each scheduled job; the match is done by finding a compute-slot that has at least the capabilities required by the job being matched.

The other part of the Condor system is a component, called the *condor_startd* (start daemon), whose function is to start jobs running on a node, to collect information on the node capabilities and generate the *classAd*, and to monitor the progress of the job. If the job completes successfully, the *condor_startd* advertises the availability of the slot by issuing a new *classAd*. If the job is interrupted and was not completed, it communicates with *condor_schedd* to schedule the job again. As can be seen from Figure 1, the *condor_startd* was installed on every worker node.

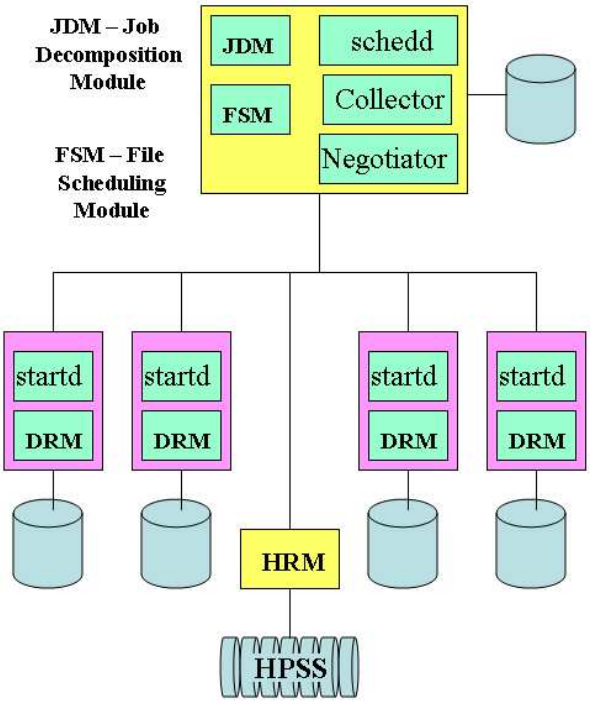


Figure 1. The architecture of the co-scheduling system using Condor and DRM components.

A disk version of an SRM, called a DRM, developed at LBNL (http://sdm.lbl.gov/srm) is also installed on every worker node. Its function is to manage the disk cache associated with the node. That includes allocating space for every file that has to be moved into the disk cache, keeping track of popular files (so called "hot files" that are accessed multiple times), and removing unwanted files ("cold" files). The DRM performs this function by "pinning" a file as soon as the file was advertised as required by a job to be scheduled to run on that node, and by releasing the file as soon as the job is finished. Note that a file may be pinned multiple times if multiple jobs are using it, and the DRM keeps track of that as well. Finally, the DRM also initiates file transfers from the mass storage system we use, by communicating with a version of an SRM, called a Hierarchical Storage Manager (HRM) that can request file staging out of the mass storage system (we use HPSS). The DRM can also request a file from a neighbor node if it is asked to do so.

We encountered one issue that could be a barrier to the scaling of the co-scheduling system. The issue was of advertising the data files that a node currently has in the Condor *classAd* for the resource available on each node. Since disk caches of worker nodes can be very large and contain thousands of files, providing this information to the component that performs matches may overwhelm the scheduler. While one can design more efficient schedulers with smart indexing technology, there was no certainty that this will scale, and it would require a serious enhancement of the existing system. Our solution to this problem is to put in the *classAds* only files that are *relevant* to the pending jobs. This is achieved by having a component on each node that gets from the Storage Resource Manager the information about which of the needed files in the job queues are currently on the node's disk cache. The ability to achieve this solution was critical to the practical success of this co-scheduling problem.

We identified two components that are needed to achieve this functionality and the co-scheduling operation: the Job Decomposition Module (JDM) and the File Scheduling Module (FSM). Both of these components are located on the master node. We explain their functionality next.

The JDM is the component which accepts jobs submitted by clients. Each job consists of an executable, a set of input files and, optionally, a set of output files. The JDM parses this information and decomposes each job into one or more jobs each requesting one or more files (referred to as a "bundle") [†]. The JDM performs the following tasks: i) decomposing all incoming jobs dynamically; ii) generating a list of files that is the union of all requested files; iii) communicating with each *condor_startd* and provide them with this list; iv) providing the FSM with a list of jobs to be

scheduled with Condor's *condor_schedd*; v) keeping track of completed jobs; and vi) providing the client with information on the progress of the job, as well as when it completes.

The FSM was designed to interact with the Condor system. It is responsible for the following actions: i) schedule with each DRM the files that it should acquire; ii) schedule all jobs with Condor's *condor_schedd*; iii) monitor the progress of jobs by inquiring from Condor which jobs are being delayed; iv) analyze the reasons for the delays and issue replication requests to DRMs; iv) decide when pre-staging of files from HPSS is warranted, and v) notify the JDM when a job completes. As can be seen, the algorithms for optimizing the co-scheduling belong in the FSM.

There were relatively small changes required to accomplish the coordination between Condor and the DRMs. The main change required from the *condor_startd* is the ability to accept a list of files from the JDM and invoke the DRM to find which of these files the DRM has. *Condor_startd* then includes these files in the *classAd* it advertises. The DRM had to be modified to provide a response to an inquiry that amounts to "which of these files do you currently have?".

Taking advantage of the fairly complex middleware systems developed over many years by making relatively modest modifications was the reason for our success in developing the co-scheduling system. This provided us with a real environment to explore the behavior of the system under different scheduling strategies. We describe in Section 5 some of the results achieved so far by running experiments on this system.

3.1 Information Flow

Figure 2 describes the information flow between the components of the co-scheduling system. For ease of explanation the steps are labeled in the logical order of flow, however, in reality these steps may repeat and run asynchronously. The steps are as follows:

[†]We note that in HEP applications it is typically possible to run an analysis job on a single file at a time because processing of collision events contained in files are independent of each other. However, in other applications it may be necessary to specify the subset of files that are needed concurently to execute the analysis job.

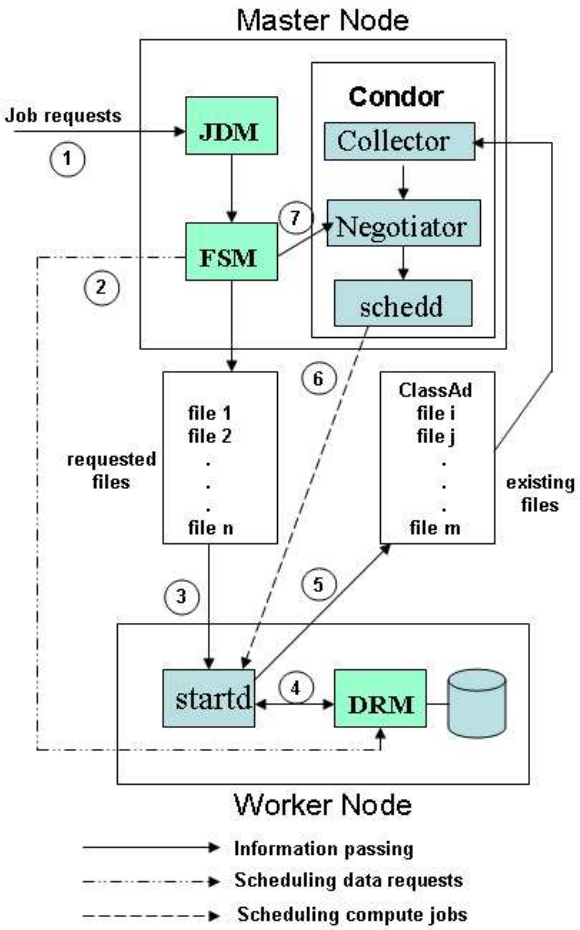


Figure 2. The steps of the co-scheduling system

1. Job requests arrive to the Job Decomposition Module (JDM), are parsed and decomposed to multiple smaller jobs, and are passed to the File Scheduling Module (FSM).

2. The FSM schedules data requests to the DRMs according to the scheduling algorithm it uses. The simplest one is round robin.

3. The FSM composes a list of all requested files that have not been processed yet. It extracts the information on job completion from Condor logs. This list of files is passed to *condor_startd*. The FSM also submits the jobs to Condor.

4. *condor_startd* communicates with its local DRM to find out which of the files in the requested-

file list it actually has. These are refered to as existing-files in the figure.

5. *condor_startd* puts the existing-files into a *classAd* that it passes to the *condor_collector*.
6. Condor finds a match for an available compute-slot on a node that has the file needed by the job and schedules that job.
7. The FSM checks with the *condor_negotiator* for jobs in queue. If there are free compute-slots, it chooses a file to replicate based on the length of time the jobs requesting it have been waiting in the queue (for details see section 4).

This iterative process is performed continuously when new jobs arrive, or when the monitoring thread triggers a replication action. The files in the DRMs stay in the cache until space is needed. The DRMs currently use a least-recently-used caching algorithm.

3.2 Dealing with bundles of files

The JDM can accept requests that may have multiple bundles of files. Recall that a bundle of files refers to a set of files that are needed concurrently to execute the computation. Our initial implementation targeted jobs where each bundle has a single file only. As noted earlier this was sufficient for typical HEP applications, because collision events are independent of each other. However, we have implemented the support for scheduling bundles of files.

Several modifications were required for this enhancement. First, the job descriptions had to be extended to represent bundle requests, and the JDM had to keep track of bundles rather than only files. Second, the job *ClassAds* to Condor had to be extended, where the condition for scheduling a bundle was expressed as AND conditions. Third, the SRMs were given multi-file requests (i.e. all the files in the bundle), a feature that was already supported by current SRMs. In this paper we describe experimental results for the case of a single file per bundle only, since it was more easily tractable in terms of performance.

4 Scheduling Algorithms

In typical particle physics analysis applications a job that requires n files can be decomposed into n smaller jobs each requiring one of the files. This increases the opportunity for parallelism since after performing decomposition, some subset of the n jobs can be scheduled to run in parallel on different servers. According to this simple model, we assume that the original jobs have been decomposed and each of the resulting jobs requires exactly one file. The scheduling problem we are considering here consists of the following inputs:

- a set of files $F = \{f_1, f_2, f_3, \ldots, f_n\}$
- a set of queued jobs $J = \{j_1, j_2, \ldots, j_m\}$ each requesting a single file from F
- a set of worker nodes $N = \{N_1, N_2, \ldots, N_k\}$ where each node N_i is associated with one or more compute-slots and a cache $C(N_i)$ that contains some subset of the files in F.

Let $f(j_i)$ be the file requested by job j_i. The job j_i can be assigned to run on a node N_k (j_i is matched to N_k in Condor terminology) if (a) N_k has an available compute slot and (b) the cache $C(N_k)$ contains the file $f(j_i)$. Among the many possible scheduling algorithms we chose to evaluate two basic algorithms. These are described below:

scheduling with no-replication. This algorithm retains at most one copy of a file on the cluster. It copies a file, f_i, from remote storage to a node N_k (selected in a round-robin fashion) only if f_i is requested by a job and cannot be found in any of the caches of the worker nodes. As long as f_i is not purged from the cache $C(N_k)$, any subsequent jobs that request the file f_i will be automatically matched with N_k by Condor once N_k has a free compute slot. In order to avoid purging f_i from the cache $C(N_k)$ prematurely, it is pinned by the system as long as there are jobs waiting for it in the queue.

scheduling with replication. Files may be replicated in the cluster across multiple nodes. A replication decision is made whenever there are jobs waiting in the queue and there are available compute-slots in the system. The selection of which file to replicate next is determined by a weight function $w(f_i)$ computed as follows: For each file f_i requested by one or more jobs in the queue, $w(f_i)$ is

equal to the total time these jobs have been waiting for it. The file with the maximal $w(f_i)$ is then replicated on one the worker nodes with a free compute slot chosen at random. The rationale behind this replication algorithm is that "popular" files should be available on multiple nodes for better load-balancing of the system and jobs that have been waiting for a long time should get some priority to avoid starvation.

The first algorithm (scheduling with no-replication) attempts to minimize file transfers from the mass storage system across the network and also saves on disk storage requirements. It may be attractive in situations where file transfers and disk storage costs are relatively expensive resources. The second algorithm (scheduling with replication), tends to move more files but reduces queue waiting times and improves system utilization. We chose to experiment first with these two basic algorithms since they are simple to implement and do not introduce excessive overhead costs on the system. We are planning to evaluate more elaborate algorithms that take into account node capacities, different replication costs (remote vs. local) using mathematical optimization techniques.

5 Experimental Results

5.1 Description of physics analysis environment and data characteristics

We experimented with data from BaBar, which is a High Energy Physics experiment with over 600 world-wide collaborators (http://www.slac.stanford.edu/BFROOT). The data for this experiment is stored on tapes at the Stanford Linear Accelerator Collider (SLAC) managed by IBM's Mass Storage System HPSS storing over 1.3 petabytes of data on about 13,000 tapes managed by 6 StorageTek tape silos. To deliver data to jobs in a reasonable time the system is currently backed by 160TB of disk cache implemented on thousands of physical disks bound into large arrays managed by Sun's Solaris 9 UFS. The analysis jobs run on a cluster of hundreds of nodes accessing the persistent data through a high performance data server [10]. The work logs used in our paper were extracted from trace data produced by the data server. The raw

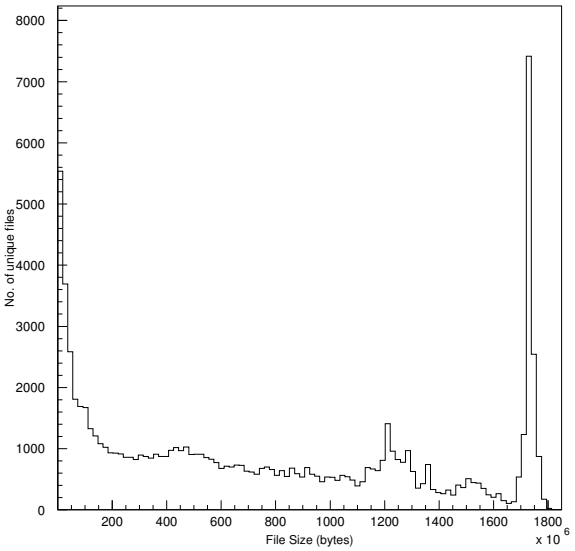


Figure 3. Unique file size distribution

trace data contained, for each job, only its job id, the files accessed by the job, and the time of access of each file. We had to match this information with the file characteristic data stored in a Oracle database at SLAC in order to get file size information. We analyzed trace logs taken from October 1 to October 26, 2004. During this time interval 504,493 jobs were submitted requesting a total of 2,028,541 files, 86,378 of which were unique. Figure 3 shows the size distribution of the 86,378 unique files. Note that the maximum size of a file is close to 2GB, due to file system limitations. There is also a significant number of files of size less than 200MB. The large files most likely represent raw detector data, whereas the smaller sized files are in most cases filtered data ("skims") for the purpose of user analysis.

In Figure 4 we plot the number of file requests as a function of file size. We note that the large files appear in more requests as compared to the small files. One possible explanation for this access pattern is the automated running of "skimming" which use the raw data files as input to produce the user-analysis files.

5.2 Description of the cluster environment

We ran our experiment on a cluster of 8 single and 1 dual CPU 1.5GHz AMD Athlon processors, each with

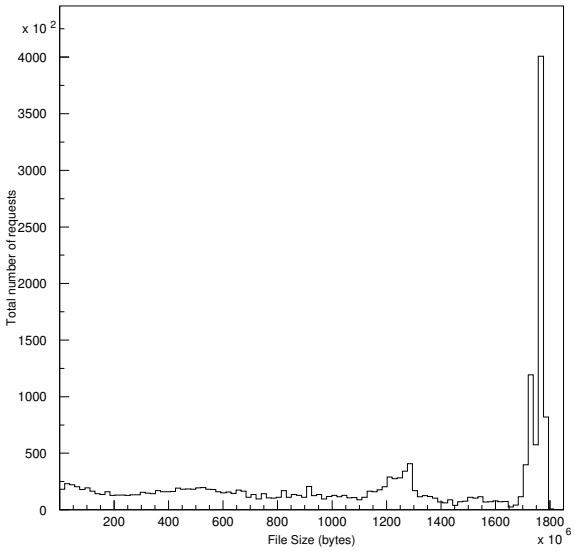


Figure 4. Requested file size distribution

a 20GB disk cache and 2GB of RAM for a total of 10 compute slots. The cluster nodes are on a Giga-Bit network. We installed one DRM on each machine. Condor software and the FSM were installed on the dual processor. The data files needed for the analysis jobs were stored on the HPSS mass storage system at LBNL. We observed that the average transfer rate from the HPSS system was 15MB/s.

5.3 Experimental setup and performance results

Each experiment consisted of a sample of 1500 jobs from the pool of jobs presented above. Assuming the duration of a typical analysis job is proportional to the size of the input, we simulated a job by running 100 empty for-loops per each byte requested. We experimented with different job arrival rates, and for the purpose of this study we chose an interval of 60 seconds between job submissions based on the average job execution time to match the job arrival rate to the job service rate. Shorter arrival intervals would lead to a saturation of computing resources, while a very long arrival interval would underutilize the cluster. Based on these parameters, each experiment took about 27 hours of continuous running time. Furthermore, we had to set the Condor configuration parameters to much smaller values than the default values. For example, the default resource matching negotiation cycle, whose default value is 300 seconds was lowered to 60 seconds to make the matchmaking more responsive to our job arrival and data transfer rates.

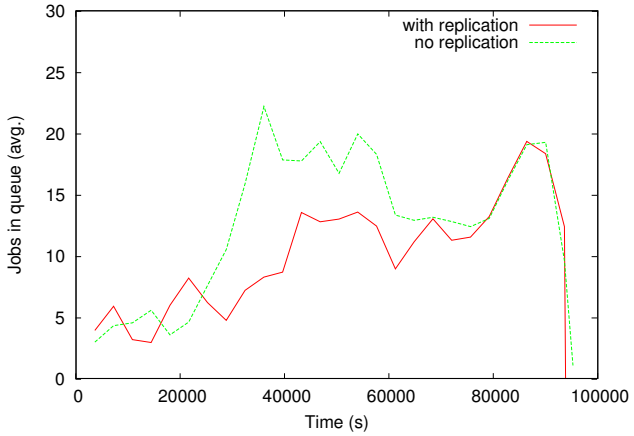


Figure 5. Average number of queued jobs

In Figure 5 we compare the two algorithms in terms of the number of jobs waiting to be matched in the Condor queue during the running time of the experiment. As expected, the number of jobs in the queue under the *with-replication* algorithm is almost always smaller than that of the *no-replication* algorithm. This is due to the fact that the *with-replication* algorithm removes jobs from the queue as soon as compute slots become available.

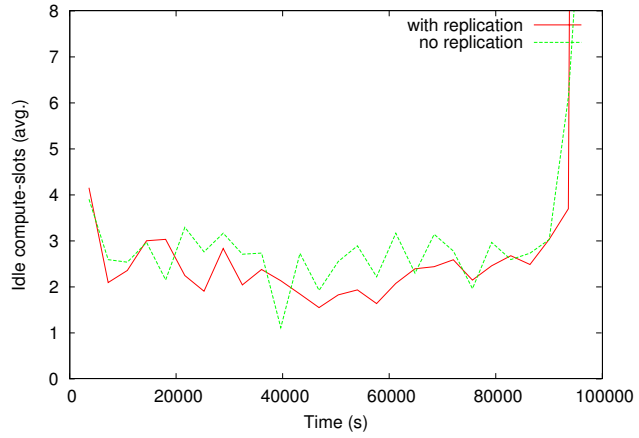


Figure 6. Average number of idle compute-slots

In Figure 6 we compare the system utilization under the two algorithms by counting the number of idle compute-slots during the running time of the experiment. Existence of idle compute-slots while there are

jobs waiting in the Condor queue represent wasted system resources. Again the *with-replication* algorithm achieves better system utilization and shows almost always fewer idle compute-slots as compared with the *no-replication* algorithm.

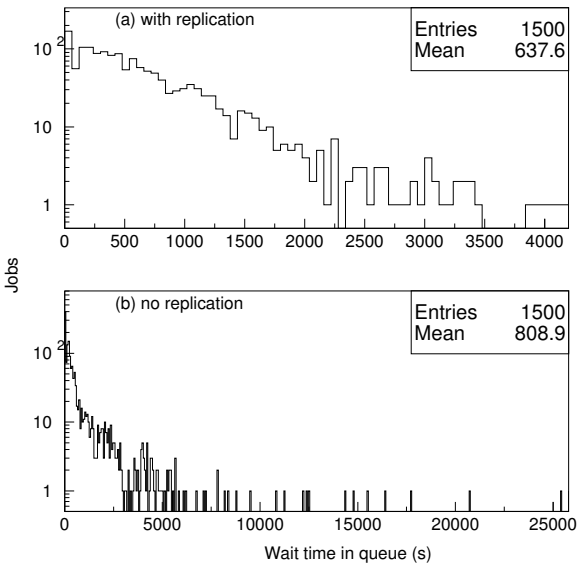


Figure 7. Waiting time in queue

Next we looked at the average time that jobs waited in the Condor queue under both algorithms. Waiting time in the queue is calculated as the number of seconds between the time a job arrives at the system until the time it is matched by Condor with some compute-slot and submitted for processing. The first important observation here is that the *with-replication* algorithm has a maximum waiting time of 4000 seconds whereas the no-replication has a maximum waiting time of 25000 seconds. This represents a dramatic improvement (a factor of 6) in terms of worst case behaviour. The mean waiting time of the *with-replication* algorithm is also better (by about 25%).

6 Conclusions and future plans

The main accomplishment described in this paper is the ability to put together a co-scheduling system from existing Grid middleware components that were designed to manage compute and storage separately, by making relatively small enhancement to these systems. The problem of optimizing the behavior of these systems is the task of an external File Scheduling Module that makes file replication choices based on monitoring information of queues from the Condor scheduler. File replication is facilitated by making requests to the DRMs on each node. Garbage collection from the nodes disk caches are performed by the DRM based on usage policies. Thus, the tasks of scheduling, executing, monitoring, file movement, and garbage collection are performed by the existing Condor and DRM systems.

Future work includes trying out various optimization algorithms that are typically based on approximation results from scheduling theory [11] as computing optimal solutions for the co-scheduling problem is known to be NP-complete [12]. While we performed simulations to compare several algorithms, it is important to verify the performance on real systems. Furthermore, the next step will be to implement and test this combined co-scheduling system on Grid testbeds. Since the components we used on the cluster are general Grid components we believe that applying them in a real Grid system will be straightforward. The real challenge will be to measure and understand performance in an uncontrolled environment such as the Grid.

Acknowledgement

We thank Ekow Otoo from LBNL for his early participation in this project, and John Bent from the University of Wisconsin for pointing out relevant work. This work was supported by the Director, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under contract No. DE-AC03-76SF00098.

References

[1] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor – a distributed job scheduler," in *Beowulf Cluster Computing with Linux*, T. Sterling, Ed. MIT Press, October 2001.

[2] A. Shoshani, A. Sim, and J. Gu, "Storage resource managers: Essential components for the grid," in *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers, 2003.

[3] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *Proc. 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 2002)*, Edinburgh, Scotland, 23-26 July, 2002, pp. 352–358.

[4] W. Bell, D. Cameron, L. Capozza, A. Millar, K. Stockinger, and F. Zini, "Simulation of dynamic grid replication strategies in optorsim," in *Proc. Grid Computing - GRID 2002, Third International Workshop*, Baltimore, MD, USA, November 18 2002, pp. 46–57.

[5] T. Kosar and M. Livny, "Scheduling data placement activities in grid," University of Wisconsin - Madison Computer Sciences Department, Tech. Rep. UW-CS-TR-1483, July 2003.

[6] R. Buyya, H. Stockinger, J. Giddy, and D. Abramson, "Economic models for management of resources in peer-to-peer and grid computing," in *Proceedings of the SPIE International Conference on Commercial Applications for High-Performance Computing*, Denver, USA, August 20-24 2001.

[7] W. H. Bell, D. G. Cameron, A. P. M. Ruben Carvajal-Schiaffino, K. Stockinger, and F. Zini, "Evaluation of an economy-based file replication strategy for a data grid," in *International Workshop on Agent based Cluster and Grid Computing at CCGrid 2003, Tokyo, Japan*. IEEE Computer Society, 2003.

[8] L. Capozza, K. Stockinger, and F. Zini, "Preliminary evaluation of revenue prediction functions for economically-effective file replication," CERN Geneva, Switzerland, Tech. Rep. DataGrid-02-TED-020724, July 2002.

[9] J. B. Weissman, "Predicting the cost and benefit of adapting data parallel applications in clusters," *J. Parallel Distrib. Comput.*, vol. 62, no. 8, pp. 1248–1271, 2002.

[10] J. Becla and D. L. Wang, "Lessons learned from managing a petabyte." in *CIDR*, 2005, pp. 70–83.

[11] M.Pinedo., *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 2001.

[12] J. Bent, D. Rotem, A. Romosan, and A. Shoshani, "Coordination of data movement with computation scheduling on a cluster," in *Proceedings of the Workshop on Challenges of Large Applications in Distributed Environments (CLADE)*. To be published, July 2005.

Scientific Models Management in Computational Grids

Halisson Matos de Brito[1], Julia Strauch[2], Jano Moreira de Souza[1, 4], Carla Osthoff[3]

[1] *COPPE/UFRJ – Systems Engineering and Computer Science Program*
Federal University of Rio de Janeiro – PO Box 68511, ZIP Code: 21945-970, Rio de Janeiro, RJ, Brazil.
{hmbrito, jano}@cos.ufrj.br
[2] *ENCE /IBGE – National School of Statistical Sciences, R. André Cavalcanti, 106 s. 401*
ZIP Code: 20231-050, Rio de Janeiro, RJ, Brazil.
juliast@ibge.gov.br
[3] *LNCC – National Laboratory for Scientific Computing*
Av. Getúlio Vargas, 333, Quitandinha, ZIP Code: 25651-075, Petrópolis, RJ, Brazil.
osthoff@lncc.br
[4] *IM/UFRJ – Institute of Mathematics/Federal University of Rio de Janeiro*
PO Box 68511, ZIP Code: 21945-970, Rio de Janeiro, RJ, Brazil.

Abstract

This paper presents MODENA, an architecture for scientific models management using Computational Grid platform. This architecture is comprised of two systems: ModManager and ModRunner. ModManager deals with knowledge management about scientific models, acting as a scientific models library allowing for cataloguing, searching, reutilization and generation of new models. To achieve this, a metamodel is proposed to classify models, in order to support the organization, searching and retrieving of models. ModRunner manages the execution of models in a Grid environment allowing for model composition to generate a scientific Grid Workflow to be executed by distributed services offered by Grid Services. An initial prototype of ModManager is presented.

1. Introduction

Models are simplified representations of reality, whose goal is to abstract the reality portion which maters to the solution of a problem. Besides, models contain relevant information on phenomena or processes with the advantage of hiding irrelevant details of real problems.

In scientific work, phenomena or processes are usually more complex and sometimes unknown, highlighting the importance of using models to map them. So, models are an essential part of any scientific experiment. An experiment usually tries to prove some hypothesis stated by the scientist and it may have an underlying model, or even a combination of models about the phenomenon it is intended to prove. Thus, models play an important role both in the research area and in the practical applications in many knowledge areas.

To deal with this great variety of models, or even to keep a history of them, a researcher, a team of researchers or even a research organization can take advantage of a system which manages existing models, using database management techniques to perform cataloguing and retrieving of models in a "scientific models library".

Researchers could have their models stored in a library and managed by an application, increasing their means to share models and co-operate each other. An advance in this state-of-the-art could be the reuse of existing models stored in different researcher bases, to compose a new model and execute it as a distributed models workflow.

This is possible due to the growing popularity of the Internet, associated with the increasing availability of powerful computers and high-speed networks, which brings the concept of Computational Grids.

So this work presents MODENA, an architecture for scientific model management using Computational Grid platform. This architecture allows for cataloguing and retrieving models from a models library, as well as the generation of model composition under which workflows can be created to be executed on Computational Grids. This architecture is under development on the DesenSus Project (a Platform for Generation and Execution of Integrated Models in Sustainable Development Research), to support researchers of Geoma Project (Thematic Network for Research in Environmental Modeling of Amazon) [1], which aims at the development of models to evaluate and foresee sustainability scenarios under different kinds of human activities and public politics for the Amazon.

2. Metamodel for model classification

According to Christofoletti [2] the most common modeling goals are concept communication and short-term forecast, allowing for answering, predicting or comparing alternatives as planning instrument. However, it is worth mentioning that, although models are a subjective problem-solving approach, as they do not include all reality details,

this feature is valuable for enabling the arising of fundamental aspects of reality.

In literature, the concept of model is broad, varying from the simplest definitions – as *it is any simplified representation of reality* – to the most elaborate and specific as in [2], [3], [4], [5], [6] and [7]. As models are used in the most diverse knowledge areas, there is not a pattern to classify and describe them. Model taxonomy varies from one research area to another, or even in the same area, as it can be observed in the classifications proposed by [2], [4], [5], [6] and [7].

Nevertheless, the adoption of a model classification is useful, especially when it aims at selecting or storing models. Model classification allows for defining more efficient search and selection mechanisms as well as systematization of data identification, organization and retrieval.

Hence, this work proposes a metamodel for model classification which allows for organizing, searching and retrieving models. This metamodel is a hierarchical classification without any restriction on the number of levels of the tree in order to make researcher or research teams feel free to use an existing classification form or even to define their own classification forms.

Figure 1 presents the Entity-Relationship (ER) Model of the database which stores model metadata. The ER model is quite simple, although expressive enough to support scientific model management.

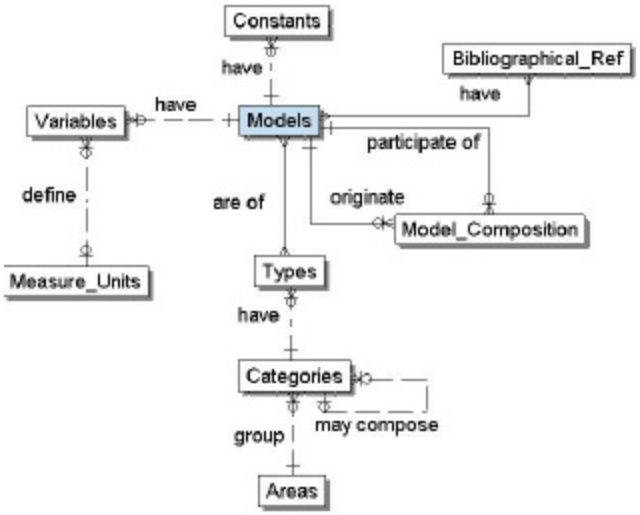


Figure 1. Entity-Relationship Model of the system

The hierarchical classification depicted above is represented by the entities 'Types', 'Categories' and 'Areas'. This means that models are of 'Types', grouped in 'Categories', which belong to knowledge 'Areas'. As may be seen in Figure 1, the relationship cardinality between models and types is (NxN), indicating that models can be classified as having more than one type.

Still regarding the classification one should notice in the ER model that the entity 'Categories' bears a self-relationship, indicating that it is possible to have any number of levels of subcategories to any category.

Finally, a self-relationship in the entity 'Models' originates the entity 'Model_Composition'. Thus any model may be composed by several models. Also, one model can be within several compositions.

3. MODENA: Architecture for Scientific Models Management in Computational Grids

MODENA (MOdel DEvelopmeNt Architecture) presents three levels (Figure 2). In the first level, lies the interface with all functionalities for knowledge management on the models and the execution management of model instances. The second level contains the processing layer in computational grids. In the third level are the distributed databases, metadata base, models base, ontology base and knowledge base.

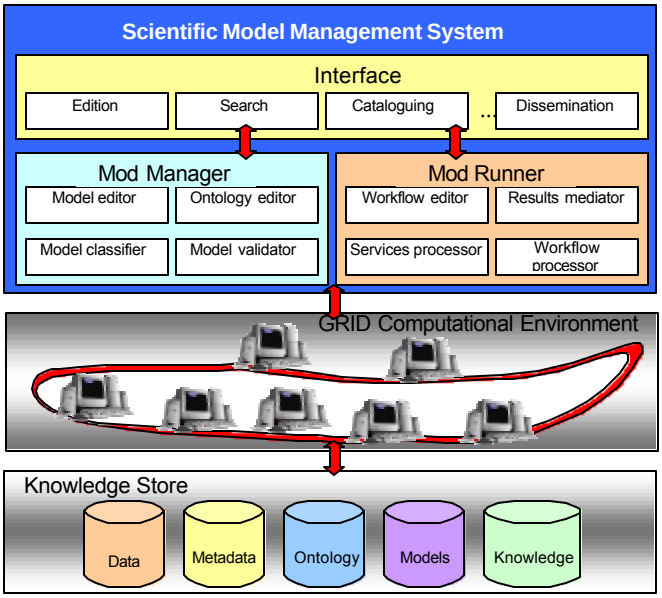


Figure 2. MODENA Architecture

MODENA consists of two modules. The first one, called ModManager, comprises a system for Knowledge Management on Scientific Models, responsible for capture, recovery, generation and knowledge exchange stages.

The second, called ModRunner, is a tool to aid the performing of scientific experiments through the execution of instances of the models stored in the base. These instances are constituted of workflows which represent the steps for model execution. So, ModRunner comprises a Workflow Management System on Scientific Models, in grid environment.

For workflow description, we have analyzed three languages described in literature: GSFL (Grid Services Flow Language) [8], GFDL (Grid Flow Description Language) [9] and GWEL (Grid Workflow Execution Language) [10].

The chosen language for use in MODENA was GSFL, due to its features as well as for the fact that the group responsible for it is developing more complex structures for workflow definition.

3.2. Mod Manager: a system for Scientific Model Knowledge Management

The requirements for the development of the knowledge management system on models took the needs bellow into consideration:

- Capturing scientific knowledge existing in an organization or research group, through the model cataloguing system;
- Retrieving knowledge through search in the model base;
- Generating new knowledge by model composition, in other words, the generation of new models starting from existing ones; and
- Allowing knowledge interchange among researchers' teams, whether by exchange among model bases (importing and exporting of models), or by making models available as Knowledge Objects (KO) [11].

One of the premises which give support to the model management, according to Dolk [12], is that models, as well as data, are an important organizational resource and should be managed as strictly and with so attention as the data. Therefore, for Dolk [12], a MMS (Model Management System) should provide similar functionalities to a Database Management System (DBMS): Model Description; Model Manipulation; and Model Control.

In this way, knowledge management on models was implemented with a database on which a management system acts to accomplish the cataloguing and search for models. This meets the needs mentioned above, in which the description of the models lies in the database and model manipulation and control is exercised by the management system.

3.3. ModRunner: Workflow Management on Scientific Models using Grid Services

Workflow management on Scientific Models aims at managing the following stages of model instance execution, presented in Figure 3:

1. Selection of the model to be submitted to execution;
2. Generation of a workflow representing the steps for the execution of the model;
3. Discovery of grid resources available to the researchers;
4. Selection of resources to be used for workflow execution;
5. Submission of the workflow to execution in grid environment; and
6. Capture and storage of the results.

The selection of the model to be executed can be accomplished directly by ModRunner through a search on the models base or via ModManager once a model is already selected. Alternatively, model composition can be accomplished aiming at carrying out experiments with new models in order to identify the most appropriate one for problem solution. Besides, this model composition instance can later be added to the base, as a new model.

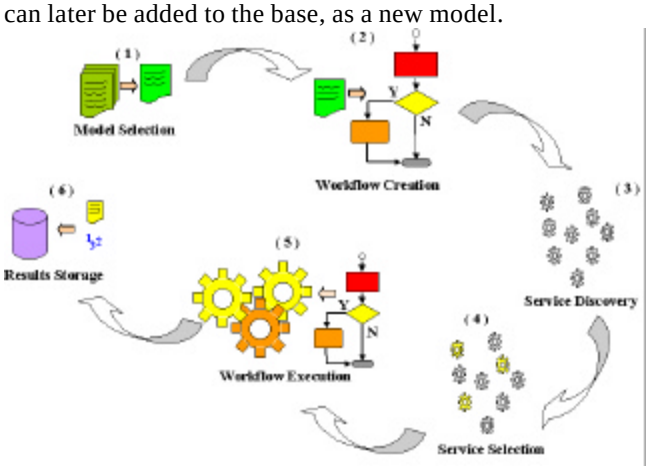


Figure 3. Workflow execution in computational grid

After model selection, a workflow will be generated containing the steps for the execution of the model. That generation will be accomplished through the description of the execution steps in the GSFL language so that a workflow in grid is described.

Workflow generation is one of the most complex stages of the system, as the mapping from model to workflow is not a trivial task, on account of the enormous variety of model types and representations. This stage also involves a certain degree of automation and the user's participation, among others, in the definition of the data which will serve as a basis for workflow execution.

Soon afterwards, discovery of available resource in Grid for workflow execution will be actually accomplished through the discovery of the Grid Services accessible by the research group. This search will be made following the patterns defined by GGF (Global Grid Forum).

Once the discovery of the resources have been made, these can be selected to participate in the execution of the workflow, based on certain criteria, such as: proximity, readiness, storage capacity, computational power, network connection capacity, among others. Selection can be made manually, by the system user himself, or automatically, in which the system decides based on user's previous configurations.

After resource selection, the workflow would be submitted to execution through the submission of the corresponding GSFL document to the workflow execution machine based on this language. Krishnan (2002) [8] presents the prototype of a machine for this end and he affirms that the same is in constant development, inside the GGF proposals.

4. MODENA Prototype

MODENA architecture is being developed to validate our proposals besides acting as a model database for the systems to be created in the DesenSus Project scope.

The portion of the architecture already developed corresponds to the ModManager system, which admits cataloguing and searching for models, acting as a scientific models electronic library. This is being developed using Sun Java technology, and also allows importing and exporting of models in formats like CSV and XML, as well as generation of KOs, enabling model exchange among researchers in order to support scientific cooperation. Moreover, the Model Composition feature aiming the generation of new models is under development.

The system can be accessed via any web browser and presents, on the left of its initial screen, a menu with the available options. Figure 4 shows the model cataloguing screen, after the options 'Catalog', and then 'Model' are chosen. Besides models, it is possible to catalogue knowledge areas, categories and types, in addition to bibliographical references, related to the models.

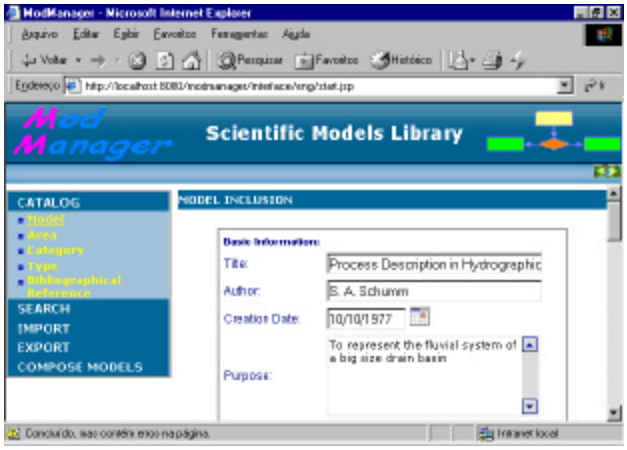


Figure 4. Model cataloguing screen

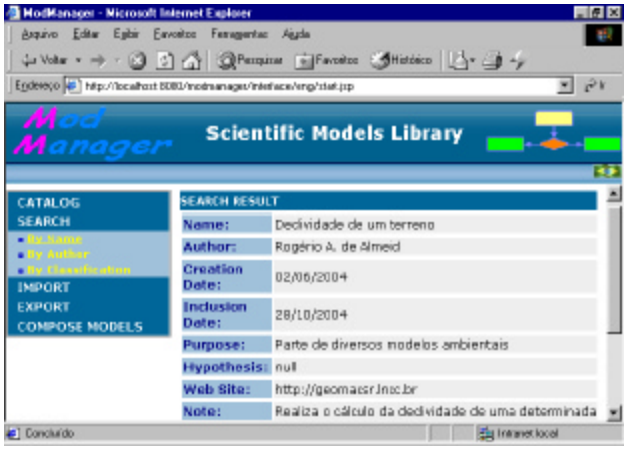


Figure 5. Search by classification result

On the search section, the system returns models which satisfy some search criteria. Three types of search are possible: by 'Name' (caption of model), by 'Author' (creator of model) and by 'Classification' (type, category and area of model).

The result of a search by classification is shown in Figure 5, in which details of the model can be observed.

5. Final considerations

The MODENA value lies mainly in the fact of allowing researcher groups to store and manage their models efficiently, avoiding problems such as the loss of models not classified and time waste in search for a model needed for execution of a scientific experiment.

Besides, MODENA makes cooperation possible among scientific teams through model sharing or even the composition of distributed models. Another contribution of this paper is the metamodel proposal to classify models providing an efficient way of organizing the models, by respecting particularities of specific areas.

Acknowledgement: This work was partially funded by CNPq (Brazilian National Council for Research) Agency.

References

[1] Geoma Project. http://www.geoma.lncc.br/. (2004)

[2] Antonio Christofoletti. Environmental Systems Modeling. Ed. Edgard Blücher. São Paulo. (1999).

[3] P. Haggett, R. J. Chorley. Models, Paradigms and the New Geography. In Models in Geography (R. J. Chorley, P. Haggett, Eds.). London, Methuen & Co. (1967).

[4] J. K. Berry. What's in a model. GIS World. 8(1):26-28. (1995).

[5] Anita S. Coleman. Scientific Models as Works. ASIST 03. Long Beach, California. (2003).

[6] M. J. Kirkby. Computer Simulation in Physical Geography. New York, John Willey & Sons. (1987).

[7] S. Banerjee, A. Basu. Model Type Selection in an Integrated DSS Environment. Decision Support System. Vol. 9. (1993).

[8] S. Krishnan, P. Wagstrom, G. von Laszewski. GSFL: A Workflow Framework for Grid Services. http://www.globus.org/cog/papers/gsfl-paper.pdf. (2002).

[9] Z. Guan, et. al. Grid-Flow: A Grid-Enabled Scientific Workflow System with a Petri Net-Based Interface. (2004).

[10] Dieter Cybok. A Grid Workflow Infrastructure. Proceedings of Workflow in Grid Systems Workshop in GGF10. Berlin, Germany. (2004).

[11] Jonice Oliveira, Jano M. Souza. Improving Knowledge Sharing through Knowledge Objects Representation. 4th International Conference on Knowledge Management, Austria. (2004).

[12] Daniel R. Dolk. A Generalized Model Management System for Mathematical Programming. ACM Transactions on Mathematical Software, V. 12, No. 2. p. 92-126. (1986).

Lineage Path Integration for Phylogenetic Resources

Katherine G. Herbert* Shashikanth Puspati† Jason T. L. Wang‡ William H. Piel§

Abstract

With the increase of genome and proteome data, phylogenetic information and phylogenetic analysis tools are increasing greatly in current biological repositories. First, many repositories allow users to browse information about species through taxonomic tools. These tools present the species with its lineage path and links to the various types of information the repository provides about the species. Second, some multiple sequence alignment tools offer users basic phylogenetic data through applying basic reconstruction algorithms to the alignment. With the availability of this information in multiple locations, integrated tools are needed to allow the user to compare this data. This paper presents data integration research on lineage paths using the BIO-AJAX framework. It introduces BIO-AJAX for Lineage Paths, a tool that integrates lineage path information for NCBI Taxonomy Database [1] and the Integrated Taxonomic Information System (ITIS)[6].

1. Introduction

Biological data, specifically phylogenetic data, is rich with issues that can be addressed with data quality and integration methodologies. Data quality in biological data is an important function necessary for the analysis of biological data. It can standardize the data for further computation and improve the quality of the data for searching. Data integration is also an important function necessary for analyzing biological data [3, 5, 7, 8, 9]. The very core purpose for most biological databases is to create a repository, integrating work from numerous scientists. Phylogenetic data encapsulates these problems. It is a complex data set that consists of processed wet lab results, data obtained through knowledge discovery, structural data and metadata. Moreover, phylogenetic studies also have multiple applications from Tree of Life studies to biomedical investigations [14].

Lineage path studies represent a unique opportunity in phylogenetic data integration. Currently, many repositories model lineage paths in some manner. These lineage paths can represent various types of knowledge including the taxonomy of a species, the development of the Tree of Life with respect to a databases data model or the intermediary nodes developed from a phylogenetic construction. With these different purposes for lineage paths, finding similarity between repositories concerning lineage paths can be confusing and sometimes inconsistent from the user's point of view.

This short paper introduces research issues and concerns regarding the lineage path integration and data quality problems in phylogenetic studies. Lineage paths themselves tend to relate to taxonomy studies. However, since this method can be employed on both taxonomic databases and phylogenetic tree databases, we will discuss this tool in reference to its phylogenetic purposes. It first identifies the lineage path problem, discussing why it is of interest to researchers. Next, it discusses applying data quality and data integration techniques to lineage paths to create an environment that facilitates user comparisons of lineage paths. Finally, it identifies future work for this project as well as makes some concluding remarks.

2. Lineage Paths

A lineage path is the path from a given point on a phylogenetic tree(such as the Tree of Life) to a specific taxon. Sometimes this taxon can be a species, which tends to be a terminal node on the Tree of Life or it can be an intermediary node within the tree. Most lineage paths concern the path from the root node of the Tree of Life to a specific taxon. Lineage paths pose many different problems for phylogenetic researchers. For example, in phylogenetic nomenclature, the lineage or ranking of an evolutionary unit or taxon is not standard. NCBI Taxonomy Database [1] uses 28 distinct ranks for classification while the International Code of Botanical Nomenclature uses 25 rankings [13]. These databases maintain, besides different categories

*To whom correspondence should be addressed: Department of Computer Science, Montclair State University, Montclair, NJ 07043, USA, herbertk@mail.montclair.edu

†Department of Computer Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102, USA.

‡Department of Computer Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102, USA.

§Department of Biological Sciences, 608 Cooke, University at Buffalo, Buffalo, NY 14260, USA.

for their taxonomies, depreciated lineage paths. Therefore, semantic integration must be used so that the matching of the rankings and the treatment of the depreciated paths are correct. When adding the complexity of a research phylogenetic tree repository such as TreeBASE, where there are multiple trees for one species and non-standard nomenclature, integration becomes more complex.

Lineage paths and lineage path comparison offers phylogenetic researches many interesting functionalities for their research. Through their comparison, researchers can analyze differences in phylogenetic reconstruction methods, understand differences in rankings among phylogenetic databases as well as perform some advanced queries upon the phylogenetic studies. Some possible queries of interest associated with lineage paths include: Compare the lineage paths for taxon X from database D1 and database D2; Given taxon X, find all lineage paths containing taxon X and; Given taxon X, find all taxa which are descendents (or ancestors) of X.

Foremost, lineage path querying offers the ability to compare ranking systems among the supported databases. For example, as mentioned previously, there are differences in ranking systems from database to database. By offering lineage path querying, a user can compare side by side the differences between the paths among the different databases. These rankings are important since the scientific name of the species is dependent upon where it falls within a given ranking system. If a species is classified differently from database to database, it affects the type of data that can be retrieved about a particular taxon. A user may have an understanding of one ranking system without understanding another. By allowing for comparison, we permit the user to evaluate each databases ranking system, seeing where the data he is interested in may be located [11, 13].

Extending these path comparisons to phylogenetic studies, it can also help scrutinize the differences between two trees which have similar taxa but use different reconstruction methods. By breaking the trees down to their paths, we can monitor the differences in the classification of a specific taxon through various reconstruction methods. This can be useful in analyzing the effectiveness of a reconstruction method as well as determining a species proper ranking.

Finally, lineage path queries can also help with advanced querying. One extremely powerful query that no phylogenetic database supports effectively is given a node N of the phylogenetic tree, find all ancestors or descendents from N. To perform this type of search, most databases require that the user navigate through some hierarchical browsing method of analyzing the tree. To find a specific path of the tree, the user must be familiar with the tree. For example, if a novice user tries to find "Homo sapiens" from the root of the tree, most novices would not understand enough about phylogenetics to know that the first classification "Homo sapiens" falls under is "Eukaryota" . Therefore, a user can enter a portion of the path, for example "Homo sapiens", and get the path beginning (or ending) at "Homo sapiens". Also, if the user is unclear about what type of organisms would fall under "Eukaryota", he can enter "Eukaryota" as a partial path query, receiving back all paths that contain "Eukaryota".

3. The BIO-AJAX Toolkit for Lineage Paths

The BIO-AJAX [4] toolkit facilitates improving the state of lineage path querying through integrating lineage path resources. By applying the data cleaning architecture employed by BIO-AJAX [2, 4], various lineage path resources can be integrated together to offer the user the ability to query and compare these lineage paths.

The BIO-AJAX framework for Lineage Paths has been implemented using the NCBI Taxonomy Database and the Integrated Taxonomic Information System (ITIS) [6]. Both tools allow querying upon their taxon set and return lineage paths as a part of the answer to the query. In the current implementation of BIO-AJAX for Lineage Paths, the following queries are addressed: Compare the lineage paths for taxon X from database D1 and database D2 and; Given taxon X, find all taxa which are descendents (or ancestors) of X.

In future versions of this tool, the option of finding any lineage path that contains taxon X will be provided. Moreover, other database repositories' lineage paths, such as TreeBASE's will also be incorporated into the integrated framework.

4. Implementation

The current implementation for BIO-AJAX for Lineage Paths uses the data warehousing technique for integrating the data and is accessible through a World Wide Web interface [5]. Figure 2 displays the interface for this tool. The lineage paths are extracted from both NCBI Taxonomy Database and ITIS and stored locally. In previous versions of the tool and previous instantiations of BIO-AJAX, the mediator method was used to display the paths. However, due to limitations in accessing each repository, this method had to be replaced with the warehousing method. Moreover, since each lineage path needed to be manipulated to get very specific data out of it, the mediator method became impractical. Also, for finding all ancestors and descendents, the lineage path strings needed to be parsed creating a long lag time for the Web interface. Therefore, the platform was shifted from using the mediator method to the data warehousing method. Figure 1 demonstrates the configuration of the system.

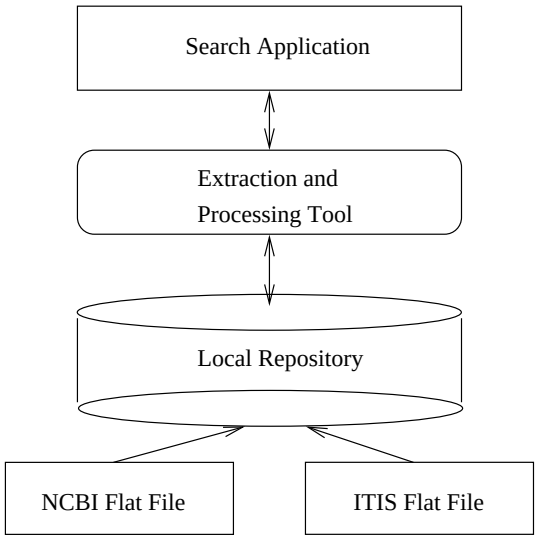


Figure 1. The software architecture of BIO-AJAX for Lineage Paths.

4.1. Lineage Path Extraction

For both data repositories, lineage paths can be extracted from the flat files each repository provides for download. While both databases vary in format, both store the lineage path information in similar ways. Both databases have signified one taxon as a "root" for the tree of life its data model represents. Each taxon in both repositories has various types of information stored about it, including the taxon id and the number of its immediate ancestor. Therefore, to obtain any taxons lineage path, a user would traverse the flat file recursively, until he obtains the root taxon.

Initially, the flat files from the repositories are downloaded to local storage in a MySQL database. Next, the scientific nomenclature is associated with its taxon identification number. To facilitate traversal of the paths, all taxa are also indexed through their taxon identification numbers. For both the ancestor and descendent tables, the paths are extracted from these flat files and stored locally. Moreover, the ranking files from each database are also extracted to provide the user with information about the terms in the paths so that informed comparison is possible.

4.2. Lineage Path Retrieval

Querying for the purpose of comparison, for finding all ancestors and for finding all descendents poses different problems that must be addressed so that the query is executed properly. Concerning querying for comparison and querying to find all ancestors, the data returned is very similar for both queries. However, for finding all descendents, the path must be manipulated in a different way to obtain these answers both efficiently and effectively.

To execute these three queries, a user can interact with the indices through a web based search mechanism. On this web page a user enters a taxon name within a text box. He next selects from three choices (Ancestor, Descendent and Home) listed beneath the text box what type of query he wants executed. From this, the appropriate query is executed. For each query, results from both NCBI and ITIS are displayed along with the ranking for each member taxon of the lineage path. Figure 2 displays the output for an ancestor query.

4.3. Lineage Paths and Data Cleaning

In the current method for finding the descendents, a number of taxa can possibly be returned numerous times, depending upon how close the query taxon is to the root of the tree when employing the descendent query. Therefore, data cleaning methods can be applied to these paths to eliminate redundancy [10, 15].

One possible method for eliminating redundancy is to apply the sorted neighborhood method to the array to detect similarities. Since the paths are highly structured and common errors such as spelling errors are rare, this becomes an ideal application for using the sorted neighborhood method. In a simple implementation of the sorted neighborhood method on lineage paths, the very least common paths between taxa on the same level of the tree can be found by comparing the lineage paths up to but not including the final taxon in the path. Identical paths can be merged so that redundancy is eliminated. For more advanced applications, there can be a number of iterations of this comparison, where the first iteration starts by comparing for one common taxon and builds to more complex paths.

5. Conclusion and Future Work

Lineage paths offer an interesting and rich method for phylogenetic researchers to explore various interpretations of the Tree of Life. By creating a comparative environment for phylogenetic researchers, problems concerning the correctness of the Tree of Life can be addressed.

Future work concerning this research includes integrating more repositories into the tool as well as improving the user interface [12, 16]. Moreover, this work can be further applied to consensus tree and supertree generation problems. Also, work can be done to further the visualization aspects of this tool concerning the comparisons.

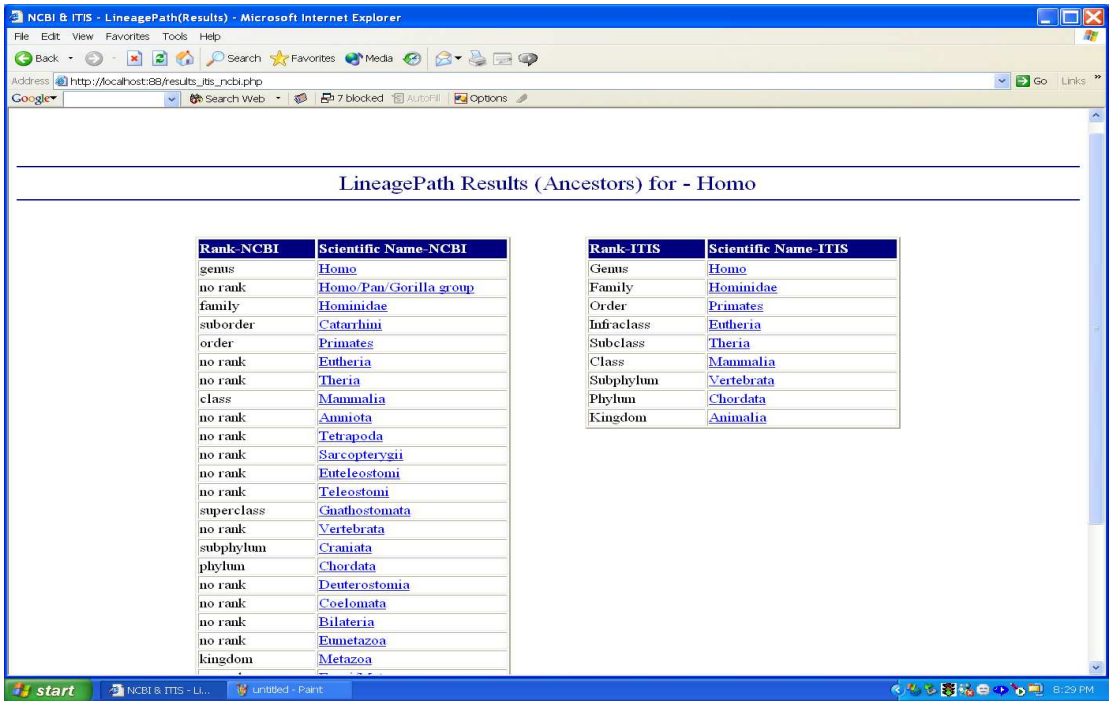


Figure 2. The BIO-AJAX interface for showing an example output for an ancestor query.

References

[1] S. Federhen, I. Harrison, C. Hotton, D. Leipe, V. Soussov, R. Sternberg, and S. Turner. NCBI Taxonomy Homepage, `http://www.ncbi.nlm.nih.gov/ Taxonomy/taxonomyhome.html/`, 15 January 2005.

[2] H. Galahardas, D. Florescu, D. Shasha, E. Simon and C.A. Saita. Declarative Data Cleaning: Language, Model and Algorithms, in *Proc. of 27th International Conference on Very Large Data Bases*, September 11-14, 2001, Roma, Italy : 371-380.

[3] A.Y. Halevy. Answering queries using views: A survey. *Very Large Database Journal*, 10(4): 270-294, 2001.

[4] K.G. Herbert, N.H. Gehani, J.T.L. Wang, W.H. Piel and C.H. Wu. BIO-AJAX: An Extensible Framework for Biological Data Cleaning. *ACM SIGMOD Record*, 33(2): 51-57, June 2004.

[5] T. Hernandez and S. Kambhampati. Integration of Biological Sources: Current Systems and Challenges Ahead. *ACM SIGMOD Record*, 33(3):51-60, September 2004.

[6] The ITIS Organization, The Integrated Taxonomic Information System (ITIS) `http://www.itis.usda.gov/`, 15 January 2005.

[7] Z. Lacroix and T. Critchlow. *Bioinformatics: Managing Scientific Data*, Morgan Kaufmann, 2003.

[8] M. Lenzerini. Data integration: a theoretical perspective. In *Proc. of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2002.

[9] B. Ludäscher, A. Gupta, and M.E. Martone. "A Model Based Mediator System for Scientifi c Data Management." Eds. Z. Lacroix and T. Critchlow, *Bioinformatics: Managing Scientific Data*, Morgan Kaufmann Publishers, 2003, pp. 335-370.

[10] W.L. Low, M.L. Lee, and T.W. Ling. "A knowledge-based approach for duplicate elimination in data cleaning." *Information Systems*, 26:8(Dec. 2001): 585-606.

[11] L. Nakhleh, D.P. Miranker, F. Barbancon, W.H. Piel and M. Donoghue. Requirements of Phylogenetic Databases, in *Proc. Of the 3rd IEEE International Symposium on BioInformatics and BioEngineering*, 10-12 March 2003, Bethesda, MD: 141-148.

[12] W.H. Piel, M.J. Donoghue, and M.J. Sanderson. Treebase: A database of phylogenetic information. In *Proceedings of the 2nd International Workshop of Species 2000*, 2000.

[13] R.D.M Page: Phyloinformatics: Towards a Phylogenetic Database, in *Data Mining in Bioinformatics*, Wang, J.T.L, et al., Eds., October 2004.

[14] R.D.M. Page and E.C. Holmes. *Molecular Evolution: A phylogenetic approach*. Blackwell Science, 1998.

[15] E. Rahm and H.H. Do. Data Cleaning: Problems and Current Approaches , in *Bulletin of the Technical Committee on Data Engineering, Special Issue on Data Cleaning*. 23.4 (Dec 2000): 3-13.

[16] M. J. Sanderson, M.J. Donoghue, W. H. Piel, and T. Eriksson. Treebase: A prototype database of phylogenetic analyses and an interactive tool for browsing the phylogeny of life. *Am. J. Bot.*, 81(6), 1994.

Session 5: Sensors and Streams

Source-aware Join Strategies of Sensor Data Streams

Sven Schmidt, Marc Fiedler, Wolfgang Lehner
Database Technology Group
Dresden University of Technology, Germany
dbgroup@mail.inf.tu-dresden.de

Abstract

The growing number of data produced as streams requires sophisticated data stream processing. One challenge comprises join operations on data streams. Basic concepts are derived from Database Management Systems and are extended in terms of window techniques to handle infinite data streams. Within our paper, we show that existing window-based join algorithms are not sufficient for processing data streams of various sensor data sources. As a solution, we propose novel join strategies, which are oriented towards three basic data stream classes. Thereby, we focus on the temporal relation between the stream tuples and introduce the bandwidth-based stream resampling. A real-world example for data streams originating from a casting process accompanies our paper.

1. Introduction

In today's Data Stream Management Systems (DSMS), data streams of various origins (e.g. sensors) flow together for on-the-fly processing and, optionally, for being stored. Sophisticated stream processing may not only consist of operations like filtering or aggregating tuples of individual data streams: for combining data disseminated by different data sources, join operations come into play. Compared to joins over relations of a Database Management System (DBMS), the stream join operator can read the DSMS data only once (unless all historical tuples are buffered), and it can only see the current tuples because data streams are potentially infinite. The following example, first of all, shows that window-based stream join techniques are not sufficient for sophisticated sensor data stream processing, and second, it motivates producing exact join results by adjusting different data rates of the input streams before joining.

Example: Imagine the industrial process of casting metal workpieces. To achieve optimal results in casting quality and in the lifetime of the casting molds, modern foundries currently experiment with equipping the casting mold with sensors (figure 1).

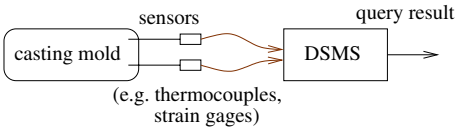


Figure 1. Casting mold monitoring

These sensors mainly measure temperature and pressure (analog signals) at different points of the casting mold and many switcher signals (digital signals) describing the progress of the casting process. These signals form a data stream, which is to be queried by a DSMS.

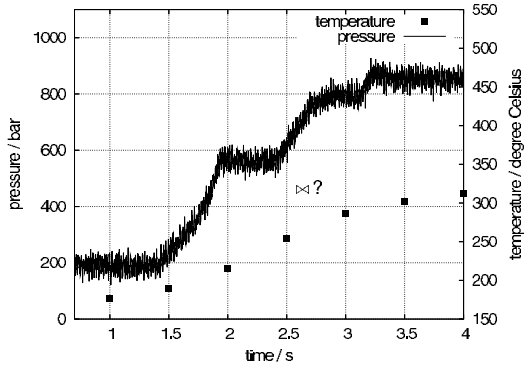


Figure 2. Casting mold's data streams

When analog signals are monitored, the DSMS data source access operator determines the amount of tuples gathered during a time unit. Figure 2 shows the example data streams for measuring the temperature and the pressure inside the casting mold at the time the cast is pushed in. The temperature rises slowly, whereas the casting mold's pressure increases rather suddenly and thus, has to be mea-

sured much more frequently to obtain a good signal representation by the DSMS tuples.

An example query for this scenario may be to join and output measured tuples with temperature and pressure data, if the pressure exceeds a certain threshold. The appropriate query graph is shown in figure 3. We introduce Θ as a symbol for the data source access operation.

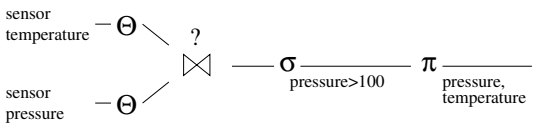


Figure 3. Query for joining sensor values

We assume that the pressure sensor data (timestamp TS, attribute A) arrives with a data rate of 1000 tuples/s, while the temperature sensor data (timestamp TS, attribute B) arrives only with 2 tuples/s. Thus, window-based join techniques (as introduced in [6,7,14]) would maintain two sliding windows for the two input streams and would output a number of tuples (TS, A, B) each time a new tuple arrives at one of the windows. Assumed that the temperature window and the pressure window are of logical size of 0.5 seconds, they contain 500 and 1 tuple(s) resp. The join result is sketched in figure 4.

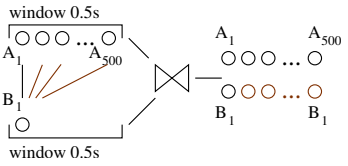


Figure 4. Join result of window-based join

With window-based joins, two problems arise in our application context: first, the attribute value of the stream with the slower input rate (B) remains unchanged during every output of 500 result tuples. It seems that, every 500 result tuples, the attribute B changes in an 'erratic' manner, which is obviously not true for the (original) analog sensor signal. The second problem comprises the sizes of the sliding windows: traditionally, larger join windows are assumed to produce more accurate results. This does not hold for our kind of sensor input data: if the window sizes are chosen arbitrarily, the produced result would lack any semantics because a higher number of result tuples may either be caused by higher sensor input rates or by larger join window sizes – however, the user who observes the join (query) result will be unable to distinguish.

Obviously, we will not come to a solution when applying stream join techniques which operate on single tuples or sliding windows. Our approach of joining such data streams is sketched as query graph for the casting mold example in figure 5.

First, we use a specific data source access operator for the analog sensor input signals, which annotates characteristics of the data stream (e.g. classification information, bandwidth). Second, within a join operator specific to analog sensor data, we either interpolate the stream with the lower bandwidth or we sample down the stream with the higher bandwidth to find *one-to-one join partners*. This enables us to reconstruct intermediate tuple attribute values at any point of the DSMS because the temporal relation between consecutive stream tuples is preserved.

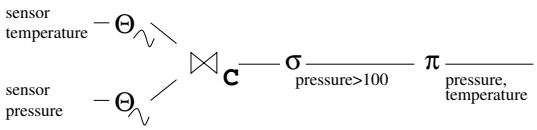


Figure 5. Specialized query graph

Following that, we produce join results with well-defined semantics, i.e. the attribute values of our result tuples *change smoothly* within our example. In detail, the contribution of our paper is the following:

- We propose a classification for data streams based on the data source characteristics.
- We adopt the sampling and resampling techniques from the field of digital signal processing for our stream processing purpose.
- We solve the challenge of the application scenario above by providing join implementations specific to the classes which the data streams belong to.

Structure of this paper: After a summary of related work in the next section, section 3 starts with describing different kinds of data streams. Section 4 inspects the data acquisition process of analog data sources and introduces sampling techniques specific to these stream classes. The main part of the paper is section 5, where we propose a join algorithm for data streams of analog origin. Thereafter, section 6 discusses join algorithms dedicated to other kinds of data streams. Finally, section 7 concludes this paper.

2. Related Work

Much research activity has been directed at data stream systems during the past years. Examples for DSMS-related implementations are [1, 3, 5, 8, 10, 11, 17]. Based on that, a lot of attention has been paid to the management of resource requirements, i.e. to the apriori reservation of resources [2] or to the handling of overload situations by shed-

ding some of the stream load [16]. The load shedding approaches can be based either on naive sampling techniques or on slightly more sophisticated QoS-based user preferences, as in [16]. The problem with this technique of load shedding is that it is impossible to reason quantitatively about the information content reduction or about the resulting quality for the remaining data stream portion. For the class of (sensor-originated) continuous data streams, we will present an approach of downsampling the data stream, which comes along with a well-defined loss of stream information and with a certain reduction of the stream data rate. As opposed to sampling approaches known from Database Management Systems [4], we will refer to the correlation of consecutive DSMS tuple values to apply techniques from the field of digital signal processing. This enables us to reduce information content based on signal processing theorems [15].

Examples for approximated data stream joins are proposed in [6, 7, 14]. The max-subset problem is specifically addressed in [6]. Thereby, gathered statistics form the basis for the applied semantic load shedding techniques for the data streams. The goal is to provide best quality, which is associated with the lowest deviation between the exact and the real join result. Different processing techniques ('fast' and 'slow' CPU) are introduced to avoid that the stream arrival rate will be too high (overload situation). In contrast, we assume to perform only *exact joins* within our approach, due to the exclusive use of the (ordered) timestamp as the join attribute, and due to specific resampling techniques, which we are going to implant into the join operator.

[7] states cost metrics for different join implementations and proposes that it can slow down total join costs, if join implementations are used asymmetrically depending on a unit-time basis cost model and thus, depending on the characteristics of each input data stream. Some of our join implementations may be used asymmetrically, too, but it affects the result semantics. Compared to [7], our goal is to enable the DSMS user to use the appropriate join strategy for his application scenario.

In [14], two approaches of approximation techniques are compared regarding the capability to work with limited memory: The max-subset technique uses statistics in form of so-called 'age curves' to replace stream tuples which do not fit into the limited input windows of the join. As we only join on the timestamp attribute, the amount of output tuples is dependent on the timely discrepancy of the arriving tuples (in case the timeout of the join operator is exceeded) as well as on the intervals each of the input streams is defined for. Within our approach, we do not need any stream statistics (which are obtained during a warm-up phase in [14]). We only rely on stream metadata to classify the data sources and the arising data streams. These metadata are assumed to be available at the time a stream is connected to the DSMS.

3. Data Stream Source Classification

Data processed by a Data Stream Management System comes from various sources. Each data source is a producer, e.g. a sensor, which measures a real or artificial process. In our context, we assume that the DSMS processes tuples of data. In general, a tuple $T_i(TS_i, A_{i_j})$ $(i, j \in \mathbb{N})$ consists of a timestamp TS_i and one or more attributes A_{i_j}. Without loss of generality, we assume in our paper that the tuples produced by a data source have the form (TS_i, A_i). If the data source does not initially associate a timestamp with a measured value, then the timestamp may be associated at the time of entrance into the DSMS to order the values regarding the time domain.

According to the arrangement of the tuples, we can categorize the data sources as well as the produced streams by the characteristic of the timestamp and the characteristic of the measured values (figure 6) following the categorization in [12].

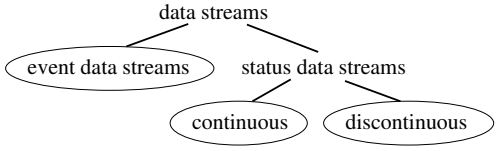


Figure 6. Data stream classification

A *continuous data stream* (*CS*), originating from a sensor measuring physical values such as temperature or pressure of a natural or an industrial process, is uniformly continuous (figure 7a). The sensor is supposed to output analog values which have to be discretized in time and quantized in value. The time discretization of analog signals will be discussed in detail within section 4.1. The obvious need for quantizing the sensor values lies beyond the scope of this paper, because it is dedicated to data integration in general, not to stream processing in particular.

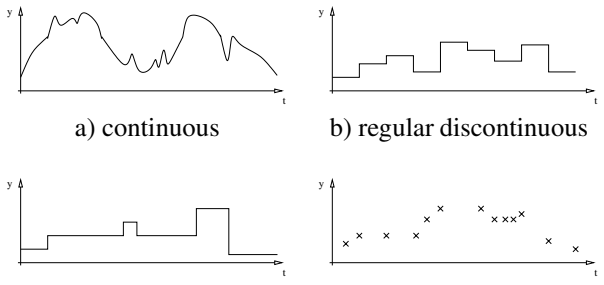


Figure 7. Stream classes

A *discontinuous data stream* (*DS*) contains data values which are constant during certain time intervals. Aggregated data, like sum or average values of business volumes (per day, month, year), may be represented this way. Furthermore, this class can be subdivided into *regular* data streams (figure 7b) and *irregular* data streams (figure 7c). Data, such as supermarket prices with daily updates or stock exchange rates disseminated every hour, minute or second, belongs to the former subclass (constant time intervals), whereas auction prices growing step-by-step, depending on the placed bids, are assigned to the latter subclass.

Finally, we consider the well-known *event data streams* (*ES*). These kinds of data depend on sporadic real-world events and are only defined at the time of the event (as opposed to the former classes). Such streams consist, for example, of tuples coming from network traffic observations or from the monitoring of click-streams (figure 7d).

4. Continuous Data Stream Adaptation

A continuous data stream is an infinite sequence of tuples T_i. The timestamps of two neighboring tuples differ constantly by $\Delta TS = TS_{i+1} - TS_i$. Furthermore, a *bandwidth* BW is assigned to each continuous stream; it describes the maximum information content which may be represented within the stream. It is defined as $BW = \frac{1}{2 \cdot \Delta TS}$. As a specific feature of a continuous data stream, attribute values not present in the stream may be reconstructed completely by applying bandlimited interpolation (see section 4.2).

4.1. Transforming Analog Signals to Continuous Data Streams

To process a continuous analog signal within a digital computer system, the signal curve has to be converted from an analog to a digital state. The discretization in the time domain is called *sampling* (figure 8c).

The samples form the basis for the data stream, which can then be processed using a DSMS. The sampling frequency $f_s = \frac{1}{\Delta TS}$ (or: the sampling rate) depends on the maximum signal bandwidth and determines its maximum information content. Following the sampling theorem [15], the sampling frequency f_s must be larger than two times the signal's highest frequency $f_{sig,max}$:

$$f_s > 2 \cdot f_{sig,max} \tag{1}$$

If condition (1) of the sampling theorem is not met, aliasing effects occur and prevent the correct reconstruction of the original signal. In order to avoid aliasing effects, the signal has to be bandlimited by applying a low-pass filter (figure 8a + b).

In the data stream application, this means that the user defines the maximum signal frequency of interest or the maximum frequency at which a sensor signal is supposed to change, and the sampling frequency can be obtained in a straightforward manner. Then the bandwidth BW always equals the cut-off frequency f_c of the bandwidth delimiter (low-pass filter) and it holds $BW = f_c$.

The result of this initial sampling is a sequence of tuples with equidistant timestamps TS_i and the measured values as attributes A_i.

Summarized, the sampling operation consists of two sub-operations:

1. The bandwidth delimiter reduces the bandwidth of the signal, if necessary depending on the user's interest. At this time, some information will get irrecoverably lost.
2. The consecutive sampling operation does not reduce the information content anymore; it just picks out values of the analog signal which are absolutely necessary to reconstruct a continuous signal of the delimited bandwidth. This operation does not relate to load shedding approaches.

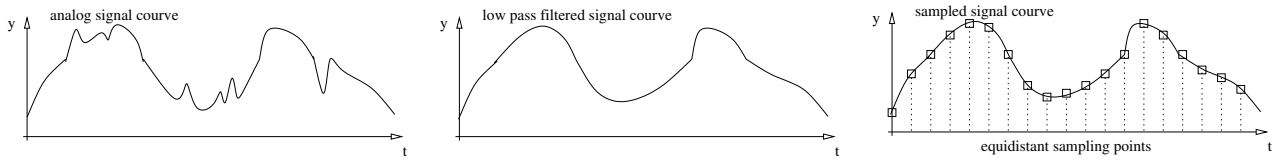
4.2. Bandwidth-aware Sampling of a Tuple Stream – Resampling

Following this initial sampling, the bandwidth of the continuous data stream may be changed at any point during the time the DSMS is applying a resampling operation. Resampling, in our context, means changing the sample rate of a data stream up or down. By applying bandwidth-aware sampling techniques [13], it is possible to interpolate tuple values between consecutive timestamps or to reduce the amount of tuples of data streams with a defined loss of information content.

Signal interpolation and digital filter design is a well-known domain and we refer to [15] for further information. One resampling technique for digital signals is described in [9, 13].

The resampling operation resamples a signal by a factor $\frac{p}{q}$ ($p, q \in \mathbb{N}$), where p is the interpolation factor and q is the downsampling factor. If $\frac{p}{q} < 1$, a low-pass filter has to be applied first to avoid aliasing effects. Depending on the required resampling factor, either p or q may be equal to 1 and thus, one of the sub-operations (interpolation, downsampling) can be left out.

The resampling operation uses a finite impulse response (FIR) low-pass filter [15] with a cut-off frequency f_c for interpolating or bandlimiting a tuple stream. In both cases, a FIR filter kernel with L coefficients of its transfer function is calculated. The cut-off-frequency f_c of the digital low-pass filter depends on the parameters p and q. To interpolate or bandlimit a tuple stream, a stream-based convolution between the filter kernel and the continuous data stream



(a) analog sensor signal (b) applying analog low-pass filter (c) sampling

Figure 8. Sampling an analog input signal

has to be applied. The resulting tuple stream has the bandwidth $BW = f_c$.

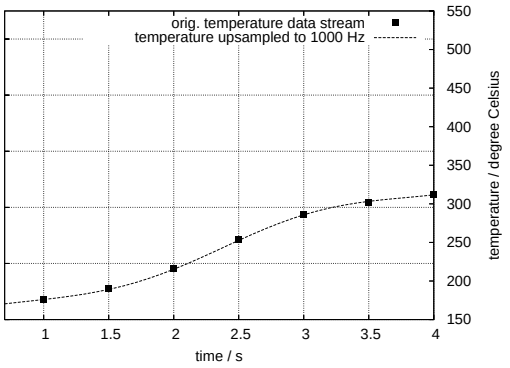


Figure 9. Interpolating the temperature data stream

Bandlimited signal interpolation: Interpolation is needed to reconstruct tuple values between consecutive timestamps. This is done in two steps: firstly, zero-tuples are padded between consecutive timestamps TS_i and TS_{i+1}. That means, that between the tuples at timestamp TS_i and TS_{i+1}, a number of $(p-1)$ zero-tuples are inserted. Secondly, a bandlimited signal interpolation, which uses a low-pass filter with a cut-off frequency $f_c = p \cdot BW$, must be applied.

This results in a tuple stream with a higher data rate and exact interpolated values (figure 9). The application of bandlimited signal interpolation is appropriate for handling signals of analog origin. This operation does not increase the information content of the tuple stream.

Bandlimited downsampling: Bandlimited downsampling is a combination of, first, bandlimiting a tuple stream by applying a low-pass filter with the desired cut-off frequency f_c, and second, passing by only every q-th tuple, because it is sufficient to represent the bandlimited stream. Furthermore, the new tuples do not necessarily lie on the original signal curve, because the information content of the stream was reduced and only the trend of the signal remains after downsampling (figure 10).

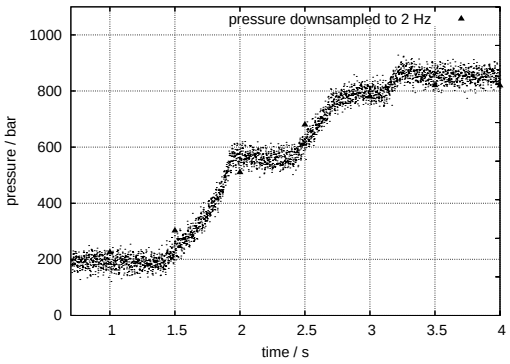


Figure 10. Downsampling the pressure data stream

Summary: The result of the resampling operation in either case is a bandwidth-delimited tuple-oriented data stream. We make use of this when performing join operations on continuous data streams, which is the topic of the following section.

5. Data Stream Joins

Based on the proposed data stream classes, we discuss the different possibilities for joining two streams at a time. The input streams to be joined may be of the same or of different classes and thus, both cases shall be considered. For each of the proposed joins, we sketch an algorithm describing how to perform the appropriate join in a streaming fashion.

5.1. Join Classification

Table 1 provides an overview of the different join options between continuous streams (*CS*), regular and irregular discontinuous streams (*DS*), and event streams (*ES*); it also states the number of the subsection which will handle the appropriate join.

The join between two event-based data streams is left out here (marked with 'x'). For these well-known join techniques, we refer to related works, such as [6, 7, 14].

	CS	DS	ES
CS	5.2	6.2	6.3
DS		6.1	6.3
ES			x

Table 1. Join possibilities

Generally, the join process consists of three steps:

1. The *streams* to be joined must be *compatible*. Based on our classification, joins between all data stream types are possible with the option of preprocessing (e.g. upsampling or downsampling) one input stream.
2. The *suitable stream portions* which are allowed to join must be *identified*. In general, only stream data with the same timestamp are join candidates. However, we allow the user to tolerate a certain 'time shift' between the two input streams.
3. Based on the input data streams, an appropriate join implementation is selected and the specific *join algorithm is applied* as described in the following sections.

5.2. CJOIN: Joining Continuous Data Streams

The CJOIN associates two continuous input streams. If we would simply join streams of different bandwidths, it would be impossible to determine the bandwidth (and thus the maximum information content) of the resulting stream. Thus, our CJOIN implementation works asymmetrically: we adjust different bandwidths either by upsampling the stream with the lower bandwidth or by downsampling the stream with the higher bandwidth. The latter comes along with reducing the information content of one input stream. Figure 11 illustrates the two possibilities for joining continuous streams. The desired strategy is up to the user and depends on the required query result.

Due to the data source dissemination characteristics and due to previous DSMS operations, the input streams may not be defined all the time but only during certain time slices (figure 11). When joining such continuous streams, the result stream is defined only during those slices where *both* of the input streams contain tuples.

The CJOIN implementation is based on three components (figure 12): an optional interpolation or downsampling component and one component for identifying join partners and thus, for performing the join.

We assume the stream, which may pass the two preprocessing components, to be $CS1$ (which becomes $CS1'$ and $CS1''$ resp.). The stream $CS2$ is forwarded directly to the join component. Whether or not stream $CS1$ has to pass one or both of the preprocessing components depends on the required resampling factor $\frac{p}{q}$, which is the pruned fraction of

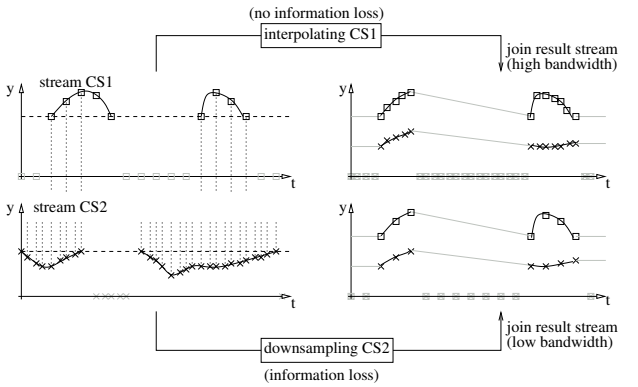


Figure 11. Joining two signal curves of different bandwidth

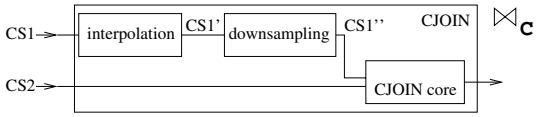


Figure 12. CJOIN components

the two streams' bandwidths $\frac{BW_{CS2}}{BW_{CS1}}$: If $p > 1$ or $q > 1$; an interpolation or a filtering and downsampling resp. has to be applied before joining.

Algorithms 1 and 2 describe the bandlimited signal resampling process (announced in section 4) which we adopted to work in the streaming context without exposing any pipeline-breaking behavior, which we call *stream-based convolution*. Both algorithms show a warm-up phase during which a number of $L - 1$ result tuples must be discarded because they would result in wrong attribute values. In both cases, L is the number of stored filter coefficients which depend on the digital filter's order and thus, on its quality properties. Realistic values range from 20 to 200 (see [9] for details). Furthermore, after interpolation and downsampling, the streams' attribute values become phase-shifted by $\frac{L-1}{2}$ tuples. This is a typical property of digital filters and very important for the following core join component.

The CJOIN core is described in algorithm 3. Both input streams are continuously scanned to find tuples of the preprocessed stream $CS1''$ which match with tuples of stream $CS2$ regarding their timestamp. Two points must be considered: first, due to the preprocessing, the tuples from $CS1''$ arrive later and phase-shifted compared to tuples from $CS2$ (figure 13). We take care of the 'later arrival' by buffering a number of $(L_p - 1) + (L_q - 1)$ tuples of $CS2$. Due to the phase shift, the number is reduced to $\frac{(L_p-1)+(L_q-1)}{2}$ tuples. Thereafter, the first tuple of $CS1''$ joins with the

Algorithm 1 interpolation ($CS1 \rightarrow CS1'$)

Require: input stream $CS1$; output stream $CS1'$
filter kernel size L_p; buffer (FIFO) B_p of size L_p
interpolation factor p
H_p := create_filter_kernel (p, L_p)

while NOT EOS ($CS1$) **do**
insert (B_p, read_tuple($CS1$))
// zero padding
for $i := 1; i < p; i++$ **do**
insert ($B_p, 0$)
end for
//convolution with filter kernel
write_tuple ($CS1'$, stream_conv (B_p, H_p))
end while

Algorithm 2 downsampling ($CS1' \rightarrow CS1''$)

Require: input stream $CS1'$; output stream $CS1''$
filter kernel size L_q; buffer (FIFO) B_q of size L_q
downsampling factor q
H_q := create_filter_kernel (q, L_q)
$count := 0$

while NOT EOS ($CS1'$) **do**
insert (B_q, read_tuple($CS1'$))
//convolution with filter kernel
$temp$:= stream_conv (B_q, H_q)
// discard every q-th output tuple
if $count == q$ **then**
write_tuple ($CS1'', temp$)
$count := 0$
end if
$count++$
end while

$(\frac{(L_p-1)+(L_q-1)}{2} + 1)$-th tuple of $CS2$. As a consequence, the first $\frac{L_p+L_q}{2}$ tuples of stream $CS2$ must be discarded, too.

Second, the exact timestamps of $CS1''$ tuples depend on interpolation and downsampling and will rarely match exactly the timestamps $TS2$ of $CS2$. To produce a join result, we assume that within the resampled data stream $CS1''$, a tuple is valid during the time $\Delta T = \frac{1}{BW_{CS1''}}$. This allows

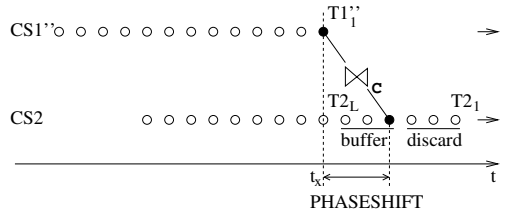


Figure 13. Stream phase shift

Algorithm 3 join ($CS1'' \bowtie_C CS2 \rightarrow R$)

Require: input streams $CS1''$, $CS2$ of bandwidth BW_{CS2}
output stream R; filter kernel sizes L_p, L_q
buffer B for holding $\frac{L_p}{2} + \frac{L_q}{2}$ tuples of $CS2$

$PHASESHIFT := (\frac{L_p}{2} + \frac{L_q}{2}) \cdot \frac{1}{BW_{CS2}}$

// initial read
$(TS1, V1)$:= READ ($CS1''$)
B := fillbuffer ($CS2$)
$(TS2, V2)$:= READ (B)

while (NOT EOS ($CS1''$) AND NOT EOS ($CS2$)) **do**
while $TS1 < (TS2 - PHASESHIFT - \frac{T}{2})$ **do**
(TS1, V1) := read ($CS1''$) // read from stream
end while
while $TS1 > (TS2 - PHASESHIFT + \frac{T}{2})$ **do**
$(TS2, V2)$:= read (B) // read from buffer
insert (B, read_tuple($CS2$)) // write to buffer
end while
if $|TS1 - (TS2 - PHASESHIFT)| \leq \frac{T}{2}$ **then**
WRITE ($TS2, V1, V2$)
end if
end while

us to assign the timestamp $TS2$ of the $CS2$ tuple to the result.

Furthermore, a user-defined time shift may be added to PHASESHIFT, in case one stream's incoming tuples are delayed somehow. Obviously, additional buffer space must be allocated for such situations. The CJOIN result will be a *continuous stream*, independently from the resampling strategy.

Example: To point out the necessity for the proposed join algorithms, we illustrate the join between the temperature and the pressure data streams from the casting mold example. If the user does not tolerate any information loss during the join operation, the temperature stream must be interpolated up to the bandwidth of the pressure stream (from 2Hz to 1000Hz). Thereafter, each pressure stream tuple can be associated with a tuple from the temperature stream (figure 14) and further filtering can be applied.

The DSMS query for the casting mold example scenario may be expressed as

```
SELECT time,
       pressure,
       temperature
FROM pressure JOIN temperature
USING CJOIN (SAMPLING=upsampling)
WHERE pressure > 100
```

The join result is a continuous stream with a bandwidth of 1000 Hz. Otherwise, if the user is interested in a result

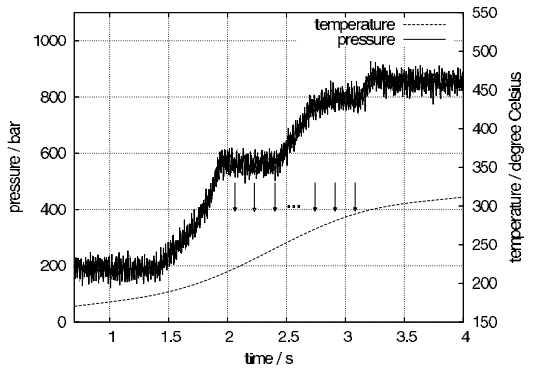


Figure 14. CJOIN example: temperature stream upsampled

stream of the temperature stream's bandwidth and thus, tolerates losing details of the pressure stream, the pressure stream has to be low-pass-filtered and sampled down to a bandwidth of 2 Hz. As a consequence, each resulting pressure stream tuple can be associated with an original temperature stream tuple (figure 15). The CJOIN parameter must be `SAMPLING=downsampling` in that case.

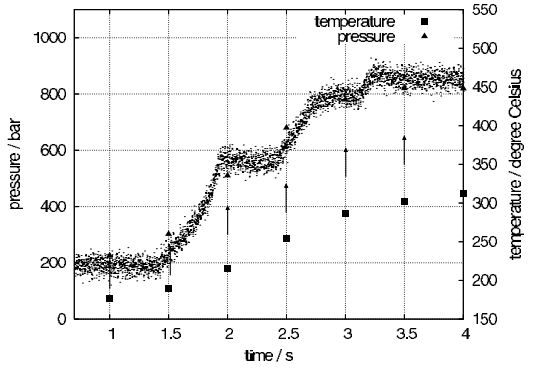


Figure 15. CJOIN example: pressure stream downsampled

6. Further Join Semantics

In the following sections, we extend our join concept to other data stream classes.

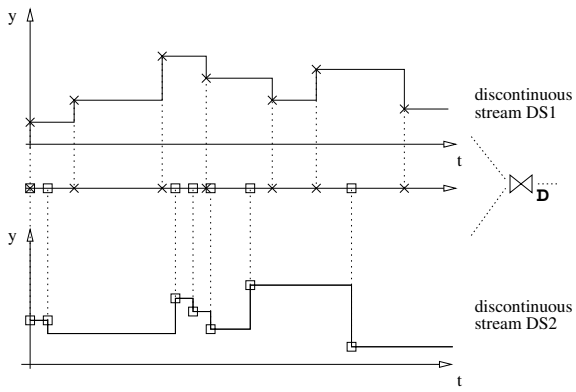


Figure 16. DJOIN

6.1. DJOIN: Joining Discontinuous Data Streams

Discontinuous data streams do not have a bandwidth property assigned. The exact values and timestamps (of the 'jumps' in the signal curve) are important and have to be considered for the join operation. Figure 16 illustrates the most common case, the join of two discontinuous, irregular data streams. Discontinuous streams must not be sampled up or down - this would bastardize the information contained within the streams. Thus, for performing the appropriate join, we propose to consider all tuples of both input streams for constructing the result tuples as follows:

- Create an output tuple at each timestamp TS at which a tuple from one of the input streams arrives.
- If a tuple at timestamp TS arrives from both input streams, pick up the tuple $(TS1, V1)$ from stream $DS1$ and $(TS2, V2)$ from stream $DS2$ and construct the output triple as $(TS1, V1, V2)$.
- In case only one input stream (e.g $DS1$) has a tuple defined at TS, construct the output triple as $(TS1, V1, V2_{last})$ where $V2_{last}$ is the value of the last arriving tuple of $DS2$ (which has to be kept in buffer). If only $DS2$ contains a tuple of timestamp TS, apply the same strategy the other way around.

The result of this most general join between two discontinuous data streams is a stream with the maximum data rate $r_{result} = r_1 + r_2$, if no two input tuples were of the same timestamp.

The data rate may be reduced further if

a) both discontinuous input streams are *regular* and

b) the input streams are 'synchronized regarding time'.

This means, that the dissemination points of tuples follow some general time, such as a clock or a calendar, and contain at least one tuple of the same timestamp (hourly,

daily, monthly etc.). Then, the output data rate is reduced to $r_{result} = r_1 + r_2 - \frac{1}{LCM(P_1', P_2')}$, whereby $LCM(P_1', P_2')$ stands for the lowest common multiple of the whole numbered period lengths of P_1 and P_2. With that periodicity, a tuple of exactly the same timestamp will be disseminated from $DS1$ and $DS2$.

As an example, tuples of $DS1$ arrive every hour, whereas tuples of $DS2$ arrive every day. If the streams are synchronized, every 24 hours a tuple of both streams with identical timestamp will arrive:

$$\begin{aligned} r_{result} &= r_1 + r_2 - \frac{1}{LCM(P_1', P_2')} \\ &= 1\frac{tuple}{hour} + \frac{1}{24}\frac{tuple}{hour} - \frac{1}{LCM(24,1)\frac{hours}{tuple}} \\ &= 1\frac{tuple}{hour} \end{aligned}$$

Again, if the input streams are not synchronized, the result data becomes $r_{result} = r_1 + r_2$. Furthermore, if one of the input streams is not defined for a certain period of time (e.g. due to previous stream operators), no join results will be produced during that time.

If two discontinuous regular streams $DS1$ and $DS2$ are joined, and if it holds that $P_1 = n \cdot P_2$ with n being an integer value, then a *discontinuous regular* stream will be produced. Otherwise (and if one of the input streams is irregular), the result stream will be *discontinuous* and *irregular*.

6.2. CDJOIN: Joining Continuous with Discontinuous Data Streams

Figure 17 illustrates the join characteristics of a continuous (CS) and a discontinuous (DS) data stream: the continuous data stream is described by its samples, whereas the characteristics of the discontinuous (regular or irregular) stream are the timestamps and the erratic attribute value changes. Therefore, we propose two join strategies:

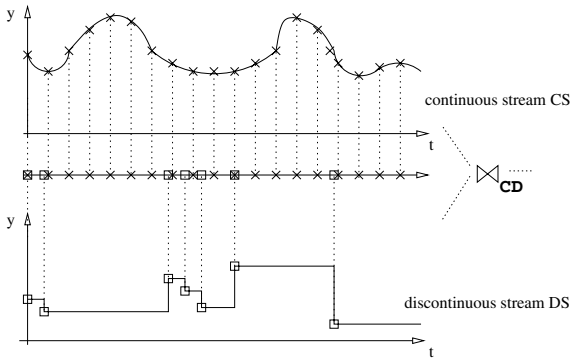


Figure 17. CDJOIN

Strategy 1: If the discontinuous data stream is irregular, the join has to be performed asymmetrically as described in the following: *strategy (1a)* performs the join based on the discontinuous stream and consists of two steps: first, picking up the last arrived value $V1_{last}$ of CS for each timestamp $TS2$ of a DS tuple $(TS2, V2)$, and second, outputting triples like $(TS2, V1_{last}, V2)$.

If the distance in time between tuples of CS is too long and thus, the deviation of using the last arrived tuple is too large, CS may be upsampled before joining.

As a general drawback of this solution, many tuples (or samples) of CS are thrown away because they do not find a join partner. The result stream is heterogeneous: the sequence of the values of DS can be seen as discontinuous and irregular, whereas the values of CS consists of incoherent events and thus, would form an event sub-stream.

As a second possibility, *strategy (1b)* reads the (timely equidistant) tuples of the continuous stream CS and pairs the currently valid value of the discontinuous stream DS with them. Thus, the result triple will look like $(TS1, V1, V2_{last})$, where $V2_{last}$ is the value of the last arriving tuple of DS which has to be kept in the buffer (see section 6.1). The drawback of this strategy lies in losing the exact timestamps of the value jumps in DS. The result stream is heterogeneous again: the values $V1$ of CS form a continuous sub-stream, whereas the values $V2_{last}$ have to be seen as incoherent events.

Strategy 2: If the discontinuous stream DS is regular, it is possible to resample the continuous stream CS to have exactly the same period length as DS (the other way around is impossible because discontinuous data streams must not be resampled!). Then, we follow strategy 1(a) to pick up the closest value of CS for every timestamp DS contains. The result is a combination of a continuous *and* a discontinuous regular attribute stream.

6.3. EJOIN: Joining with Event Data Streams

Joining different stream classes with an event data stream ES works asymmetrically and comprises going through the event data stream ES $(TS1, V1)$ and finding partner tuples $(TS2, V2)$ in the other data stream. The explicit events of ES must not be tampered in their timestamps or in their values (as the continuous or discontinuous streams may be) and thus the procedure is the following: every time a tuple of ES with timestamp $TS1$ is read, a partner of the continuous (CS) or the discontinuous (DS) stream is acquired. Should a partner tuple for the exact timestamp $TS1$ be found, it may be used naturally. The result triples have the form $(TS1, V1, V2_{last})$.

7. Summary and Conclusion

Within our paper, we provided motivation for joining data streams with regard to specific stream classes. Therefore, we proposed different join algorithms, and we reasoned about the semantics of the individual join results. We showed that the DSMS can offer support in terms of providing basic join implementations, but it is up to the user to supply join parameters depending on the application context or the desired query result.

References

[1] D.J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S.B. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB Journal*, 12(2):120–139, 2003.

[2] H. Berthold, S. Schmidt, W. Lehner, and C.-J. Hamann. Integrated resource management for data stream systems. *Proceedings of the Annual ACM Symposium on Applied Computing (SAC'05, March 13-17, Santa Fe (NM), USA)*, 2005.

[3] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Fred Reiss, and Mehul A. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *Proceedings of Biennial Conference on Innovative Data Systems Research (CIDR'03, January 5-8, Asilomar (CA), USA)*, 2003.

[4] W.G. Cochran. *Sampling Techniques*. Wilay, New York, 3. edition, 1977.

[5] Charles D. Cranor, Theodore Johnson, Oliver Spatscheck, and Vladislav Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of the International Conference on Management of Data (ACM SIGMOD'03, June 9-12, San Diego (CA), USA)*, pages 647–651, 2003.

[6] Abhinandan Das, Johannes Gehrke, and Mirek Riedewald. Approximate join processing over data streams. In *Proceedings of the International Conference on Management of Data (ACM SIGMOD'03, June 9-12, San Diego (CA), USA)*, pages 40–51, 2003.

[7] Jaewoo Kang, Jeffrey F. Naughton, and Stratis D. Viglas. Evaluating window joins over unbounded streams. In *Proceedings of the International Conference on Data Engineering (ICDE'03, March 05 - 08, Bangalore, India)*, pages 341–352, 2003.

[8] Jürgen Krämer and Bernhard Seeger. Pipes - a public infrastructure for processing and exploring streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'04, June 13-18, Paris, France)*, pages 925–926, 2004.

[9] The MathWorks. *Signal Processing Toolbox User's Guide*, 2004. `http://www.mathworks.com/access/helpdesk/help/toolbox/signal/`.

[10] Rajeev Motwani, Jennifer Widom, Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Gurmeet Singh Manku, Chris Olston, Justin Rosenstein, and Rohit Varma. Query processing, approximation, and resource management in a data stream management system. In *Proceedings of Biennial Conference on Innovative Data Systems Research (CIDR'03, January 5-8, Asilomar (CA), USA)*, 2003.

[11] S. Schmidt, H. Berthold, and W. Lehner. Qstream: Deterministic querying of data streams (demo). *Proceedings of International Conference on Very Large Data Bases (VLDB'04, August 30 - September 3, Toronto, Canada)*, 2004.

[12] Arje Shoshani and Kyoji Kawagoe. Temporal data management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB'86, August 25-28, Kyoto, Japan)*, pages 79–88, 1986.

[13] J. O. Smith and P. Gossett. A flexible sampling-rate conversion method. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'84, March 19 - 21, San Diego (CA), USA)*, volume 9, pages 112–115, 1984. expanded version available under: `http://ccrma.stanford.edu/~jos/resample/`.

[14] Utkarsh Srivastava and Jennifer Widom. Memory-limited execution of windowed stream joins. In *Proceedings of the International Conference on Very Large Databases (VLDB'04, August 30 - September 3, Toronto, Canada)*, pages 324–335, 2004.

[15] Samuel D. Stearns and Don R. Hush. *Digital Signal Analysis*. Prentice Hall, 2nd edition, 1990.

[16] Nesime Tatbul, Ugur Çetintemel, Stanley B. Zdonik, Mitch Cherniack, and Michael Stonebraker. Load shedding in a data stream manager. In *Proceedings of International Conference on Very Large Databases (VLDB'03, September 9-12, Berlin, Germany)*, pages 309–320, 2003.

[17] Stanley B. Zdonik, Michael Stonebraker, Mitch Cherniack, Ugur Çetintemel, Magdalena Balazinska, and Hari Balakrishnan. The aurora and medusa projects. *IEEE Data Eng. Bull.*, 26(1):3–10, 2003.

Using Linear Models to Monitor the Physical World with Sensors *

Fatih Emekci† Sezai E. Tuna‡ Divyakant Agrawal† Amr El Abbadi†
Department of Computer Science† Department of Elec. and Comp. Engineering‡
University of California Santa Barbara
Santa Barbara, CA 93106, USA
{fatih,agrawal,amr}@cs.ucsb.edu† emre@ece.ucsb.edu‡

Abstract

Recent advances in hardware technology facilitate applications requiring a large number of sensor devices, where each sensor device has computational, storage, and communication capabilities. However these sensors are subject to certain constraints such as limited power, high communication cost, low computation capability, presence of noise in readings and low bandwidth. Since sensor devices are powered by ordinary batteries, power is a limiting resource in sensor networks and power consumption is dominated by communication. In order to reduce power consumption, we propose to use a linear model of temporal, spatial and spatio-temporal correlations among sensor readings. With this model, readings of all sensors can be estimated using the readings of a few sensors by using linear observers. Since a small set of sensors are accessed for query processing, communication is significantly reduced. Furthermore, sensors are usually deployed over hostile environments where failure of sensors is common. In fact, it is quite possible that readings from unreachable sensors are needed. Therefore, fault tolerant monitoring techniques are needed to estimate the readings of the unreachable sensors. We propose a fault tolerant monitoring system using linear models and linear observers.

1. Introduction

Due to advances in miniaturization, low power, and low cost design of sensors, large-scale sensor networks are being deployed to monitor environmental, physical and chemical processes. Examples include environment monitoring on Great Duck Island and James Reserve [2, 9]. In sensor networks, each sensor can be modeled as a full fledged computer with computational, communication, and sensing capabilities. Therefore, sensor networks can be thought of as large scale distributed systems. However, these sensor networks are subject to several constraints such as limited power, high communication cost, low computation capability, presence of noise in readings and low bandwidth. Because of these constraints, techniques for distributed systems, databases, and data stream management cannot be applied directly to sensor networks.

One way to interact with sensor networks is through declarative queries [15]. Basically, these are used to collect the desired data from a sensor network. Therefore, collecting data from sensor networks can be thought of as query processing. There have been many related research efforts in the database and data stream management areas to process queries efficiently. Traditional database management aims to reduce query response time using indexes. On the other hand, the main goal in the context of data streams is to reduce the storage and computational cost and give fast approximate answers to queries. However, monitoring a system (a system can be any measurable phenomenon in the physical world) with sensors is quite different from query processing over data streams and database management systems. The cost of query execution in sensor networks is not only bounded by computational and storage costs but also bounded by data collection cost. In data stream and database management systems, however, data collection cost is not taken into account explicitly; instead it is assumed that the data is already available. This assumption is quite reasonable in database and data stream management systems which are built on wired systems that do not have energy and bandwidth constraints. This, however, is not true in sensor networks where each sensor is powered by ordinary batteries and has energy and bandwidth constraints which directly affect the quality of monitoring.

Many proposals have been made to reduce the cost of data collection to prolong the lifetime of the sensors. In [7], Madden and Franklin proposed the Fjords architecture for managing multiple queries over many sensors. The system collects the readings of all sensors and tries to compute

*This research was funded in parts by NSF grants NSF grants IIS 02-09112, IIS 02-23022, CNF-04-23336, and EIA-00-80134

common subexpressions among queries only once. Several proposals have been made to process queries in-network such as [16, 8, 17, 18]. In general, in-network aggregation can reduce the power usage by pushing part of the computation into the network. However, these proposals only consider aggregation queries and do not consider multiple queries. Lazaridis and Mehrotra [6] proposed to compress the raw data at each sensor node, then the compressed data is sent to the basestation only when the precision is out of bound. Goel and Imielinski [5] proposed a prediction technique to monitor the environment by applying MPEG encoding techniques in prediction. Elnahrawy and Nath [4] modeled the sensor generated data and use that to reduce noise. Recently, Deshpande et al. [3] proposed to use probabilistic models to drive the data acquisition in sensor networks thus reducing the rate of communication and extending the battery lifetime of sensors.

When monitoring the physical environment, there are usually physical rules relating to data originating from different data sources, i.e, there is a physical rule between readings of sensors. Most of the time, these physical rules can be discovered and modeled using correlations among sensor readings. Once this model is known, the query processor can use this model to observe the environment by collecting data from a few sensors instead of all of them. Furthermore, this model can be used to reduce the noise in the measurements. Our main observation underpinning this work is that if two sensors are close to each other, their readings have temporal, spatial, and spatio-temporal correlations. For example, if two sensors are 100 meters apart from each other then their temperature measurements are correlated. Therefore, if these correlations are determined and modeled using historical data, the query processor can use that model to estimate the readings of all sensors using the readings of a few sensors. Formally, if a system is identified and modeled using a linear model, then that linear model can be used to observe all readings using readings from only a subset of the sensors.

The linear model of sensor readings is not only beneficial in reducing the energy consumption but also in dealing with missing values. Since sensors are often deployed over a hostile environment, some sensors may fail or be unable to communicate with the base station. However, their readings might be needed to answer queries. Unlike traditional database and data stream management systems, any query processing technique for sensors should deal with these unreachable readings. In other words, the query processing technique should be fault tolerant, i.e., queries can be answered using reachable sensors with an acceptable error rate. In order to build such a fault tolerant monitoring systems, we propose to use linear models of correlations among sensor readings to estimate the readings of unreachable sensors by collecting data from reachable sensors.

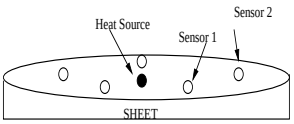


Figure 1. Motivating Example

In order to monitor systems efficiently we propose a monitoring technique called *BINOCULAR*. BINOCULAR models the readings of sensors as a linear system to observe the readings of all sensors using only a small subset of sensor readings. Therefore, it is an energy efficient monitoring system. Furthermore, BINOCULAR balances energy consumption among sensors while providing a fault tolerant monitoring scheme.

The rest of the paper is organized as follows: Background and motivation will be discussed in Section 2. Section 3 formalizes the problem of monitoring systems with queries and gives our solution overview. System modeling is discussed in Section 4. A formal model for observers is introduced in Section 5. The usage of system model and linear observers is discussed in Section 6. Section 7 describes the proposed query processing technique. Section 8 reports the results of our experimental evaluations. Section 9 concludes the paper, and presents future work.

2. Motivation

We now motivate modeling sensor generated data using an illustrative example. Consider a large insulated sheet of thickness b with thermal conductivity k, specific heat s and density ρ and assume that a heat source delivers heat energy E at a point P on the sheet (as shown in Figure 1). Let $u(r,t)$ be the excess temperature t time units later at a point on the sheet that is r away from the heat source. $u(r,t)$ can be expressed as follows: $u(r,t) = (\alpha/t)exp(-r^2/(2\beta t))$ where $\alpha = E/(4\pi bk)$ and $\beta = 2k/(s\rho)$. The derivation of this formula can be found in [13]. Assume we monitor the temperature of the sheet with five sensors located at different points on the sheet *without knowing P and E*. Assume one of the sensors, s_1, is located r_1 distance away and another sensor, s_2 is located $2r_1$ distance away from P. The following observations can be made:

- $T_{2_t} = T_{1_t} exp(-3r_1^2/(2\beta t))$ where T_{1_t} and T_{2_t} are the temperature readings of sensors s_1 and s_2 respectively at time t. (**Spatial Correlation**)
- $T_{2_t} = T_{1_{t/4}}/4$ where $T_{1_{t/4}}$ and T_{2_t} are the temperature readings of sensors s_1 and s_2 at time $t/4$ and t respectively. (**Spatio-temporal Correlation**)
- $T_{2_{t+1}} = T_{2_t} + \frac{\partial u(r,t)}{\partial t}$ where T_{2_t} and $T_{2_{t+1}}$ are the temperature reading of sensor s_2 at time t and $t+1$

respectively. (**Temporal Correlation**)

This physical system can be monitored without knowing these correlations by collecting the readings of the five sensors continuously. However, if these correlations can be modeled with a linear model, then a single sensor reading and the model can be used to estimate the readings of all five sensors.

3. Problem Formulation and Solution Overview

The proposed monitoring technique derives a model of correlations among sensor readings called the *system model* based on the historical data. Then, the derived model is used to execute queries. Given a set of sensors, BINOCULAR divides them into two types: *working* and *sleeping sensors*. In order to estimate the readings of all sensors, BINOCULAR only collects data from the working sensors and uses the system model to estimate the readings of the sleeping sensors.

A system model expresses an estimate for the sleeping sensors in the next time interval (x_{k+1}) based on the current readings of the working sensors (u_k) and the current estimate of the sleeping sensors (x_k). In a linear system model, this is expressed by a linear relationship between x_{k+1} and (x_k, u_k) based on a linear correlation using system matrices A and B. This can be expressed as follows:

$$x_{k+1} = Ax_k + Bu_k, \quad (1)$$

where $x \in R^{n\times 1}$ is the *state* (the estimated readings of the sleeping sensors), $u \in R^{m\times 1}$ is the *input* (the actual readings of the working sensors), $A \in R^{n\times n}$ is the system matrix, $B \in R^{n\times m}$ is the input matrix, m is the number of working sensors and n is the number of sleeping sensors. Matrices A and B can be derived using the system modeling technique discussed in Section 4.

If the correlations among sensor readings can be captured by a perfectly linear model, then we can estimate the exact readings of all sensors using the working sensors. However, in a real physical environment these correlations cannot be perfectly captured by a linear model but need to be approximated by a linear model. Because of the approximation, the error in the estimate accumulates over time. Therefore, the system model should be reset periodically (every D time units) with the actual readings of the sleeping sensors to avoid error accumulation. The appropriate value of D can be derived using the formula discussed in the Appendix.

However, if the readings of sleeping sensors can be estimated by a small subset of the sleeping sensors called *linear observers*, then collecting readings from that subset is enough to estimate the readings of all sleeping sensors. Hence, between resettings, i.e, every D time units, instead of activating all sleeping sensors the linear observers are activated for a short period to recalibrate the errors in the system model. Since the system model is reset periodically with an accurate estimate of the sensor readings, the error does not accumulate over time.

Formally, a linear observer is another linear system built from the original system model. Given a system model in form (1) and a vector of a subset of the sleeping sensors, y_k, of size p, our goal is to determine whether all sleeping sensors can be observable via y_k, referred to as the observer. If it is possible to observe, then we use the linear observer specified by $y_k = Cx_k$ where $C \in R^{p\times n}$ and $C(i, j)$ is 1 if sensor j is in the observer set and it is the only 1 in that row.

Example 1. Consider an environment monitored by three sleeping sensors and one working sensor with a system model:

$$x_{k+1} = Ax_k + Bu_k, \quad (2)$$

where x_k is a 3×1 matrix such that $x_k(i)$ is the estimated reading of sleeping sensor i and u_k is a 1×1 matrix and $u_k(1, 1)$ is the actual reading of the working sensor at time k. If this model is accurate then the actual readings of sleeping sensors need not be collected, because all the readings of the sleeping sensors can be estimated from the reading of the working sensor. However, if the model is a linear approximation, then the error given by the system model will accumulate over time. In order to avoid this, we may collect the readings of the three sleeping sensors every D time units and reset the system model with the actual values. However, if the readings of these three sensors are observable by any one of them, then the readings of the other two sleeping sensors can be estimated accurately by collecting data from that observer. And the system model can be reset with these accurate estimations periodically. For example, if the system model is observable via $y_k = Cx_k$ where $C = [0\ 0\ 1]$ (the reading of the third sensor), then only the reading of the third sensor is collected periodically instead of all three.

The problem of monitoring systems with sensors can be formulated as follows: Given a set of historical readings of sensors, and a set of continuous queries to monitor, building a scheduler to schedule sensors as working or sleeping, discovering a linear model between the readings of working and sleeping sensors, and constructing linear observers and an observer scheduler to execute queries while reducing and balancing the energy consumption. Section 4 discusses how to schedule sensors as working or sleeping and how to discover a linear model of correlation between working and sleeping sensors. Then, Section 5 defines the notion of observability and how to construct a linear observer.

4. System Modeling

Given a set of sleeping sensors, a set of working sensors and their historical readings, the correlations between the readings of working sensors and sleeping sensors can be modeled as a linear system. There are several commercial system identification toolboxes for identifying such correlations as a linear model expressed by Equation (1), e.g, Matlab [10]. Basically these system identification toolboxes generate the matrices A and B in the system model by using some well-known state space identification techniques [12].

Using the same system model (i.e, the same sensors as working sensors) drains the energy of the working sensors and results in unbalanced energy consumption. In order to balance energy consumption, we need a set of system models and need to switch among them over time.

It is possible to find 2^N different system models each of which has a different set of working sensors and a different set of sleeping sensors with N sensors. For small N, all possible models can be derived and energy efficient (i.e, less number of working sensors) and accurate enough models can be used. Energy efficiency and accuracy is a tradeoff for a system model since it will be more accurate if the number of working sensors is increased and vice versa.

On the other hand, it is not practical to derive all possible 2^N models for large N. Therefore, we need a polynomial time heuristic to find these models. The intuition behind our heuristic is that the readings of all sensors are more likely to be modeled by a set of working sensors uniformly distributed over a monitored region. Thus, we use a group of working sensors uniformly distributed over the monitored region. However, the actual data distribution may not be uniform, hence, if the number of working sensors in a group is not enough to yield an accurate model, we increase the number of working sensors. Basically, we divide the sensors into N/K groups (sensors in a group should be evenly distributed over the physical region) with K sensors in each. For each group G, we model the correlation between the readings of the sensors in G (i.e, working sensors) and the rest of the sensors (i.e, sleeping sensors) using historical data. Then, if the accuracy of the model is good enough, that model is used in future estimation. However, if the model is not accurate enough, we increase the number of working sensors in that model. Therefore, each group giving an inaccurate model is merged with another group giving an inaccurate model to yield a more accurate model. This process continues until no group remains to merge. The process is summarized in Algorithm 1.

Algorithm 1 outputs a set of system models. Figure 2 demonstrates an example in which sensors $1, 2, ..., 9$ are distributed over a physical region depicted as a square region. Sensors are initially divided into three groups $G_1 = \{1, 4, 7\}$, $G_2 = \{2, 5, 8\}$ and $G_3 = \{3, 6, 9\}$. The system model, M_1, found by G_1 is accurate enough but G_2 and G_3 are not able to give an accurate model. Therefore we merge these two groups and find a new system model, M_2, with this new group. System models M_1 and M_2 can be switched over time instead of using one of them continuously.

Algorithm 1 System Modeling Algorithm

Input:
R: Physical region covered by sensors;
N: Total number of sensors over R;
K: Total number of sub-regions over R;
Procedure:
Divide R into K sub-regions such that the number of sensors in each sub-region is the same;
Construct K groups $G_1, ..., G_K$ by taking a sensor from each sub-region such that the intersection of any two is empty;
$RemainG = \{G_1, ..., G_K\}$
while TRUE **do**
  for Every G_i in $RemainG$ **do**
    Find a model, M (model of correlations between G_i and the rest of the sensors)
    if M is accurate enough **then**
      Delete G_i from $RemainG$;
      Output M
    end if
  end for
  if $Size\ of\ RemainG = 1$ **then**
    Return;
  end if
  Construct new groups by merging each pair G_i and G_j in $RemainG$.
  Make $RemainG$ empty and put new groups in $RemainG$.
end while
End Procedure

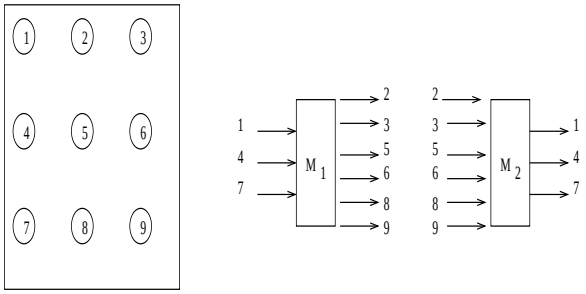


Figure 2. System modeling example

The needed accuracy of a model is application dependent. We assume that users specify the accuracy for their application appropriately.

5. Formal Model For Observers

Linear observers are used to estimate unknown states from a set of measured states. As mentioned earlier, states are readings of the sleeping sensors in our setting. Therefore, the usage of the linear system in our setting is to estimate the readings of all sleeping sensors (i.e, all states) using the readings of a few sleeping sensors (i.e, a set of

measured states). The actual readings of all sleeping sensors are needed to reset the system model periodically because of error accumulation. Instead of collecting all actual readings, they can be estimated accurately with a linear observer that collects readings from a subset.

Given a system model and a matrix, C, where $y_k = Cx_k$, we can construct a linear observer if the pair $(A,\ C)$ is an observable pair which is defined as follows [1]:

Definition 1 *The pair $(A,\ C)$ is said to be an observable pair if the matrix $[C\ CA \ldots CA^{n-1}]^T$ is full column rank (*Observability Condition*).*

Theorem 1 states how and why a linear observer can be constructed with an observable pair $(A,\ C)$ [1].

Theorem 1 *Given a system model and y_k as follows:*

$$x_{k+1} = Ax_k + Bu_k \quad (3)$$

$$y_k = Cx_k\ , \quad (4)$$

If the pair (A, C) is observable, then the states of all sleeping sensors can be observed via y_k using the following linear observer:

$$\hat{x}_{k+1} = A\hat{x}_k + L(y_k - C\hat{x}_k) + Bu_k \quad (5)$$

where L is an observer matrix.

Suppose we are given the system matrix A and we can measure two of the sleeping sensors, say $x(1)$ and $x(3)$, where $x = [x(1)\ \ x(2)\ \ \ldots\ \ x(n)]^T$. Then we can construct a linear observer with a decaying observer error if the $(A,\ C)$ pair is observable where

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \end{bmatrix}$$

The observer error decreases exponentially over time since all eigen values of $A - LC$ are strictly smaller than one. Therefore, we can estimate the readings of all sleeping sensors by collecting the readings of sleeping sensors 1 and 3 as well as the working sensors via the linear observer.

The main challenge now is to derive a set of C matrices such that the system is observable via $y_k = Cx_k$. The number of possible different observers is 2^N, where N is the number of sleeping sensors. However, for large N it is impossible to test all of the possible 2^N cases. Therefore, we propose a technique similar to Algorithm 1 to find a set of observers by trying only $O(N)$ different (A, C) pairs. Basically we divide the sensors into N/K groups where N is the total number of sensors and K is the number of sensors in each group. Then, we decide whether the system is observable from each group or not. For each of the observable groups, we construct an observer. For the remaining groups we merge each group with another and create new groups. Then, we repeat the observability test for these new groups. We repeat these steps until no group remains to merge. When choosing the initial groups we divide the physical region into K sub-regions such that the number of sensors in each sub-region is equal. Then we create initial groups by taking one sensor from each of the K sub-regions. The details of this algorithm are similar to Algorithm 1 and not mentioned here due to space limitations.

6. Using System Model and Observers

There are four ways to estimate the readings of sensors using the system model and linear observers:

Using system model without resetting: The readings of the sleeping sensors are estimated with the readings of working sensors using the system model continuously. Because of the approximations, error accumulates over time. As shown in Figure 3(a), the error can be arbitrarily large after some time.

Using system model with resetting: The system model is reset periodically to avoid error accumulation. Since the system is reset with the actual readings of the sleeping sensors, the error goes to zero periodically. Figure 3(b) shows the behavior of the error over time.

Periodic Observers: One of the linear observers is accessed periodically to estimate the readings of the sleeping sensors. The system model is reset with the estimated values. Since the system model is reset with the estimated values derived from an observer, the error approaches zero periodically as shown in Figure 3(c).

Continuous Observers: So far, we use the system model continuously and execute linear observers periodically to estimate the readings of sleeping sensors. However, the linear observer given by (5) can also be used continuously to estimate the readings of the sleeping sensors instead of the system model. To balance the energy consumption, an observer scheduler switches among observers over time. Figure 3(d) shows the behavior of the error. The reason for the sinusoidal behavior in Figure 3(d) is that approximations in the model introduce error while observers decrease error. Although this method seems inefficient in terms of energy consumption, it may be beneficial to get more accurate results.

Example 2. Consider a sensor network with m working and n sleeping sensors. Furthermore, the correlations among working and sleeping sensors are modeled by a system model M in the form of (3). Assume M is observable with two linear observers called O_1 and O_2 that are in the form of (5). Moreover, M needs to be reset every 200 time units because of the approximations in the model. We want to estimate the readings of sleeping sensors using M, O_1 and O_2 for 1000 time units. Between resettings the sleeping

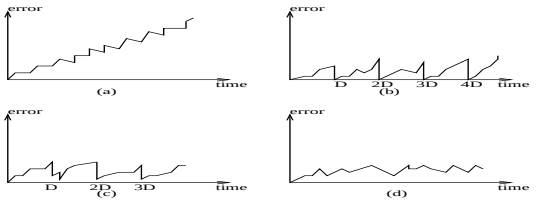


Figure 3. The visual interpretation of observers

sensors readings are estimated by M. If we estimate with only resetting, we need to reset M every 200 time units. In other words, we need to collect readings of all m sleeping sensors at times $200, 400, ..., 800$. In the *periodic observers* case, instead of collecting the readings of all m sleeping sensors, one of O_1 and O_2 can be used periodically for a short amount of time at times $200, 400, ..., 800$ in a Round Robin fashion. Therefore, the readings of all m sensors can be estimated with one of the observers periodically and M can be reset with these estimations using (5). On the other hand, in the *continous observers* case, only one of O_1 and O_2 is used continuously for estimation using (5). For example, only O_1 is used continuously for first 500 time units and then only O_2 is used continuously for last 500 time units.

7. The BINOCULAR System

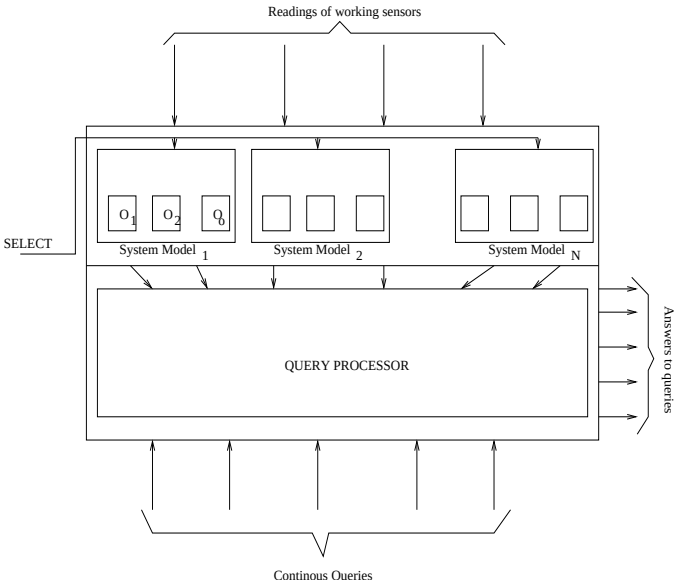


Figure 4. The structure of BINOCULAR monitoring system

The structure of BINOCULAR, which is run in the base station is shown in Figure 4. Users can pose continuous queries and answers to queries are returned to the users. It is the query processor's responsibility to execute queries by using system models and readings of a few sensors. Each system model has several observers.

7.1. Query Processing

The readings of the sensors are needed to answer queries. Since using the same system model results in unbalanced energy consumption, system models are scheduled in a Round-Robin fashion (i.e, each of them is operational the same amount of time). Recall given one of the system models, there are four methods to estimate the readings of the sensors. However, in order to save energy and bandwidth we will use only *periodic observers* and *continuous observers*.

The task of the query processor is to schedule observers to balance energy consumption among sensors. The observer scheduler chooses the observer with the highest score where the score is defined as follows:

$$\frac{Avg.Avaliable.En}{Avg.En.Consumption},$$

where Avg.En.Consumption is the average energy consumption to collect the readings of the observer and Avg.Avaliable.En is the average available energy of sensors in the observer. At any time the observer scheduler chooses the observer with the lowest cost and the highest energy.

The proposed technique does not give any guarantee on the accuracy of the query answer, if the model does not capture the real time correlations. However, if the model captures the correlations in the incoming data, then the historical data can be used to derive the error associated with the query answer. Alternatively, our technique can be used to give exact error guarantees using the methods in [5] where users specify the error bound in the query answer. Basically, in order to give such error guarantees both the base station and the sleeping sensors need to run the system model. Furthermore, the sleeping sensors need to send their actual readings to the base station if the estimated value in the base station is out of the user specified error bound.

7.2. Fault-tolerant Monitoring

Since monitored physical environments are usually hostile environments, it is quite possible to have unreachable sensors (either due to a failure or disconnection from the network). However, their readings are still needed to answer queries. Therefore, any monitoring technique should be *fault tolerant*, i.e, the readings of unreachable sensors can be estimated within an acceptable error rate.

BINOCULAR provides a fault tolerant monitoring technique, since it estimates the readings of unreachable sensors using the readings of reachable sensors, the system models and linear observers. If the working sensors in the system model are reachable and the readings of the sleeping sensors in that model are observable with the reachable sleeping sensors, then that system model can be used to estimate the readings of all sensors (i.e, reachable and unreachable sensors) either with periodic observers or with continuous observers. Therefore, BINOCULAR provides fault tolerant monitoring if the following conditions are satisfied: (1) if there exists a system model M whose working sensors are reachable. (2) If M is observable by reachable sleeping sensors.

Systems usually have more than one system model each of which is observable with a small subset of the reachable sleeping sensors. In this case, the observer determination and scheduling techniques introduced in previous sections can be applied to find reachable working sensors and linear observers when some sensors have failed or are unreachable.

7.3. Probabilistic Extensions to the Linear Model

Recently, Deshpande et al. [3] modeled the spatial correlations among sensor readings with a multivariate Gaussian model. Basically, they denote a model as a probability density function, $p(X_1, X_2, .., X_n)$, where X_i is the reading of sensor i. A simple probabilistic transition model is used to capture temporal correlations. Furthermore, [3] proposed to use the Gaussian model to answer queries with probabilistic confidence instead of exact error guarantees.

In our system, the system model and the linear observers should be run by every sensor as well as the base station in order to provide the exact error guarantee specified by the user. Furthermore, readings of both the working sensors and the observers should be broadcast to every sensor. Once the system model and the linear observers are known to a sleeping sensor, then it can derive the base station estimate by simulating the system model. The sleeping sensor can send the actual reading if the estimated value is out of the user specified error bound.

However, if probabilistic guarantees are sufficient for query posers, our linear modeling approach and the probabilistic approach in [3] can be used together to build a more efficient system. The model in [3] captures spatial correlations using a Gaussian distribution and captures temporal correlations with a simple probabilistic transition model. On the other hand, our linear model exactly captures all of the temporal, spatial and spatio-temporal at the extra cost of performing the calculations at all sensors by simulating the system model. In order to build an efficient system providing probabilistic confidence, a linear model and a Gaussian model could be used together to form a linear Gaussian model. The linear Gaussian model, a well known multi-dimensional time series modeling technique in statistics, is a good technique to model sensor generated data due to its following nice properties [14]: (1) The sum of two independent Gaussian quantities is also Gaussian distributed. (2) The output of a linear system whose input is Gaussian distributed is again Gaussian distributed. (3) It captures temporal, spatial and spatio-temporal correlations.

The following linear Gaussian system can be used to estimate the readings of the sensors [14]:

$$x_{t+1} = Ax_t + Bu_t$$

where x_t is a Gaussian distribution of readings of sensors at time t: $N(\mu_t, \sigma_t)$. Since the above linear system estimates the Gaussian distribution of the readings of the sleeping sensors in the next time interval, the confidence can be calculated with the techniques discussed in [3]. Now, the sleeping sensors do not have to perform calculations (i.e, simulate the system model). Since a linear Gaussian system is again a linear system, the fault tolerant monitoring techniques discussed in this paper can be used. Therefore, a fault tolerant monitoring system giving probabilistic confidence can be built with a linear Gaussian model. The properties of linear models, Gaussian models and linear Gaussian models are listed in Table 1. The three approaches give complementary advantages in terms of computation requirement, fault tolerance and error guarantees.

Table 1. Properties of Models

	Linear	Gaussian	Linear Gaussian
Probabilistic confi dence		√	√
Exact error guarantee	√		
Fault tolerance	√		√

8. Experimental Evaluations

In this section, we present experimental results for BINOCULAR. In our experiments, we measure the average error for each sensor's reading and the average number of messages sent by each sensor. In addition, we measure the fault tolerance of our technique. We use n4sid, a system identification toolbox, provided by [10] in our experiments.

We conduct experiments over the following datasets:

- Intel Lab Data: Temperature data collected from 30 sensors deployed in the Intel Berkeley Research lab between February 28th and April 5th, 2004.
- Ocean Surface Temperature Data: Real temperature dataset from the Tropical Atmosphere Ocean Project [11] consisting of readings of 20 sensors.

8.1. Intel Lab Data

The data set contains $240,000$ readings of 30 sensors. We modeled the correlation between the working sensors and sleeping sensors using $1/3$ of the data. The number of working sensors is varied from 1 to 5. Since the model is a linear approximation, we collected the actual readings of each sleeping sensor every D time units and vary D from 50 to 1000. Observers are not used to estimate the readings of sleeping sensors, because the cost of using an observer is equal to the cost of collecting readings from all sleeping sensors (the derived system model is observable with at least 10 sleeping sensors).

Figure 5 shows the average error in percentage, E, for different D values and different number of working sensors. $E = [(1/30T) * \sum_{t<T} \sum_{i<30} \mid \hat{s_{i_t}} - s_{i_t} \mid / s_{i_t}] * 100$ where $\hat{s_{i_t}}$ is the estimated reading and s_{i_t} is the actual reading of sensor i at time t and T is the total monitoring time. As we expected, the error increased as D increased since error accumulates over a time. However, the increase in the error decreases when the number of working sensors is increased, since the system model is more accurate with two or more working sensors.

The cost of using different D values and varying the number of working sensors are shown in Figure 6 in terms of the average number of messages sent by each sensor. The top line in Figure 6 shows the number of messages that need to be sent in order to collect the exact readings of the sensors. The results show that our method can estimate the readings of the sensors within 2 percent error (i.e, $\leq \mp 0.7^oC$) with 2 working sensors and 28 sleeping sensors by collecting the readings of the sleeping sensors every $D = 100$ time units. Therefore, the cost is reduced dramatically by an order of magnitude compared to collecting the exact readings of the sensors (i.e, top line in Figure 6). The average number of messages sent by each sensor decreases from 8000 to approximately 800. Figure 7 shows the maximum error at any time during monitoring with two working sensors. At time t, Figure 7 shows the maximum error in the prediction of sensor readings.

We also tested BINOCULAR in a hostile environment in which p percent of the sensors are unreachable. We vary p from 15 to 70. BINOCULAR predicts the readings of all 30 sensors with the reachable sensors using continuous observers and periodic observers with only 1 working sensor. The results are shown in Figure 8. The error increases in both continuous observers and periodic observers as the percentage of failures increases, since the number of observer sensors decreases. As we expected, the error is low in the continuous observers case since the error is recalibrated continuously. The cost of using continuous and periodic observers is shown in Figure 9 in terms of *the total number of messages* sent by *all sensors* during *the entire monitor-*

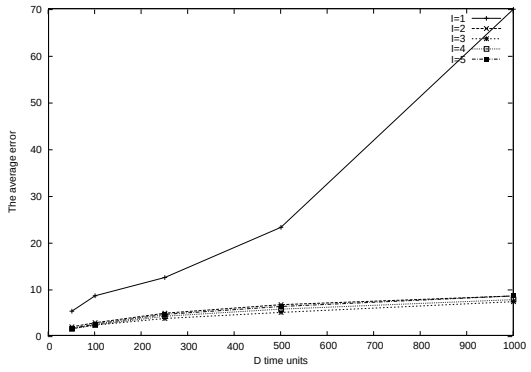


Figure 5. The average error (in percentage) of sensor readings for different number of working sensors varying from 1 to 5 ($I = 1, .., 5$)

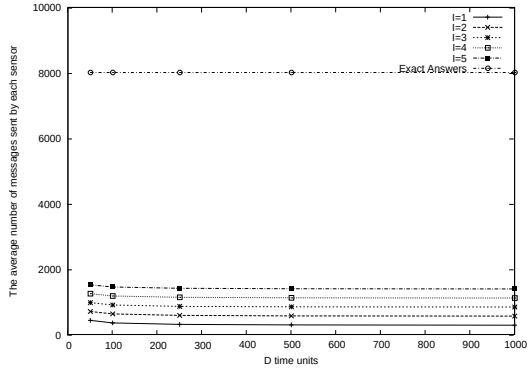


Figure 6. The average number of messages sent by each sensor for different number of working sensors varying from 1 to 5 ($I = 1, .., 5$)

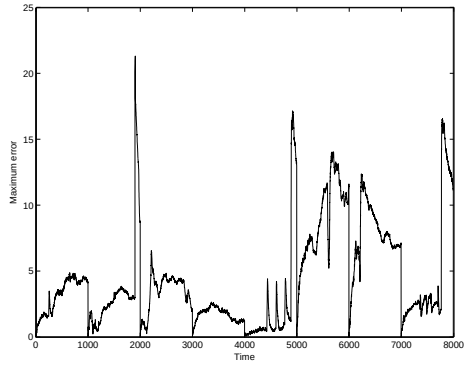


Figure 7. The maximum error (in percantage) in the prediction with two working sensors

ing period. The total number of messages (total overhead on the network) decreases as the percentage of failure increases, since failing sensors are not involved in observers (note that the average number of message sent by each sensor decreases). The results demonstrate that using continuous observers or periodic observers is an energy versus accuracy tradeoff.

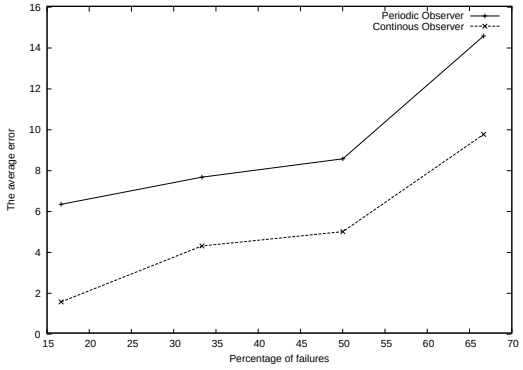


Figure 8. The average error (in percentage) of sensor readings with observers

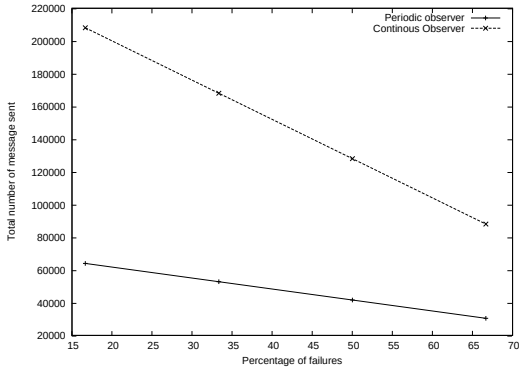


Figure 9. Total number of messages sent by the sensors

8.2. Ocean Surface Temperature Data

BINOCULAR was tested over a real temperature dataset from the Tropical Atmosphere Ocean Project [11]. The average daily temperature readings of 20 sensors for 1000 days were taken. The correlations between the readings of each sensor and the rest of the sensors were modeled based on the first 300 days. Therefore, there are 20 system models, one for each sensor. These models were tried with the remaining 700 days (using each system model for $700/20 = 35$ time units). Then, the average error for time t is calculated as follows: $((1/20) * \sum_{i<20} \mid \hat{s_i} - s_i \mid / s_i) * 100$ where $\hat{s_i}$ is the estimated reading and s_i is the actual reading of a sensor i at time t. With the system models and one working sensor at a time, BINOCULAR can estimate the reading of all sensors within 4.5 percent error on the average (this is $\approx \mp 1.5^oC$) over 700 time units. Figure 10 shows the maximum error at any time during monitoring with two working sensors. At time t, Figure 10 shows the maximum error in the prediction of sensor readings.

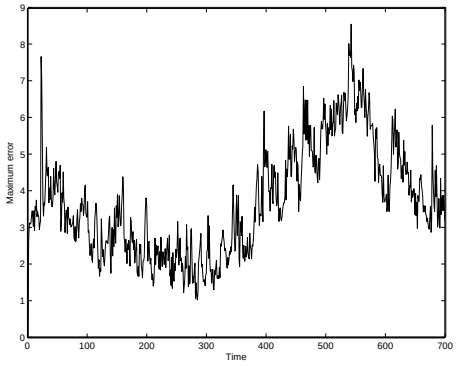


Figure 10. The maximum error (in percentage) in the prediction with two working sensors for Ocean Data

9. Conclusion and Future Work

In this paper we presented a fault tolerant system monitoring framework, BINOCULAR. BINOCULAR uses a linear model between the working sensors and the sleeping sensors to answer queries while using a small set of sensors. We introduced the notion of linear observers to account for the fact that the linear model will always be an approximation of the physical environment. By using the linear observers, the modeling error can be reduced exponentially over time. This results in less communication cost and prolongs the lifetime of sensors. Furthermore, the notions of linear model and linear observers are used to provide fault tolerant monitoring. Since sensors are usually deployed over a hostile environment, it is quite possible to have unreachable sensors whose readings are needed. BINOCULAR provides a novel fault tolerant monitoring system to monitor hostile environments by using linear models and linear observers. Our results show that using linear models and observers reduces the energy consumption significantly and increases the lifetime of sensors with an acceptable error in the estimation of readings of the sensors. As future work, we plan to build a system on top of BINOCULAR to mine the collected data to detect significant events in a physical environment.

Acknowledgment The authors would like to thank Dr. Samuel Madden for providing the Intel Lab data.

References

[1] Pans J. Antsaklis and Anthony N. Michel. *Linear Systems*. The McGraw-Hill Companies, Inc, 1997.

[2] A. Cerpa, J. Elson, D. Estrin, L. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology. *In ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, pages 88–97, 2001.

[3] Amol Deshpande, Carlos Guestrin, Samuel Madden, Joseph M. Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. In *VLDB2004*, pages 588–599, 2004.

[4] Eiman Elnahrawy and Badri Nath. Cleaning and querying noisy sensors. In *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications*, pages 78–87. ACM Press, 2003.

[5] S. Goel and T. Imielinski. Prediction-based monitoring in sensor networks: Taking lessons from mpeg. *ACM Computer Communication Review*, 31(5):82–98, October 2001.

[6] Iosif Lazaridis and Sharad Mehrotra. Capturing sensor-generated time series with quality guarantees. *International Conference on Data Engineering (ICDE 2003)*, pages 429–440, March 2003.

[7] S. Madden and M.J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. *International Conference on Data Engineering (ICDE 2002)*, pages 555–566, March 2002.

[8] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. *5th Symposium on Operating System Design and Implementation*, pages 131–146, December 2002.

[9] A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler. Wireless sensor networks for habitat monitoring. *In ACM Workshop on Sensor Networks and Applications*, pages 88–97, 2002.

[10] Matlab. http://www.mathworks.com/products/sysid/.

[11] M. J. McPhaden. Tropical atmosphere ocean project. *Pacific marine environmental laboratory*. http://www.pmel.noaa.gov/tao/.

[12] Van Overschee P. and De Moor B. N4sid : Subspace algorithms for the identifi cation of combined deterministic-stochastic systems. *Automatica, Special Issue on Statistical Signal Processing and Control*, pages 75–93, 1994.

[13] K. F. Riley, M. P.Hobson, and S.J. Bence. *Mathematical Methods for Physics and Engineering*. Cambridge, 2002.

[14] S. Roweis and Z. Ghahramani. A unifying review of linear gaussian models. *Neural Computation*, pages 305–345, 1999.

[15] TinyDB. http://telegraph.cs.berkeley.edu/tinydb.

[16] Anantha Chandrakasan Wendi Rabiner Heinzelman and Hari Balakrishnan. Energy-effi cient communication protocol for wireless microsensor networks. *Hawaii International Conference on System Sciences (HICSS)*, page 8020, January 2000.

[17] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3):9–18, 2002.

[18] Yong Yao and Johannes Gehrke. Query processing for sensor networks. *In CIDR 2003*, January 2003.

A. Appendix

The linear model we derive from the historical data may not match precisely to the data due mainly to that the actual system probably has nonlinear dynamics and that the data collected has measurement error. Therefore the linear model we are given by the software being used to obtain that model will only be an approximation to the actual behavior of the real system. One way to account for the discrepancy between the approximate model and the real system can be by adding a disturbance term $d \in \mathbb{R}^n$ to the linear model as such

$$x_{k+1} = Ax_k + Bu_k + d_k \ . \tag{6}$$

That d term in the above equation will divert the simulated system from the actual one in time. Note that if d is zero then the approximate system will become exact and the smaller d is the longer it takes the two systems to move apart from each other. Let us now formulate this intuition. Let the simulated system be

$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k \tag{7}$$

and $x_0 = \hat{x}_0$. If we define $e_k := x_k - \hat{x}_k$ then we can write

$$e_{k+1} = Ae_k + d_k \tag{8}$$

hence it follows, since $|e_0| = 0$,

$$e_k = \sum_{i=0}^{k-1} A^i d_i \ , \tag{9}$$

for $k \in \mathbb{N}$. Now if we suppose $|d_k| \leq \varepsilon$ for all $k \in \mathbb{N}$ then we can write

$$|e_k| \leq \varepsilon \sum_{i=0}^{k-1} \alpha^i \tag{10}$$

$$= \varepsilon \frac{\alpha^k - 1}{\alpha - 1} \tag{11}$$

where $\alpha := |A|$ (suppose it is greater than one). Hence given some $\delta > 0$ we can write $|e_k| \leq \delta$ for all $k \in \{0, 1, \ldots, D\}$ where

$$D := \left\lfloor \ln\left(\frac{\delta(\alpha - 1) + \varepsilon}{\varepsilon}\right) \Big/ \ln(\alpha) \right\rfloor . \tag{12}$$

Optimizing In-Order Execution of Continuous Queries over Streamed Sensor Data

Moustafa A. Hammad
University of Calgary
Calgary, Alberta, Canada T2N 1N4
hammad@cpsc.ucalgary.ca

Walid G. Aref * Ahmed K. Elmagarmid *
Purdue University
West Lafayette, IN 47907, USA
{aref,ake}@cs.purdue.edu

Abstract

In this paper we study the problem of providing ordered execution of time-based sliding window queries over input streams of sensor data with inherent delays. We present three approaches to achieve the ordered execution. The first approach enforces ordered processing at the input side of the query execution plan. In the second approach we utilize the advantage of out-of-order execution to optimize query operators and enforce an ordered release of the output results. The third approach is adaptive and switches between the first and second approaches to achieve the best overall performance with current input arrival rates and level of multiprogramming. We study the performance of the proposed approaches both analytically and experimentally while using various system configurations. Our performance study is based on an extensive set of experiments using a realization of the proposed approaches in Nile, a prototype stream query processing system.

1. Introduction

Continuous queries on streaming applications depend on windows to limit the scope of interest over the infinite input streams. Several forms of windowed execution are currently proposed in the literature, of which, time-based sliding windows are commonly used by several stream data systems [1, 2, 6, 7]. Figure 1(A) gives the pipelined evaluation of an example continuous query Q that computes the online total count of the items sold in common by two different department stores. Q uses a window w time units. In the figure, the output from joining S and T is streamed as input to the DISTINCT and then to the COUNT operators at the top of the pipeline.

*This research was supported in part by the National Science Foundation under Grants: IIS-0093116, IIS-0209120, and 0010044-CCR.

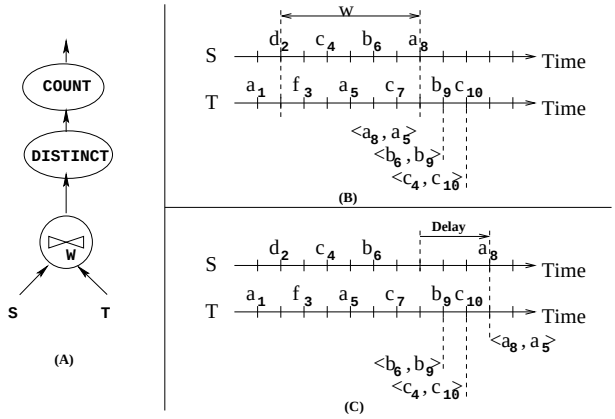
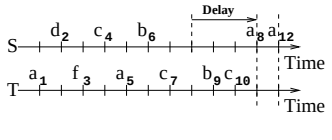


Figure 1. Motivating example.

The operation of the join over a sliding window (W-join) is described as follows [3, 4, 6]: Tuple t_k in Stream S joins with tuple t_j in Stream T iff (1) t_k and t_j satisfy the join predicate (i.e., the WHERE clause in the SQL query), (2) the timestamp of tuple t_k is within window size from the timestamp of t_j. Old tuples, say t_o, from one input stream is expired (dropped from the window) iff t_o is far by more than window size from any new tuples in the other stream. Figure 1(B) gives an example of W-join between streams S and T. The ticks on the time line of S or T are equally spaced at one time unit between two consecutive ticks. We assume that each tuple is indexed by its maximum timestamp (i.e., $TimeStamp(a_k) = k$ and $TimeStamp(< a_i, a_j >) = max(i, j)$). As a_8 arrives, W-join drops a_1 and produces the output tuple $< a_8, a_5 >$. Similarly, as b_9 and c_{10} arrive, W-join drops d_2 and produces the output tuples $< b_6, b_9 >$ and $< c_4, c_{10} >$, respectively.

W-join as described in the previous paragraph can potentially produce an *unordered* output stream. For example, in Figure 1(C), tuple a_8 in Stream S is delayed 3 time units while tuples b_9 and c_{10} in stream T arrive without delays. In this case, W-join will process tuples b_9 and c_{10} before processing the earlier tuple a_8. This will result in an out-



Arrival Time	Output Tulpe	Release Time	Delay
11	$<a_8, a_5>$	11+tp	tp
9	$<b_6, b_9>$	12+tp	3+tp
10	$<c_4, c_{10}>$	12+2tp	2+2tp

tp : time to process a new tuple

Figure 2. The Sync-Filter approach of W-join.

of-order release of the output tuples (i.e., tuples $< b_6, b_9 >$ and $< c_4, c_{10} >$ will be released before tuple $< a_8, a_5 >$).

The notion of ordered output is crucial in the pipelined evaluation, mainly for two reasons: (1) The decision of expiring an old tuple from a stored state (e.g., a stored window of tuples in an online sliding-window $COUNT$ operation) depends on receiving an ordered arrival of the input tuples. Otherwise, we may expire an old tuple early (e.g., potentially report an erroneous sequence of count values). (2) Some important applications over data streams, e.g., as in feedback control, periodicity detection, and trend prediction, require processing the input of their queries in-order (and therefore, produce ordered output). One approach to provide *in-order* execution of input tuples is to synchronize the processing of W-join over the input streams [7]. We call this approach the *Sync-Filter* approach (for synchronize then filter). In this approach, and using the example in Figure 1(C), W-join will delay the processing of b_9 and c_{10} from stream T until verifying that a new tuple from Stream S arrives and has a larger timestamp. The obvious drawback of the *Sync-Filter* approach is that W-join will *block* waiting for new tuples at both streams before every join step. This will result in increased response times of output tuples.

In this paper, we study the Sync-Filter approach in terms of the average response time. Then, we propose a new approach, termed the *Filter-Order* approach, and provide a closed form representation of the average response time. Based on the analytical study, we propose a third approach, termed the *Adaptive* approach, that has the advantages of the two previous approaches while avoiding their drawbacks. We study the three approaches experimentally using our prototype system, Nile, which is a centralized stream data system that executes time-based sliding window queries [6]. The experimental study validates our analytical results and shows that the Adaptive approach can always achieve the targeted improvement in response time by switching between the Sync-Filter and the Filter-Order approaches.

The rest of the paper is organized as follows. Section 2 presents the Sync-Filter approach of W-join. Sections 3 and 4 introduce our proposed approaches, namely the Filter-Order approach and the Adaptive approach of W-join. We present the performance study in Section 5. Section 6 contains concluding remarks.

2 The Sync-Filter Approach

One straightforward approach to get ordered output from the W-join operator is by enforcing ordered processing of input tuples. In other words, for any two tuples t_i and t_{i+1} that are processed in sequence by W-join, $TimeStamp(t_i) \leq TimeStamp(t_{i+1})$. Note that t_i and t_{i+1} may not necessarily belong to the same stream.

Figure 2 gives the execution of the Sync-Filter approach for the example of Figure 1(C). As b_9 in Stream T arrives, W-join blocks waiting for another tuple from Stream S. At time 11, a_8 arrives in Stream S. W-join processes a_8 and removes a_1 from Stream T since a_8 and a_1 are far by more than window (6 time units). Finally, W-join produces the output tuple $< a_8, a_5 >$. Notice that W-join processes b_9 and c_{10} only when tuple a_{12} arrives in Stream S. At time 12, W-join processes b_9 and produces the output tuple $< b_6, b_9 >$ at time $12 + t_p$, where t_p is the time to process an input tuple by the W-join. Then, W-join processes c_{10} and produces the output tuple $< c_4, c_{10} >$ at time $12 + 2t_p$. The delay in processing every tuple is given in the rightmost column of the table in Figure 2.

The advantage of the Sync-Filter approach, besides its simplicity and guaranteed provision of ordered output, is that W-join needs to store only those tuples that are within window from each other. Notice that tuples b_9 and c_{10} are not stored in the buffer of Stream T. Instead, b_9 and c_{10} are kept in the input queue [1]. In addition, W-join drops old tuples as new tuples are processed (e.g., dropping a_1 when W-join processes a_8). Therefore, the Sync-Filter approach eliminates the need to check the window condition (i.e., that tuples are within window from each other) while scanning the buffer of the joined stream.

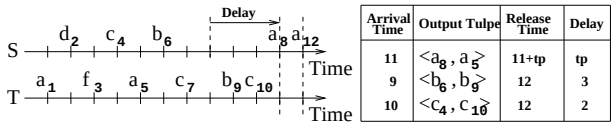
One drawback of the previous approach is that W-join blocks while waiting for a delayed tuple from one stream (e.g., a_8) even though some tuples (e.g., b_9 and c_{10}) could be waiting to join in the other stream. A better approach is to *overlap* the time of processing the waiting tuples with the waiting time to receive the delayed tuple. Apparently, this new approach has to prevent the out-of-order release of output tuples (see the example in Figure 1(C)).

3 The Filter-Order W-join Algorithm

In the Filter-Order W-join Algorithm (Filter-Order, for short), W-join processes input tuples independent of their global order. Furthermore, W-join buffers the output tuples before releasing them in-order.

Figure 3 gives the execution of W-join using the Filter-Order approach. W-join processes b_9 once b_9 arrives (without blocking to wait for a_8). The output tuple $< b_6, b_9 >$

[1] Notice that the input queue of T will not increase indefinitely since we always assume that tuples from Stream S will eventually arrive.



Arrival Time	Output Tulpe	Release Time	Delay
11	$<a_8, a_5>$	11+tp	tp
9	$<b_6, b_9>$	12	3
10	$<c_4, c_{10}>$	12	2

tp : time to process a new tuple

Figure 3. The Filter-Order approach of W-join.

is stored in the hold buffer and is not released immediately. Similarly, W-join processes c_{10} and stores the output tuple $< c_4, c_{10} >$ in the hold buffer. W-join cannot release the two output tuples since the minimum timestamp of the last tuple seen from Streams S or T, $TS_{trigger}$, equals 6 (< 9). As tuple a_8 arrives at time 11, W-join updates $TS_{trigger}$ to 8, produces $< a_8, a_5 >$ and releases this tuple immediately since $(TimeStamp(< a_8, a_5 >) = 8) \leq TS_{trigger}$. At time 12, tuple a_{12} arrives and $TS_{trigger}$ is set to 10. W-join can now release the output tuples $< b_6, b_9 >$ and $< c_4, c_{10} >$. Notice that the time to produce $< b_6, b_9 >$ and $< c_4, c_{10} >$ is overlapped with the waiting time to receive a_{12} and the total delay to receive the three output tuples is lower than that of the Sync-Filter approach by 3 tp.

By comparing the average response time of the Filter-Order approach with that of the Sync-Filter approach, it is clear that the processing time overlaps the waiting time. Therefore, the average output response time is expected to improve when using the Filter-Order approach. Let the time to perform a join operation between two tuples be c. Let λ_1 tuples/second be the average arrival rate of Stream S and let λ_2 tuples/second be the average arrival rate of Stream T. Let $|w|$ is the window size in seconds. The relative improvement in average response time when using the Filter-Order approach over the Sync-Filter approach[2], I_{Rel}, is:

$$I_{Rel} = \frac{c|w|\lambda_1\lambda_2}{1 + c|w|\lambda_1\lambda_2} \quad (1)$$

4 The Adaptive Algorithm

Equation 1 shows that the relative performance improvement when using the Filter-Order approach is significant at specific ranges of arrival rates and processing speeds. Otherwise, the Sync-Filter approach is a valuable option especially as we consider the low memory overhead in the Sync-Filter approach. In this section we introduce the Adaptive approach that switches between the Sync-Filter and the Filter-Order approaches to achieve the best average response time. Initially the Adaptive W-join algorithm adopts the Sync-Filter approach, while performing two extra steps. Step 1: Monitor λ_1 and λ_2 (the arrival rates at the input data streams S and T, respectively.) Step 2: Verify the following condition: $c|w|\lambda_1\lambda_2 \geq \alpha$, where $0 \leq \alpha < 1$. α is a user-input parameter and indicates the required relative performance. When the condition in Step 2 is fulfilled, the Adaptive approach switches to the Filter-Order approach while continuing to perform the above two steps. The Adaptive approach switches back to Sync-Filter when the test condition in Step 2 is FALSE. For example, when α equals 0.9 and the condition in Step 2 is TRUE, a relative improvement of at least $\frac{\alpha}{1+\alpha}$ [3] or 0.47 is achieved when using the Filter-Order approach.

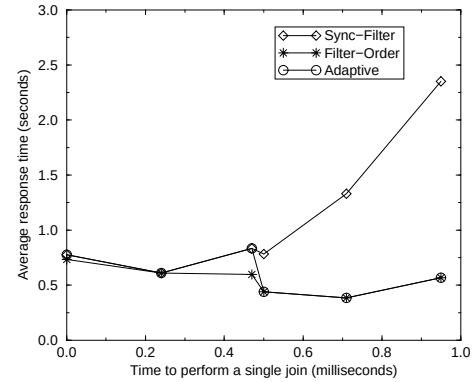
[2]The details of the equations's derivation is presented in [5].

5 Performance Study

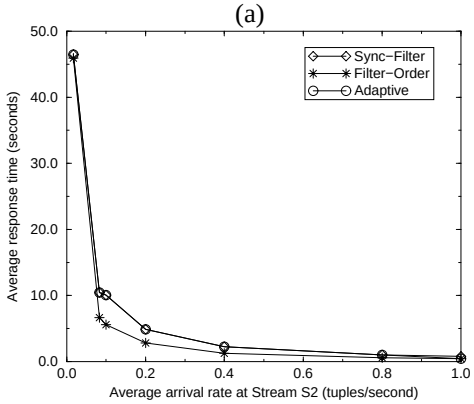
The experiments are performed on a prototype stream query processor, Nile, and uses a hash-based implementation of the W-join [6]. The join buffers are structured as hash tables that have the join attribute as the hash key. We have implemented the proposed algorithms in Sections 2, 3, and 4. Our measure of performance is the *average response time per input tuple*, which is the average time to completely process an input tuple by W-join. This time includes the waiting time, the processing time, and the time to produce an output tuple (if any). We perform our experiments on synthetic data streams, where each stream consists of a sequence of integers. In the experiments, the inter-arrival time between two consecutive tuples of an input data stream follows the Exponential distribution with mean $\frac{1}{\lambda}$. All the experiments are run on an Intel Pentium 4 CPU 2.4 GHz with 512 MB RAM running Windows XP.

Varying the Number of Concurrent Queries. In this experiment, we study the performance of the proposed approaches as we vary the number of concurrent queries. Our workload is a set of concurrent W-join queries over two data streams, S_1 and S_2. We measure the time to process a single W-join operation per query (parameter c in Section 3) as we increase the number of concurrent queries. Since c is directly proportional to the number of concurrent queries in our workload, we vary the value of c by varying the number of concurrent queries. We use a window of size one minute. The average stream arrival rate in S_1 (the slow stream) and S_2 (the fast stream) are 1 tuple/second and 10 tuples/second, respectively. We set α of the Adaptive approach to 0.3 (i.e., we would like to switch to Filter-Order if the relative improvement is greater or equal to $\frac{0.3}{1+0.3}$ or $\approx 25\%$). We collected the average response time of the input tuples during the lifetime of the experiment (20 minutes for each run). Figure 4 (a) gives the average response time when increasing c from 1 microsecond to 1 millisecond. Y-axis is the average response time per input tuple. With all processing times, Sync-Filter has the worst average response time. At large processing times, the difference between Sync-Filter and Filter-Order is significant and the difference gets smaller at small processing times. This can be interpreted

[3]The term is obtained by substituting $c|w|\lambda_1\lambda_2$ in Equation 1 by α.



(a)



(b)

Figure 4. The average response time while (a) varying the number of concurrent queries, (b) varying the input rate of S_1 (the slow stream)

as follows: Using Sync-Filter while increasing the processing time per join tuple, leads to excessive delays of tuples in the fast stream (i.e., S_2). This is the case as new tuples from S_2 must wait for a new tuple from the slow stream (i.e., S_1) to proceed in W-join. On the other hand, Filter-Order shows small or no variations in the average response time as we increase the processing time. This is mainly a result of overlapping the processing of tuples from S_2 while waiting for new tuples from S_1. The Adaptive approach behaves similar to Sync-Filter in our first three measurements since $c|w|\lambda_1\lambda_2 < \alpha$. At $c = 0.5$ milliseconds, $c|w|\lambda_1\lambda_2 = 0.3$ (i.e., $\geq \alpha$). Therefore, the Adaptive approach *switches* to the Filter-Order approach.

Changing Input Rate. In this experiment, we study the effect of the proposed approaches on the average response time while varying the arrival rate of the slow stream. We use a binary W-join with a window size of one minute as in the previous experiment. We fix the input rate of the fast stream (S_2) at 10 tuples/second and increase the input rate of the slow stream (S_1) from 0.01 to one tuple/second. As in the previous experiment, the Adaptive approach uses $\alpha = 0.3$. We fix the multiprogramming level such that $c \approx 0.5$ milliseconds. Figure 4 (b) gives the average response time. In all the proposed approaches, the average response time increases significantly (more than one minute) at small arrival rates of the slow stream. However, the increase in Sync-Filter is larger than that of Filter-Order for the same reasons, as explained in the previous experiment. Similar to the behavior in the previous experiment, the Adaptive approach switches between Sync-Filter and Filter-Order when the rate of the slow stream is one tuple/second. Having smaller α will shift the switching point to a small arrival rate of the slow stream.

6. Conclusion

In this paper, we studied the problem of providing ordered execution of window joins over data streams. We showed that the Sync-Filter approach that enforces ordered processing of input tuples to guarantee ordered output can result in increased response time. We then proposed the Filter-Order approach that applied the filter step of the window join followed by the buffering and ordering steps. In this way, the processing time of input tuples from one stream overlaps the waiting time to receive delayed tuples from the other stream. We studied both Sync-Filter and Filter-Order analytically and based on this analysis, we proposed the Adaptive approach that switches between Sync-Filter and Filter-Order to achieve a given performance goal. We showed through real implementation of the approaches on Nile the superiority of our proposed approaches over the Sync-Filter approach.

References

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, and et al. The Design of the Borealis Stream Processing Engine. In *Proc. of the CIDR Conf. Jan.*, 2005.

[2] S. Chandrasekaran, O. Cooper, A. Deshpande, and et al. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proc. of the CIDR Conf., Jan.*, 2003.

[3] D. J. DeWitt, J. F. Naughton, and D. A. Schneider. An evaluation of non-equijoin algorithms. In *Proc. of the VLDB Conf., Sep.*, 1991.

[4] L. Golab and M. T. Ozsu. Processing sliding window multi-joins in continuous queries over data streams. In *Proc. of the VLDB Conf., Sep.*, 2003.

[5] M. A. Hammad, W. G. Aref, and A. K. Elmagarmid. Optimizing In-Order Execution of Continuous Queries over Streamed Sensor Data. In *University of Calgary, Department of Computer Science TR#2004-766-31, Apr.*, 2005.

[6] M. A. Hammad, M. F. Mokbel, M. H. Ali, and et al. Nile: A query processing engine for data streams. In *Proc. of the ICDE Conf., Mar.*, 2004.

[7] U. Srivastava and J. Widom. Flexible time management in data stream systems. In *Proc. of the PODS Conf., Jun.*, 2004.

Querying Streaming Geospatial Image Data: The GeoStreams Project

Quinn Hart
CalSpace
University of California, Davis
Davis, CA, U.S.A.
qjhart@ucdavis.edu

Michael Gertz
Department of Computer Science
University of California, Davis
Davis, CA, U.S.A.
gertz@cs.ucdavis.edu

Abstract

Data products generated from remotely-sensed, geospatial imagery (RSI) used in emerging areas, such as global climatology, environmental monitoring, land use, and disaster management, require costly and time consuming efforts in processing the data. For the researcher, data is typically fully replicated using file-based approaches, then undergoes multiple processing steps, these steps often being duplicated at many sites. For the provider, data distribution is often tied directly to the data archiving task, focusing on simple, coarse grained offerings. Many RSI instruments transmit data in a continuous or semi-continuous stream, but current techniques in processing do not utilize the stream nature of the imagery. Recent research on continuous querying of data streams offer alternative processing approaches, but typically assume tuple style data objects, relying on traditional relational models as basis for query processing techniques and architectures. Complex types of stream objects, such as multidimensional data sets or raster image data, have not been considered. Our project, GeoStreams*, is a framework to process multiple continuous queries against streaming remotely-sensed geospatial image data. This paper introduces the basic features underlying the* GeoStreams *model. We describe some interesting aspects in processing streaming image data, including optimization and evaluation using specialized index structures.*

Remotely sensed data, in particular satellite imagery, play an important role in many environmental applications and models [10]. Simple, convenient access to remote sensing data has traditionally been a barrier to research and applications. The huge amounts of data generated by the Earth Observing System (EOS) platforms have precipitated a change in this scenario, and access to data products has become substantially easier. New EOS data archives offer fine examples of more transparent data access. However, access to this imagery still largely centers on choosing coarse grained, standard data products for specific regions and times. Applications that study changes in the environmental landscape require frequent, often continuous access to these data, and the temporal discontinuity in these access methods can force complicated preprocessing and synchronization steps between the data provider and the data user.

The sensors themselves, however, follow much more of a streaming paradigm. Data is acquired continuously and transmitted to receiving stations in a continuous manner. Outside the realm of image databases, there have been recent advancements in the more general field of data stream management systems (DSMSs), with new proposed query processing techniques [8] and research applications [1,3,4]. In such systems, data arrives in multiple, continuous, and time-varying data streams and does not take the form of persistent relations. There is clearly a potential benefit in taking techniques developed for DSMSs and adopting them to geospatial Remotely-Sensed Imagery (RSI) data.

The *GeoStreams* project investigates joining these two disciplines. In the *GeoStreams* architecture, researchers will explicitly consider the continuous temporal nature of RSI and formulate queries on these streams. Outputs of these queries continuously feed new RSI data to the researcher. These streams can be fed into applications to allow a continuous source of new input data from a single stream, or saved in more traditional RSI formats. As the functionality of the RSI DSMS increases, more aspects of the applications can be formulated into the queries themselves.

Requirements for the *GeoStreams* architecture include (1) identifying a query syntax that is natural for environmental application developers, as well as concise and unambiguous; (2) development of a core set of operations for RSI access; (3) query optimizations that allow a DSMS systems to tailor their execution plans to the currently active queries; and (4) execution plans that take advantage of the highly organized structure that is a trademark of RSI data. A wider range of interesting activities also include methodologies for continuous client-server data exchange, wire formats for streams of RSI, and investigating costly blocking operations on RSI data like image reprojections that can be

incorporated into a streaming system.

An Overview of the *GeoStreams* architecture is shown in Figure 1. Multiple users connect to the *GeoStreams* server and formulate queries to the system. The system is optimized for continuous queries on the input satellite stream of data. The queries are parsed and validated, then optimized. Optimization includes single and multi-query methods in this model, combining queries to minimize number and size of images that are created and maintained in the *GeoStreams* system. Minimizing the size of images reduces both memory usage and computational burden. Because of the way images can be shared between queries, however, computing query costs can be non-trivial. New queries affect the execution plan for the system, but these changes are made incrementally, because the execution is continuously working on the incoming RSI stream. This stream comes from a stream generation module that reinterprets the raw satellite data into a format more suitable for query processing.

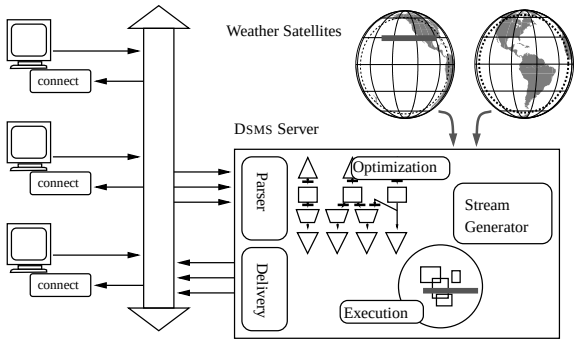


Figure 1. GeoStreams overview

Query execution is highly dependent on the structure of the incoming data. In our model, the RSI data is manipulated one row at a time. This matches the form of the satellite stream and is also convenient for satisfying multiple queries. Query execution ends with operators to return the data to the clients, which require persistent or synchronous connections on both the server and the client.

Our first RSI stream is continuous weather imagery from the National Oceanic and Atmospheric Administration (NOAA) Geostationary Operational Environmental Satellite (GOES) [6]. All data from the GOES satellite is transferred via a format specific these instruments. This continuous data stream transmits at approximately 2.1 Mb/sec. It has two instruments, the Imager and Sounder, which have 5 and 19 spectral channels respectively. GOES scans various sections of the Earth's surface about once every 15-30 minutes in spatial frames. A single frame varies in size from about 100MB to 400MB, depending on the region scanned. The ground resolution of the pixels varies between spectral channels. Data are basically delivered in a line by line manner, as GOES scans the hemisphere from North to South.

Images and image manipulation are based on image algebra [11], which is a rigorous and compact method for describing images, image transformations, and analysis. Initially, the notation for image algebra can be confusing and is kept to a minimum in this paper, although some high points in the context of streaming queries are discussed below. Image algebra is a many-valued algebra that includes *Point Sets*, *Value Sets* and *Images*.

Points Sets are defined in some topological space and correspond to the spatio-temporal location of the individual values in an image. Unlike many image definitions, the *GeoStreams* point sets typically include a temporal dimension. This allows for functional manipulations to be easily described in the algebra. Point sets are denoted with bold capital upright letter and points within a point set are denoted with lower case bold letters, i.e., $\mathbf{y} \in \mathbf{X}$.

Value sets encompass values associated with the points in the point set and are taken from a homogeneous set of operands, typically sets like integers, $\mathbb{Z}$, or real numbers, $\mathbb{R}$, although more complex, multi-valued sets can be defined. Value sets have the usual operations associated with their universal set.

Images are defined in general terms. The notation $\mathbb{F}^{\mathbf{X}}$ describes the set of all functions, $\{f \in \mathbb{F}^{\mathbf{X}} : f \text{ is a function from } \mathbf{X} \text{ to } \mathbb{F}\}$. An image is such a function that maps from a point set $\mathbf{X}$ to the value set $\mathbb{F}$. For an $\mathbb{F}$-valued image, $(\mathbf{a} : \mathbf{X} \to \mathbb{F})$, $\mathbb{F}$ is the possible *range* of the image $\mathbf{a}$ and $\mathbf{X}$ is the *domain* of $\mathbf{a}$.

Another convenient notation for an image $\mathbf{a} \in \mathbb{F}^{\mathbf{X}}$ is the *data structure representation*, $\mathbf{a} = \{(\mathbf{x}, \mathbf{a}(\mathbf{x})) : \mathbf{x} \in \mathbf{X}\}$. Here the pair $(\mathbf{x}, \mathbf{a}(\mathbf{x}))$ is a *pixel* of the image. The first coordinate $\mathbf{x} \in \mathbf{X}$ is the *pixel location* and the second coordinate $\mathbf{a}(\mathbf{x}) \in \mathbb{F}$ is the *pixel value* at location $\mathbf{x}$.

Image Operations are the basic building blocks for queries to the *GeoStreams* system. These operations include functional operations, image restrictions to specific point sets, spatial transforms on images from one point set to another, and neighborhood operations where multiple pixels from an image are combined to a single value. Figure 2 shows examples of these basic operations.

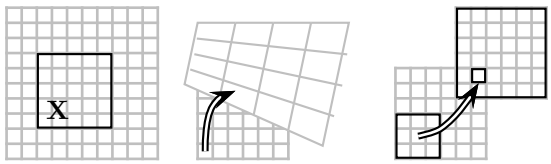
Defined operations on or among images include any operation that operates on the value set $\mathbb{F}$, which induces a natural operation on $\mathbb{F}$-valued images. For example, the addition of two images can be defined as $\mathbf{a} + \mathbf{b} = \{(x, a(x) + b(x)) : x \in \mathbf{X}\}$.

Image restrictions return images restricted to a given point set. In image algebra, if $\mathbf{a} \in \mathbb{F}^{\mathbf{X}}$, then the restriction, $\mathbf{a}|_{\mathbf{Z}}$ is defined as $\mathbf{a}|_{\mathbf{Z}} \equiv \mathbf{a} \cap \mathbf{Z} \times \mathbb{F} = \{(x, a(x)) : x \in \mathbf{Z}\}$. Some image models have formulated restrictions as selection operations, $\sigma_{x \in \mathbf{Z}}(a)$. Others formulate this as a spatial join, $a \ltimes_{a.x=Z.x} \mathbf{Z}$. Still others formulate restrictions functionally on an image data type.

Spatial restrictions are possibly the most important of all operations, and flexible methods for defining new point sets need to be included in query formulations. This is especially true in our model where point set restrictions define not only spatial, but also spatio-temporal limits on incoming data streams. Some point set manipulations are easy to represent, but many useful manipulations are more complex. Details of all potential point set manipulations have not been fully investigated, but since point sets are sets, relational algebra could be used as a framework for subset definitions.

Spatial transformations map an image from one point set to another. In general, for any function, f, between two point sets, $f : \mathbf{Y} \rightarrow \mathbf{X}$, and an image $\mathbf{a} \in \mathbb{F}^{\mathbf{X}}$; the spatial transform is defined as: $\mathbf{a} \circ f = \{(y, a(f(y))) : y \in \mathbf{Y}\}$.

Spatial transformations are used for magnification, rotation, and other general spatial manipulations. For geolocated imagery, reprojection of data into a new coordinate system is also a geometric transformation.



(a) $\mathbf{a}|_{\mathbf{X}}$ (b) $\mathbf{a} \circ f$ (c) $\mathbf{a} \bigoplus N$

Figure 2. Image Operations

Neighborhood operations allow for multiple pixels from a single image to be combined to create a single pixel value in a new image. Neighborhoods allow for aggregation functions like averaging, edge detection, speckle removal, and other operations. For example, $\mathbf{a} \bigoplus N$, indicates a local summation function where N represents an image template for the operation around local points.

Queries in the *GeoStreams* framework do not build on a variant of SQL syntax, but on something closer to the image algebra representation and on specialized interfaces. For example, consider a query for a normalized difference ratio on two satellite bands, a common type of index for environmental applications. We want to continuously receive this index for a particular region, reprojected to some convenient coordinate system, e.g. UTM. In image algebra, this could be represented as $((\mathbf{a}-\mathbf{b})/(\mathbf{a}+\mathbf{b})) \circ UTM|_X$, where $((\mathbf{a}-\mathbf{b})/(\mathbf{a}+\mathbf{b}))$ represents the index, $\circ UTM$ represents a function mapping from the satellite image to a new coordinate system, and $|_{\mathbf{X}}$ represents a restriction to some spatial extent.

This simple query demonstrates some of the problems in query formulation for a user. There must be methods to create complex spatial transform and restriction criteria, as mentioned before. These problems have been addressed in other research, and a number of workspace and workflow [2] models have been proposed, which are being investigated as a potential platform for describing general queries in *GeoStreams*.

However, in the near term, a query interface based on the OpenGIS Web Map Services (WMS) specification [5] is being developed. This simple interface does not allow for a sophisticated set of user queries, but it does investigate the most basic requirements of serving many spatial restrictions and geometric transformations to many clients. Basically, the interface allows users to specify specific data products, coordinate systems, and spatial extents. Temporal restrictions can also be identified. Queries like the one above could be specified, as long as the index itself is identified as a product in the server. The WMS specification further simplifies query formulation by standardizing and simplifying both spatial transforms and restrictions to a limited but well-defined subset. In general, the WMS specification limits queries to the form, $\mathbf{a} \bigoplus N \circ f|_X$, where $\mathbf{a}$, N, f, and X are specified in a simple standard way.

Query optimization attempts to limit the processing time and/or the amount of memory usage for the DSMS as a whole. In *GeoStreams*, query optimization is primarily concerned with two goals: query rewriting to limit the amount of work done in the system, and exploiting common subsets within the queries active against the image stream.

Consider the previous example, $((\mathbf{a} - \mathbf{b})/(\mathbf{a} + \mathbf{b})) \circ UTM|_{\mathbf{X}}$. This is a natural way to represent the query, but not an efficient computation method. As written, the index and spatial transform are performed on the entire domain of the image, most of which is discarded in the final restriction. Generally, moving restrictions to the front of the query improves efficiency. Restrictions can be reordered over spatial transforms by transforming the restriction point sets as well. For example, given $\mathbf{Y} = \{UTM(\mathbf{x}) : \mathbf{x} \in \mathbf{X}\}$, the above query can be rewritten as,

$$((\mathbf{a} - \mathbf{b})/(\mathbf{a} + \mathbf{b}))|_{\mathbf{Y}} \circ UTM \quad \text{or}$$
$$((\mathbf{a}|_{\mathbf{Y}} - \mathbf{b}|_{\mathbf{Y}})/(\mathbf{a}|_{\mathbf{Y}} + \mathbf{b}|_{\mathbf{Y}})) \circ UTM$$

Simple heuristics on queries, like those above, work well in the *GeoStreams* architecture, especially in the case were queries are limited in complexity, as they are with the WMS query interface. They also allow for some independence between the single- and multi-query optimization steps.

Once the individual queries are rewritten to optimize their individual execution, the queries are then optimized in a multi-query fashion as well. Optimization here centers around grouping similar query components into a single operation that works simultaneously for a group of queries. In DSMS research this has multiple conceptual definitions, including grouped filters [8] and query indexing [9]. Figure 3

shows a typical query index scheme for a spatial restriction operation, where rather than each query requiring its own restriction operator, a single restriction module has indexed the point sets of a number of active queries. For each continuous user query, a region is associated that describes the restriction for that query. For incoming RSI data, it is determined what data is relevant to what user queries and which queries can share incoming data.

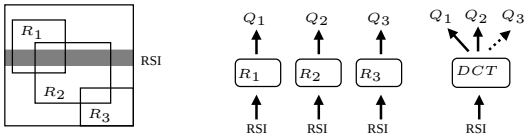


Figure 3. Restrictions on multiple queries

By developing modules for the basic image operations that can take as input a single RSI stream and distribute results to multiple output streams, the complete DSMS in the *GeoStreams* architecture is a number of these operators joined together for a complete system. This allows not only the pipelining of image data to operators to which the data is of interest, but it also facilitates the sharing of image data among queries that have non-disjoint query regions.

Query Execution is tied intrinsically to the query plan developed by the optimizer, and also by the organization of the incoming data stream. Modules are developed for each of the basic image operations, which satisfy multiple queries in a single operation. The modules are linked together for complete query execution. We have discussed how our RSI data stream comes in an ordered row-by-row arrangement. This organization plays an important role in how modules in the query plan are arranged.

Figure 3 shows an example module for processing multiple query image restrictions. For the restriction module, we have proposed the Dynamic Cascade Tree (DCT) [7], a space efficient structure to index query regions that are part of more complex queries against RSI data streams. The spatial trends inherent to most types of streaming RSI data is exploited to build a small index that is especially efficient when the incoming stream data are in close spatial proximity. Queries can be answered very quickly if the next data stream segment has the same result as the previous query and will incrementally update a new result set when the result is different. Based on the information provided by the DCT, incoming data can be pipelined to respective query operators, providing the basis for multiple-query processing models for streaming RSI data.

In Conclusion, we have described some of the basic concepts underlying the plans for a complete *GeoStreams* DSMS architecture for queries on streaming RSI data. We have already demonstrated the effectiveness of some of the basic modules within the system, for example, using the DCT as a method for indexing multiple query restrictions. Work is started on developing a preliminary system using the WMS specification as a basis for web-based access to the DSMS. There are a number of additional issues that can be investigated in this work, including determining the best wire formats for streaming query results, integrating mature publish/subscribe ideas into data delivery of RSI streams, allowing users to start queries in the past while maintaining a streaming paradigm and other issues. Our hope is that the test-bed developed here can be used to investigate these additional issues as well. The project is described at *http://db.cs.ucdavis.edu/geostreams*.

This work is supported by the NSF grant IIS-0326517.

References

[1] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, August 2003.

[2] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludaescher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *16th Intl. Conference on Scientific and Statistical Database Management (SSDBM)*, 2004.

[3] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford stream data manager. *IEEE Data Engineering Bulletin*, 26(1):19–26, March 2003.

[4] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, 2003.

[5] J. de La Beaujardiere. Web map service. OpenGIS Implementation OGC 04-024, Open Geospatial Consortium Inc., Aug 2004.

[6] *GOES I-M DataBook, Revision 1*. Space Systems-Loral, http://rsd.gsfc.nasa.gov/goes/text/goes.databook.html, 1996.

[7] Q. Hart and M. Gertz. Indexing query regions for streaming geospatial data. In *2nd Workshop on Spatio-temporal Database Management, STDBM'04*, 2004.

[8] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *Proc. of the 2002 ACM SIGMOD international conference on Management of data*, pages 49–60. ACM Press, 2002.

[9] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Trans. on Computers*, 51(10):1124–1140, 2002.

[10] A. Skidmore, editor. *Environmental Modeling with GIS and Remote Sensing*. Taylor & Francis, 2002.

[11] J. N. Wilson and G. X. Ritter. *Handbook of Computer Vision Algorithms in Image Algebra*. CRC Press, 2nd edition, 2001.

Session 6: Moving Objects

Clustering Moving Objects via Medoid Clusterings

Hans-Peter Kriegel, Martin Pfeifle
Institute for Computer Science
University of Munich, Germany
{*kriegel*, *pfeifle*}*@ifi.dbs.lmu.de*

Abstract

Modern geographic information systems do not only have to handle static information but also dynamically moving objects. Clustering algorithms for these moving objects provide new and helpful information, e.g. jam detection is possible by means of these algorithms. One of the main problems of these clustering algorithms is that only uncertain positional information of the moving objects is available. In this paper, we propose clustering approaches which take these uncertain positions into account. The uncertainty of the moving objects is modelled by spatial density functions which represent the likelihood that a certain object is located at a certain position. Based on sampling, we assign concrete positions to the objects. We then cluster such a sample set of objects by standard clustering algorithms. Repeating this procedure creates several sample clusterings. To each of these sample clusterings a ranking value is assigned which reflects its distance to the other sample clusterings. The clustering with the smallest ranking value is called the medoid clustering and can be regarded as the average clustering of all the sample clusterings. In a detailed experimental evaluation, we demonstrate the benefits of these medoid clusterings. We show that the medoid clustering is more suitable for clustering moving objects with fuzzy positions than arbitrary sample clusterings or clusterings based on the distance expectation values between the fuzzy positions of the moving objects.

1. Introduction

Clustering algorithms aim at grouping similar objects together, whereas dissimilar objects are assigned to different clusters. In the area of clustering moving objects, the similarity criterion is the distance between the objects. If we cluster objects moving on a spatial network [21], the distance between the objects on the network is used for clustering. If we aim at clustering objects which can freely move, the Euclidean distance between the objects can be used to measure the similarity, i.e. the closeness, between the objects [15].

Clustering moving objects has many different application ranges. For instance, clustering algorithms on a spatial network can be used for traffic jam detection and prediction. Clustering algorithms on freely moving objects can be used for weather forecasting [6], for detecting outliers, or for detecting animal migrations.

The problem of clustering moving objects is that often no accurate positional information is available. For instance, due to technical problems, the GPS system might not be able to pinpoint the exact positions of the moving objects. Another reason for uncertain positional information is that due to efficiency reasons it is not possible to update the exact position of the objects continuously. Clustering algorithms therefore have to deal with uncertain, outdated positional information.

In this paper, we propose an approach for clustering moving objects with uncertain positional information. We motivate a fuzzy modelling approach for describing moving objects and discuss several strategies which can be used for clustering these objects with standard clustering algorithms. After discussing the problems with the most straightforward approaches for clustering moving objects, we introduce an approach which uses the new concept of clustering rankings. Based on suitable distance functions between clusterings, we determine the medoid clustering from a set of sample clustgerings. The medoid clustering can be regarded as the clustering which represents all sample clusterings in the best possible way. Like ranking queries in databases, we can now return the sample clusterings according to their ranking values. The first returned clustering is the medoid clustering. In a give-me-more manner, the user can ask for more clusterings. Thus, the user gets a better picture of all clusterings which are possible when we cluster moving objects with uncertain positional information.

The remainder of this paper is organized as follows. In Section 2, we present the related work in the area of clustering moving objects. In Section 3, we introduce our fuzzy modelling approach which takes the uncertain positions of the moving objects into account. In Section 4, we present different approaches for clustering fuzzy moving objects. Our final approach relies on distance functions between clusterings. These distance functions are introduced in Section 5. In Section 6, we present our experimental evaluation, and conclude the paper in Section 7 with a short summary and a few remarks on future work.

2. Related Work

In this section, we will present the related work in the area of clustering moving objects. In Section 2.1, we first classify well-known clustering algorithms according to different categorization schemes. Then, in Section 2.2, we present the basic concepts of fuzzy clustering algorithms, and describe how the approach of this paper differs from the fuzzy clustering approaches presented in the literature. Finally, in Section 2.3, we present various approaches for clustering moving objects as presented in the literature.

2.1. Clustering Algorithms

Clustering algorithms can be classified along different, independent dimensions. One well-known dimension categorizes clustering methods according to the *result* they produce. Here, we can distinguish between *hierarchical* and *partitioning clustering* algorithms [12]. Partitioning algorithms construct a flat (single level) partition of a database D of n objects into a set of k clusters such that the objects in a cluster are more similar to each other than to objects in different clusters. Hierarchical algorithms decompose the database into several levels of nested partitionings (clusterings), represented for example by a dendrogram, i.e. a tree that iteratively splits D into smaller subsets until each subset consists of only one object. In such a hierarchy, each node of the tree represents a cluster of D.

Another dimension according to which we can classify clustering algorithms is from an *algorithmic* point of view. Here we can distinguish between *optimization based* or *distance based* algorithms and *density based* algorithms. Distance based methods use the distances between the objects directly in order to optimize a global criterion. In contrast, density based algorithms apply a local cluster criterion. Clusters are regarded as regions in the data space in which the objects are dense, and which are separated by regions of low object density (noise).

The following representatives of the 4 categories are used throughout our experimental evaluation:

	distance based	*density based*
partitioning	*k*-means[16]	DBSCAN[7]
hierarchical	Single-Link[12]	OPTICS[1]

2.2 Fuzzy Clustering

In real applications there is very often no sharp boundary between clusters so that fuzzy clustering is often better suited for the data. Membership degrees between zero and one are used in fuzzy clustering instead of crisp assignments of the data to clusters. The most prominent fuzzy clustering algorithm is the fuzzy c-means algorithm, a fuzzification of the partitioning clustering algorithm *k*-means. For more details about fuzzy clustering algorithms, we refer the reader to [11].

In contrast to fuzzy clustering algorithms where objects are assigned to different clusters, we cluster in this paper fuzzy object representations. The fuzzy spatial objects are assigned to exactly one cluster.

2.3. Clustering Moving Objects

In this section, we present some recent approaches from the literature dealing with the problem of clustering moving objects.

Yiu and Mamoulis [21] tackled the complex problem of clustering moving objects based on a spatial network. Here, the distance between the objects is defined by their shortest path distance over the network. Based on this distance measure they proposed variants of well-known clustering algorithms.

In [15], Han et al. applied micro-clustering [23] to moving objects. They propose techniques to keep the spatial extension of the moving micro-clusters small. To detect crucial events, e.g. split events, they measured the compactness of the moving micro-clusters by means of their bounding rectangles. If the size of the bounding rectangle exceeds a certain threshold, the micro-cluster is split. Different clustering algorithms can then be carried out on the moving micro-clusters instead of the individual points. In contrast to the experimental approach presented in [15], Har-Peled presented a more theoretical approach which also sacrifices quality in order to gain efficiency [10].

Clustering moving objects is not only interesting in its own, but can also beneficially be used for spatio-temporal selectivity estimation [22]. Zhang and Lin proposed a new clustering based spatio-temporal histogram, called CSTH, which allows to estimate the selectivity of predictive spatio-temporal queries accurately.

3. Modelling Fuzzy Moving Objects

In this section, we motivate the use of spatial density probability functions for describing the location of moving objects. This approach is quite similar to the approach presented by Behr and Güting [3] which use “degree or affinity” values to describe the probability that a certain point is included in a fuzzy spatial object.

Normally modern GPS systems can determine the exact position of moving objects very accurately. But, for instance, in the case of a war, this precision is reduced due to security aspects. Although, the system assigns a position $p(o, t) = (x, y)$ to each object o at a certain time t, we cannot be sure that the object o is located at the point (x, y) at time t. Nevertheless, it is very likely, that o is close to (x, y). This closeness can be modelled by assigning a 2-dimensional Gaussian density probability function o_{fuzzy} to the object (cf. Figure 1). The center of this probability function is at point (x, y) and the standard deviation σ is determined by the accuracy of the GPS system.

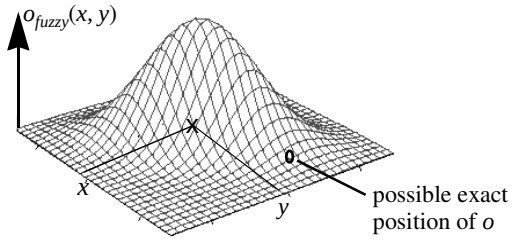


Figure 1. Fuzzy object representations for freely moving objects.

There exist other examples where it is beneficial to assign a 2-dimensional Gaussian distribution function o_{fuzzy} to an object o. For instance, animals or pedestrians which can freely move in the 2-dimensional space with a certain maximum velocity can be modelled by such a spatial density function. In order to cluster these objects effectively, it also seems reasonable to describe their positions by a 2-dimensional density probability function (cf. Figure 1). The center of this probability function is the last sent position of the object. The standard deviation σ depends on the maximum velocity of the object and the time passed since the object last sent its exact position.

If we assume that the object moves on a spatial network, e.g. cars moving on roads, we can assign a 1-dimensional Gaussian distribution function to the object (cf. Figure 2). The center of this probability function is a certain distance l away from the last sent position of the object. The value of l depends on the average velocity v_{avg} and the time t_{last} which passed since the object last sent its exact position. The standard deviation σ depends on the difference between the maximum and minimum assumed velocity, i.e. v_{max} - v_{min}, and on t_{last}.

Finally, if we, for instance, follow the approach presented in [15], we also cannot determine an exact position of an object o at clustering time. But as we know that the object is located within the bounding rectangle of the moving micro-cluster, we can assign to each object of the micro-cluster a density-probability function which assigns to each position a value $1/A_{box}$ where A_{box} denotes the area of the bounding rectangle. Note that we assign to each object of a micro-cluster the same density probability function (cf. Figure 3).

As shown in the above examples the position of a moving object cannot be described by only one single positional value. A better way, to describe a fuzzy moving object is to assign to each object a set of possible positions. To each of these positions, we assign a probability value which indicates the likelihood that this position is the exact one. Obviously, the sum of all these probability values is equal to 1.

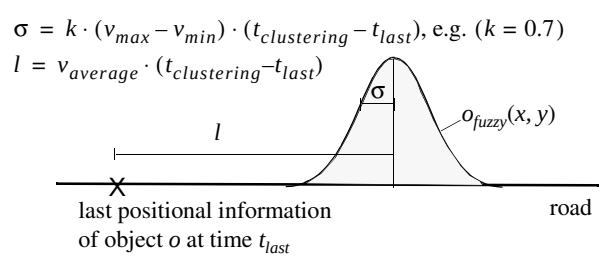


Figure 2. Fuzzy object representations for objects moving on a spatial network.

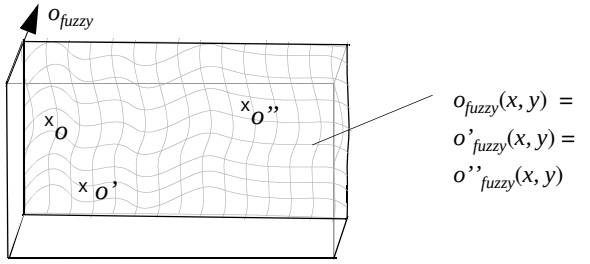


Figure 3. Fuzzy object representations for objects within a moving micro-cluster.

Definition 1 (*fuzzy moving object*)
Let $o \in DB$ be a moving object. To each moving object, we assign a fuzzy moving object function o_{fuzzy}: $IR^2 \rightarrow IR_0^+ \cup \infty$ for which the following condition holds:

$$\int_{IR^2} \int o_{fuzzy}(x, y)dxdy = 1$$

Figure 1, 2 and 3 show different fuzzy moving object functions o_{fuzzy} for two dimensional moving objects o. The functions o_{fuzzy} assigns a probability value $o_{fuzzy}(x, y) > 0$ to each possible position (x, y) of o. In the following, we use the term fuzzy moving object for both the object o and the corresponding function o_{fuzzy}.

4. Clustering Fuzzy Moving Objects

In this section, we present three different approaches which enable us to cluster fuzzy moving objects. All three approaches are based on sampling. In Section 4.1, we determine for each object o a concrete position based on the corresponding fuzzy moving object function. We use the resulting sample points as input parameters for the clustering algorithms. In Section 4.2, we carry out the clustering algorithms based on the distance expectation values between our fuzzy moving objects. The distance expectation values between our fuzzy moving objects are again computed by means of sampling. In Section 4.3, we determine a medoid clustering from a set of sample clusterings. The sample clusterings are computed as shown in Section 4.1. Then, we use suitable distance functions (cf. Section 5) between our sample clusterings to determine the corresponding medoid clustering.

4.1. Sampling

The most straightforward approach for clustering fuzzy moving objects is to assign to each moving object o an exact position according to its spatial density-probability function o_{fuzzy}. Figure 4 shows two possible positions p' and p'' of our fuzzy moving object o. Although position p' is much more

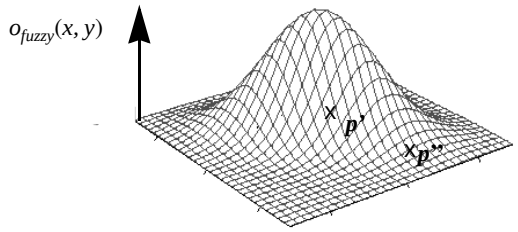


Figure 4. Two possible positions *p'* and *p'* of a moving object *o*.

likely, it is also possible that *o* is at position *p"*. For each fuzzy object o_{fuzzy}, we assume a position $p \in IR^2$. We can then apply any given clustering algorithm (cf. Section 2.1) to our fuzzy moving objects. The similarity between two fuzzy moving objects o_{fuzzy} and o'_{fuzzy} is then determined by an application dependent distance function, e.g. the Euclidean distance or the network distance, between the assumed positions *p* and *p'*. Based on this simple similarity measure between two fuzzy objects, we can apply any standard clustering algorithm.

Note that the thereby created clustering heavily depends on what positions we assumed for our fuzzy moving objects. Figure 5, for instance, shows a density-based clustering [7] based on sample positions. The resulting sample clustering does not reflect the intuitive clustering. If we look at the figure, we would rather derive a clustering $Cl = \{\{o_1, o_2\}, \{o_3, o_4\}\}$ which groups o_1 and o_2 together and o_3 and o_4. On the other hand, the sample clustering groups o_2 and o_3 together and assigns the objects o_1 and o_4 to noise.

4.2. Distance Expectation Values

In this section, we introduce the distance expectation value between fuzzy moving objects. This similarity measure between fuzzy moving objects is based on distance functions which do not express the similarity between two fuzzy moving objects by a single numerical value. Instead, we propose to use *fuzzy distance functions,* where the similarity between two objects is expressed by means of a probability function which assigns a probability value to each possible distance value. Then, we carry out the clustering algorithms based on the expectation values of the fuzzy distance functions (cf. Figure 6).

Definition 2 (*fuzzy distance function*)
Let d: $O \times O \rightarrow IR_0^+$ be a distance function, and let $P(a \le d(o, o') \le b)$ denote the probability that $d(o, o')$ is between *a* and *b*. Then, a probability density function p_d: $O \times O \rightarrow (IR_0^+ \rightarrow IR_0^+ \cup \infty)$ is called a *fuzzy distance function* if the following condition holds:

$$P(a \le d(o, o') \le b) = \int_a^b p_d(o, o')(x)dx$$

If the distance $\tau = d(o,o')$ between two objects can exactly be determined, the fuzzy distance function p_d is equal to the dirac-delta-function δ, i.e. $p_d(o, o')(x) = \delta(x\text{-}\tau)$ [2]. Thus the traditional approach can be regarded as a special case of Definition 2.

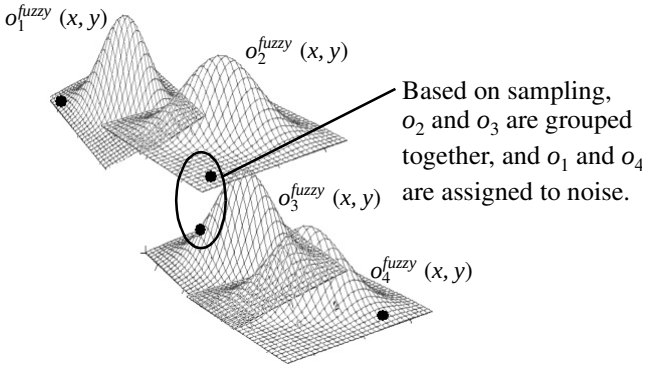


Figure 5. Clustering moving objects based on sampling.

As traditional algorithms can only handle distance functions which yield a unique distance value, we propose to extract the distance expectation value from these fuzzy distance functions. The distance expectation value E_d: $O \times O \rightarrow IR_0^+$ represents the fuzzy distance function in the best possible way by one single value $E_d(o, o') = \int_{-\infty}^{\infty} x \cdot p_d(o, o')(x)dx$ (cf. Figure 6b).

Although, this distance expectation value expresses the distance between two fuzzy moving objects in the best possible way, clustering based on these expectation values might be misleading. Figure 7, for instance, shows the computation of the core object condition for a fuzzy moving object *o*. Density based clustering algorithms like DBSCAN [7], for instance, decide for each object *o* whether *MinPts* objects are located within an ε–range of *o*. If this is the case, we call *o* a core object. Although, the object *o* in Figure 7a does not seem to be located in a very dense area, it is a core object according to the distance expectation approach. This holds as the distance expectation value between *o* and *MinPts*=4 other objects is smaller than ε. On the other hand, it is very unlikely that all *MinPts* objects are indeed located in $N_\varepsilon(o)$. Therefore, the probability that o is a core object is very small. In Figure 7b the reverse situation is sketched. Object *o* is located in a very dense area but there do not exist *MinPts* objects *o'* for which $E_d(o, o') \le \varepsilon$ holds. Therefore, *o* is no core object according to the distance expectation approach, although it is very likely that there exist *MinPts* elements *o'* for which $d(o, o') \le \varepsilon$ holds. To sum up, clustering based on the distance expectation values might be misleading.

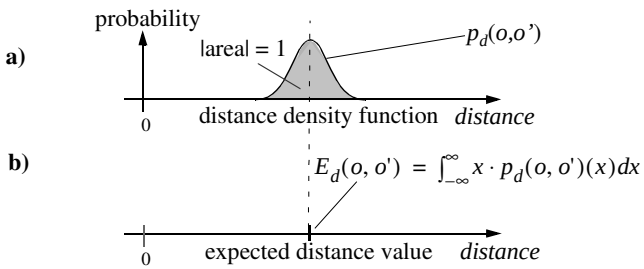
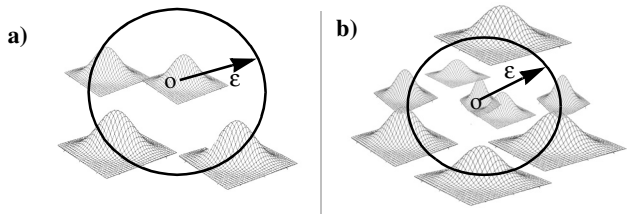


Figure 6. Fuzzy distance functions.



core object according to the approach based on the distance expectation values E_d.

no core object according to the approach based on the distance expectation values E_d.

Figure 7. Determination of the core object property based on the distance expectation value (*MinPts*=4).

4.3. Medoid Clustering

In this section, we propose a third approach which is based on the computation of sample clusterings. As shown in Section 4.1, we can compute a clustering of our moving objects based on sampling. Obviously, we can compute several of these sample clusterings. The question at issue is which is the most suitable of these sample clusterings. The idea of this paper is that we propose to compute the medoid clustering from these sample clusterings. In order to determine the average clustering, we need suitable distance functions between the sample clusterings (cf. Section 5). If we assume that we have functions which express the similarity between two clusterings, we can assign to each clustering *Cl* a clustering ranking value (cf. Definition 3) which sums up all the distances to all the other clusterings. The clustering with the smallest ranking value is called the medoid clustering (cf. Definition 4).

Definition 3 (*clustering ranking value*)
Let *DB* be a database of fuzzy objects, and let Cl_1, ..., Cl_s be *s* sample clusterings of *DB*. Furthermore, let *d* be a distance function between clusterings. Then, we assign to each clustering Cl_i a clustering ranking value R_i:

$$R_i = \sum_{\substack{j=1 \\ j \neq i}}^{s} d(Cl_i, Cl_j)$$

Obviously, the clustering having the smallest ranking value represents the set of clusterings in the best possible way. It is called the medoid clustering.

Definition 4 (*medoid clustering*)
Let *DB* be a set of fuzzy objects, and let Cl_1, ..., Cl_s be *s* sample clusterings of *DB*. Furthermore, let *d* be a distance function between clusterings. Then, Cl_i is called the medoid clustering if $\forall j \in 1 \ldots s: R_i \leq R_j$ holds.

Note, that in the example of Figure 5 it is very unlikely that the clustering $Cl = \{o_2, o_3\}$ is the medoid clustering, although it might be one sample clustering. If we compute, for instance, $s = 5$ clusterings, we might once get the above clustering, once we would assign all objects to noise and three times the sample clusterings are identical to the intuitive clustering $Cl = \{\{o_1, o_2\}, \{o_3, o_4\}\}$. Suitable metric distance functions between clusterings (cf. Section 5) would detect that the medoid clustering corresponds to the intuitive clustering $Cl = \{\{o_1, o_2\}, \{o_3, o_4\}\}$.

Similarly, if we look at the example presented in Figure 7, our medoid clustering approach seems to be more suitable than the approach based on the distance expectation values. Although in Figure 7a it might be possible that one sample clustering would decide that *o* is a core object, the majority of the samplings would decide that *o* is no core object. Therefore, it is very likely that the resulting medoid clustering would classify *o* correctly, i.e. assign it to noise. Similar, in Figure 7b it is very likely that our medoid clustering approach would decide that *o* is a core object, and, again, would classify the object correctly.

In the following, we will present an approach which helps us to compute the medoid clustering efficiently, if we assume that several slave computers are available.

4.3.1. Parallelization. If we assume that *L* different slave computers are available, we can easily parallelize the computation of the *s* sample clusterings. Obviously, each slave has at most $\lceil s/L \rceil$ clusterings to compute. Each slave can independently compute its clusterings and send the final results to all the other slaves. So all slaves have the final *s* clusterings before the computation of the medoid clustering based on the *s* sample clusterings starts.

As the computation of the distance measures between the clusterings can be very time consuming, we propose an approach which parallelizes the execution of the $s \cdot (s-1)/2$ distance computations between our *s* sample clusterings.

The idea is that a master triggers the computation of the clustering distances which are then carried out by the available slaves. Thus, one of the primary goals is that all slaves have an equal workload. To achieve that, the master keeps an $s \times s$ matrix *M* which indicates which distance computations between clusterings have already taken place. Furthermore, the master maintains an ordered list of the clusterings. The clusterings are ordered ascendingly according to their current clustering ranking values. Initially, all ranking values are set to zero. If a slave has computed a distance between two clusterings Cl_i and Cl_j, the master updates the corresponding ranking values R_i and R_j of these two clusterings and reorganizes the sorted list of clusterings. Furthermore, the master indicates in the matrix that the distance between Cl_i and Cl_j has been computed.

After initializing the matrix and the sorted list of clusterings, the master continuously checks whether there exist a slave *S* which is out of work. If this is the case, the master takes the first clustering Cl_{first} from the sorted list and checks by means of the matrix *M* whether all distances between Cl_{first} and the other *s*-1 clusterings have already been computed. If there is still one distance computation missing, the master asks the slave *S* to carry out this distance computation. If we

assume that the distance d between the clusterings is a metric, i.e. $\forall Cl,Cl':(d(Cl,Cl') \geq 0)$ holds, the algorithm terminates if all s-1 distance computations of Cl_{first} have already been computed. Then, the master knows that Cl_{first} is the searched medoid clustering without any further distance computations. Note that the ranking value of all the other clusterings can only increase but never decrease if we carry out further distance computations. Obviously, if the user is not only interested in the clustering having the smallest ranking value, the master continues with the above described ranking process.

The approach presented in this section is applicable to arbitrary distance functions between clusterings. In the following section, we introduce concrete distance functions between clusterings which are used throughout our experimental evaluation.

5. Similarity Measures between Clusterings

In the literature there exist some approaches for comparing partitioning [5, 17] and hierarchical [9] clusterings to each other. All of these approaches do not take noise objects into consideration which naturally occur when using density-based clustering algorithms such as DBSCAN [7] or OPTICS [1]. In [13] similarity measures are introduced which are suitable for generally measuring the similarity between partitioning and hierarchical clusterings even if noise is considered. In this section, we adapt these measures to our needs. We introduce distance functions between clusterings which can be used for computing medoid clusterings from sample clusterings. Based on the similarity measures for clusterings, we introduce quality measures which allow us to compare fuzzy clustering approaches to reference clusterings. In our experimental evaluation, we use these quality measures to compare the approaches presented in Section 4 to a reference clustering which is computed based on the exact positions of the moving objects[1].

Let us first introduce some basic terms necessary for describing clusterings. The following definitions are rather generic and can be applied to both reference clusterings and approximated fuzzy clusterings.

Definition 5 (*cluster*)
A *cluster* C is a non-empty subset of objects from a database DB, i.e. $C \subseteq DB$ and $C \neq \emptyset$.

Definition 6 (*partitioning clustering*)
Let DB be a database of arbitrary objects. Furthermore, let C_1, ..., C_n be pairwise disjoint clusters of DB, i.e. $\forall\, i, j \in 1, ..., n$: $i \neq j \Rightarrow C_i \cap C_j = \emptyset$. Then, we call $CL_p = \{C_1, ..., C_n\}$ a *partitioning clustering* of DB.

Note that due to the handling of noise, we do not demand from a partitioning clustering $CL_p = \{C_1, ..., C_n\}$ that $C_1 \cup ... \cup C_n = DB$ holds. Each hierarchical clustering can be represented by a tree. Even the density-based hierarchical clustering algorithm OPTICS which computes a hierarchical clustering order can be transformed into a tree structure by means of suitable cluster recognition algorithms [1, 4, 20].

Definition 7 (*hierarchical clustering*)
Let DB be a database of arbitrary objects. A *hierarchical clustering* is a tree t_{root} where each subtree t represents a cluster C_t, i.e. $t = (C_t, (t_1, ..., t_n))$, and the n subtrees t_i of t represent non-overlapping subsets C_{t_i}, i.e. $\forall i, j \in 1, ..., n$: $i \neq j \Rightarrow C_{t_i} \cap C_{t_j} = \emptyset \wedge C_{t_1} \cup ... \cup C_{t_n} \subseteq C_t$. Furthermore, the root node t_{root} represents the complete database, i.e. $C_{t_{root}} = DB$.

Again, we do not demand from the n subtrees t_i of $t = (C_t, (t_1, ..., t_n))$ that $C_{t_1} \cup ... \cup C_{t_n} = C_t$ holds.

5.1. Similarity Measure for Clusters

As outlined in the last section, both partitioning and hierarchical clusterings consist of flat clusters. In order to compare flat clusters to each other we need a suitable distance measure between sets of objects. The similarity of two clusters depends on the number of identical objects contained in both clusters which is reflected by the *symmetric set difference*.

Definition 8 (*symmetric set difference*)
Let C_1 and C_2 be two clusters of a database DB. Then the *symmetric set difference* $d_\Delta: 2^{DB} \times 2^{DB} \rightarrow [0..1]$ and the *normalized symmetric set difference* $d_\Delta^{norm}: 2^{DB} \times 2^{DB} \rightarrow [0..1]$ are defined as follows:

$$d_\Delta(C_1, C_2) = |C_1 \cup C_2| - |C_1 \cap C_2| \quad and$$

$$d_\Delta^{norm}(C_1, C_2) = \frac{|C_1 \cup C_2| - |C_1 \cap C_2|}{|C_1 \cup C_2|}$$

Note that $(2^{DB}, d_\Delta)$ and $(2^{DB}, d_\Delta^{norm})$ are metric spaces.

5.2. Similarity Measure for Partitioning Clusterings

In this section, we will introduce a suitable distance measure between sets of clusters. Several approaches for comparing two sets S and T to each other exist in the literature. In [8] the authors survey the following distance functions: the *Hausdorff distance*, the *sum of minimal distances*, the *(fair-)surjection distance* and the *link distance*. All of these approaches rely on the possibility to match several elements in one set to just one element in the compared set which is questionable when comparing clusterings to each other.

A distance measure on sets of clusters that demonstrates to be suitable for defining similarity between two partitioning clusterings is based on the *minimal weight perfect matching* of sets. This well known graph problem can be applied here by constructing a complete bipartite graph $G = (Cl, Cl', E)$ between two clusterings Cl and Cl'. The weight of each edge $(C_i, C'_j) \in Cl \times Cl'$ in this graph G is defined by the distance $d_\Delta(C_i, C'_j)$ introduced in the last section between the two clusters $C_i \in Cl$ and $C'_j \in Cl'$. A perfect matching is a subset $M \subseteq Cl \times Cl'$ that connects each clus-

1. In order to follow the main idea of this paper, you do not have to understand all details presented in this section. Thus, you might continue reading with Section 6.

ter $C_i \in Cl$ to exactly one cluster $C'_j \in Cl'$ and vice versa. A minimal weight perfect matching is a matching with maximum cardinality and a minimum sum of weights of its edges. Since a perfect matching can only be found for sets of equal cardinality, it is necessary to introduce weights for unmatched clusters when defining a distance measure between clusterings. We propose to penalize each unmatched cluster by its cardinality. Thereby, large clusters which cannot be matched are penalized more than small clusters which is a desired property for an intuitive similarity measure between partitioning clusterings.

Definition 9 (*partitioning clustering distance* $d_{clustering}^{partitioning}$)
Let *DB* be a database. Let $Cl = \{C_1, ..., C_{|Cl|}\}$ and $Cl' = \{C'_1, ..., C'_{|Cl'|}\}$ be two clusterings. We assume w.l.o.g. $|Cl| \leq |Cl'|$. Let π be a mapping that assigns to all $C' \in Cl'$ a unique number $i \in \{1, ..., |Cl'|\}$, denoted by $\pi(Cl') = (C'_1, ..., C'_{|Cl'|})$. The family of all possible permutations of Cl' is called $\Pi(Cl')$. Then the *partitioning clustering distance* $d_{clustering}^{partitioning}: 2^{2^{DB}} \times 2^{2^{DB}} \rightarrow IR$ is defined as follows:

$$d_{clustering}^{partitioning}(Cl, Cl') = \min_{\pi \in \Pi(Cl')} \left(\sum_{i=1}^{|Cl|} d_{\Delta}(C_i, C'_{\pi(i)}) + \sum_{i=|Cl|+1}^{|Cl'|} |C'_{\pi(i)}| \right)$$

Let us note that the *partitioning clustering distance* is a specialization of the metric *netflow distance* [19]. The partitioning clustering distance $d_{clustering}^{partitioning}(Cl, Cl')$ can be computed in $O(max(|Cl|,|Cl'|)^3)$ time using the algorithm proposed in [18].

Based on Definition 9, we can define our final quality criterion which helps to assess the quality of partitioning fuzzy clusterings to reference clusterings. We compare the costs for transforming the fuzzy clustering Cl^{fuzzy} into a reference clustering Cl^{ref}, to the costs piling up when transforming Cl^{fuzzy} first into $\emptyset$, i.e. a clustering consisting of no clusters, and then transforming $\emptyset$ into Cl^{ref}.

Definition 10 (*fuzzy partitioning clustering quality* Q_{FPC})
Let Cl^{fuzzy} be a fuzzy partitioning clustering and Cl^{ref} be the corresponding reference clustering. Then, the fuzzy partitioning clustering quality $Q_{FPC}(Cl^{fuzzy}, Cl^{ref})$ is equal to 1 if $Cl^{ref} = Cl^{fuzzy}$, else it is defined as

$$1 - \frac{d_{clustering}^{partitioning}(Cl^{fuzzy}, Cl^{ref})}{d_{clustering}^{partitioning}(Cl^{fuzzy}, \emptyset) + d_{clustering}^{partitioning}(\emptyset, Cl^{ref})}$$

Note that our quality measure Q_{FPC} is between 0 and 1. If Cl^{fuzzy} and Cl^{ref} are identical, $Q_{FPC}(Cl^{fuzzy}, Cl^{ref}) = 1$ holds. On the other hand, if the clusterings are not identical and the clusters from Cl^{fuzzy} and Cl^{ref} have no objects in common, i.e. $\forall C_j^{ref} \in Cl^{ref}, \forall C_i^{fuzzy} \in Cl^{fuzzy}: C_j^{ref} \cap C_i^{fuzzy} = \emptyset$ holds, then $Q_{FPC}(Cl^{fuzzy}, Cl^{ref})$ is equal to 0.

5.3 Similarity Measure for Hierarchical Clusterings

In this section, we first present a similarity measure between hierarchical clusterings. Based on these distance functions, we then introduce a quality criterion suitable for measuring the quality of fuzzy hierarchical clusterings. As already outlined, a hierarchical clustering can be represented by a tree (cf. Definition 7). In order to define a meaningful quality measure for fuzzy hierarchical clusterings, we need a suitable distance measure for describing the similarity between two trees t and t'. Note that each node of the trees reflects a flat cluster, and the complete trees represent the entire hierarchical clusterings.

A common and successfully applied approach to measure the similarity between two trees is the degree-2 edit distance [24]. It minimizes the number of edit operations necessary to transform one tree into the other using three basic operations, namely the insertion and deletion of a tree node and the change of a node label.

Definition 11 (*cost of an edit sequence*)
An edit operation e is the insertion, deletion or relabeling of a node in a tree t. Each edit operation e is assigned a non-negative cost $c(e)$. The cost $c(S)$ of a sequence of edit operations $S = \langle e_1, \ldots, e_m \rangle$ is defined as the sum of the cost of each edit operation, i.e. $c(S) = c(e_1) + \ldots + c(e_m)$.

Definition 12 (*degree-2 edit distance*)
The degree-2 edit distance is based on degree-2 edit sequences which consist only of insertions and deletions of nodes n with $degree(n) \leq 2$, and of relabelings. Then, the degree-2 edit distance between two trees t and t', $ED_2(t, t')$, is the minimum cost of all degree-2 edit sequences that transform t into t' or vice versa: $ED_2(t, t') = min\{c(S) | S \text{ is a degree-2 edit sequence transforming } t \text{ into } t'\}$.

Our final distance measure between two hierachical clusterings is based on the degree-2 edit distance.

Definition 13 (*hierarchical clustering distance* $d_{clustering}^{hierarchical}$)
Let *DB* be a database. Let Cl and Cl' be two hierarchical clusterings represented by the trees t and t'. Then, the hierarchical clustering distance $d_{clustering}^{hierarchical}$ is defined by:

$$d_{clustering}^{hierarchical}(Cl, Cl') = ED_2(t, t')$$

It is important to note that the degree-2 edit distance is well defined. Two trees can always be transformed into each other using only degree-2 edit operations. This is true because it is possible to construct any tree using only degree-2 edit operations. As the same is true for the deletion of an entire tree, it is always possible to delete t completely and then build t' from scratch resulting in a distance value for this pair of trees. In [24] Zhang, Wang, and Shasha presented an algorithm which computes the degree-2 edit distance in $O(|t| \cdot |t'| \cdot D)$ time, where D denotes the maximum fanout of the trees, and $|t|$ and $|t'|$ denote the number of tree nodes.

We propose to set the cost $c(e)$ for each insert and delete operation e to 1. Furthermore, we propose to use the *normal-*

ized symmetric set difference d_Δ^{norm} as introduced in Definition 8 to weight the relabeling cost. Using the normalized version allows us to define a well-balanced trade-off between the relabeling cost and the other edit operations, i.e. the insert and delete operations.

Based on the described similarity measure between hierarchical clusterings, we can define a quality measure for evaluating fuzzy hierarchical clustering algorithms. We compare the costs for transforming a fuzzy hierarchical clustering Cl^{fuzzy} modelled by a tree t^{fuzzy} into a reference clustering Cl^{ref} modelled by a tree t^{ref}, to the costs piling up when transforming t^{fuzzy} first into an "empty" tree t^{nil}, which does not represent any hierarchical clustering, and then transforming t^{nil} into t^{ref}.

Definition 14 (*fuzzy hierarchical clustering quality* Q_{FHC}) Let t^{ref} be a tree representing a hierarchical reference clustering Cl^{ref}, and t^{nil} a tree consisting of no nodes at all, representing an empty clustering. Furthermore, let t^{fuzzy} be a tree representing a fuzzy hierarchical clustering Cl^{fuzzy}. Then, the fuzzy hierarchical clustering quality Q_{FHC} (Cl^{fuzzy}, Cl^{ref}) is equal to 1 if $Cl^{ref} = Cl^{fuzzy}$, else it is defined as:

$$1 - \frac{d_{clustering}^{hierarchical}(C^{fuzzy}, C^{ref})}{d_{clustering}^{hierarchical}(C^{fuzzy}, \varnothing) + d_{clustering}^{hierarchical}(\varnothing, C^{ref})}$$

As the *hierarchical clustering distance* $d_{clustering}^{hierarchical}$ is a metric [24], the fuzzy hierarchical clustering quality Q_{FHC} is between 0 and 1.

6. Evaluation

In this section, we present a detailed experimental evaluation which demonstrates the characteristics and benefits of our new approach.

6.1. Settings

As test data sets for the effectiveness evaluation we used 1.000 2-dimensional points arbitrarily distributed in a data space [0..1] x [0..1]. For the efficiency evaluation, we used 10.000 of these points. The points moved at each timetick with an arbitrary velocity $v \in [0..v_{max}]$ in an arbitrary direction. Figure 8 shows that the higher the value of v_{max} is, the more uncertain is the position of the object after one timetick. Each position within the circular uncertainty area of the object is equally likely. As parameter for the experiments we used the radius r_U of the uncertainty area U.

In order to evaluate the quality of the various algorithms, we arbitrarily distributed the points in the data space. The reference clustering, was created by letting the points move as described above. A sample clustering was created by choosing one point arbitrarily from the uncertainty area of the object. From the resulting s sample clusterings we computed the medoid clustering by using the distance function of Definition 9 and 13 between clusterings. For the fuzzy clustering based on the distance expectation values, we used also the

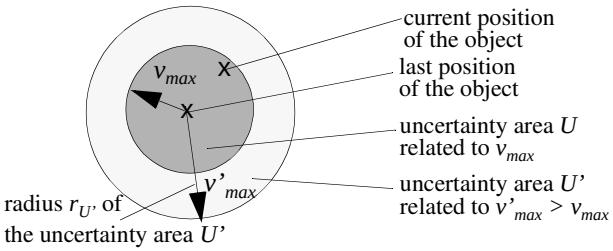


Figure 8. Test data set.

sample positions in the uncertainty areas. The distance between two moving objects is then equal to the average distance between their sample points.

The qualities of the fuzzy clusterings w.r.t. the exact clusterings were measured by the quality criterions introduced in Section 5. For DBSCAN [7] and for k-means [16], we used the one introduced in Definition 10, and for OPTICS [1] and for Single-Link [12], we used the one introduced in Definition 14.

If not otherwise stated, we used a sample rate s=10 throughout our experiments. For all clustering algorithms, we used a parameter setting which created a clustering according to intuition. For DBSCAN, for instance, we used a parameter setting so that we approximately detected 20 clusters containing 80% of all objects.

All clustering algorithms, the used quality measures, and the heuristic to accelerate the computation of the reference clustering were implemented in Java 1.4. The experiments were run on a Windows laptop with a 730 MHz processor and 512 MB main memory.

6.2. Experiments

6.2.1. Sample-Clusterings. In a first set of experiments, we investigated the maximum and minimum quality resulting from sampling w.r.t. the reference clustering. We compared these quality values to the quality achieved by the medoid clustering. Figure 9 shows clearly, that for all clustering algorithms the quality decreases with an increasing uncertainty area. Furthermore, we can see that there exist quite noticeable quality differences between the best and the worst sample clustering. This is especially true for interesting uncertainty values U which are neither too small nor too large. If the uncertainty area is too large, the quality is around zero for all sample clusterings, which means that the sample clusterings and the reference clustering are quite different from each other. On the other hand, if the uncertainty area is very small, all sample clusterings are almost identical to the reference clustering resulting in high quality values. Furthermore, the figure shows that the quality of the medoid clustering is somewhere in between the best and the worst sample clustering, and often quite close to the best sample clustering. Obviously, using the medoid clustering instead of an arbitrary sample

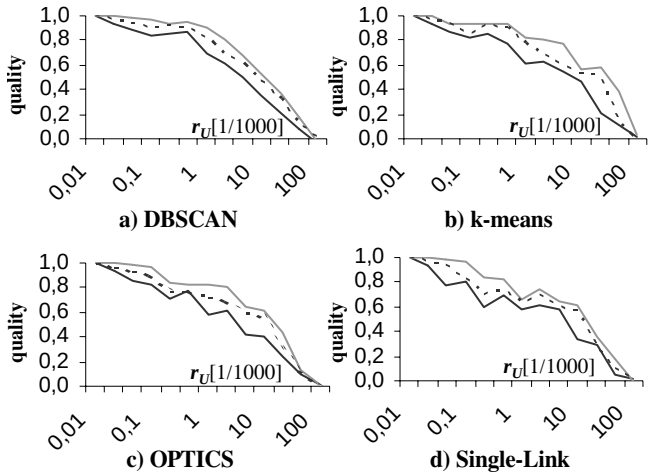


Figure 9. Sample Clusterings.

clustering reduces the probability that the determined clustering is very dissimilar to the reference clustering. Furthermore, let us note that Figure 9 also indirectly demonstrates the suitability of the distance functions and quality measures presented in Section 5.

As the partitioning density based clustering paradigm seems to be the most important and adequate clustering approach for moving objects [21], we concentrate in the following on the flat density-based clustering algorithm DBSCAN.

Figure 10 shows that the quality of the medoid clustering increases with increasing sampling rate *s*. This holds especially for small values of *s*. For values of *s* higher than 10 the increase of the quality is only marginal indicating that rather high values of *s* are not necessary to produce good clustering results. Furthermore, we can see that the quality of the worst sample clustering decreases with increasing sample rate *s*. Likewise, the quality of the best sample clustering increases. Obviously, the higher the sample rate is, the more likely it is that we generate a clustering which has a very small or a very high distance to the reference clustering. For the other clustering algorithms we made basically the same observations.

6.2.2. Distance Expectation Values. In Figure 11, the results of the clustering approaches based on the distance expectation value and the medoid clustering are compared to each other. Figure 11a shows clearly, that for high uncertainty values the quality achieved by the medoid clustering approach is much higher than the quality achieved by a DB-

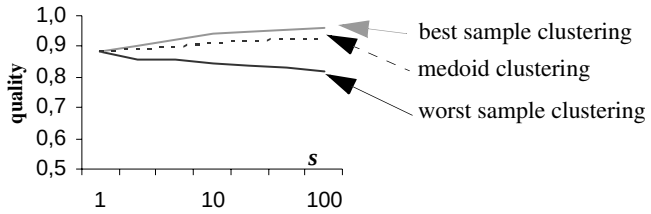


Figure 10. Medoid Clustering (DBSCAN) (r_U = 0.001).

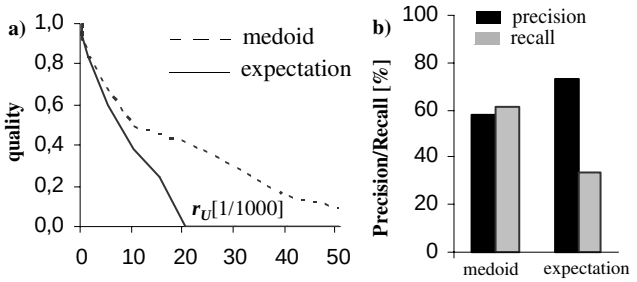


Figure 11. Distance Expectation Value (DBSCAN).
a) quality **b)** core-object classification (r_U=0.01)

SCAN run based on the distance expectation values. It is noteworthy, that in this case often the worst sample clustering achieved higher quality values than the distance expectation approach. The explanation for the bad performance of the distance expectation approach can be found in Figure 11b. Although the precision of the detected core objects is very high, the recall is very low, i.e. the approach fails to detect many core objects. Thus we have very often the situation depicted in Figure 7b. Let us note that for small uncertainty values the difference between the two approaches is less significant.

6.2.3. Other Comparison Partners. In [14] a density-based approach for clustering multi-represented objects was proposed which is based on DBSCAN. The authors propose for sparse data sets, the union-method which assumes that an object is a core object if *MinPts* objects are found within the union of all ε-neighborhoods of all representations. Furthermore, the intersection method was introduced where an object is a core-object, if it is a core object in each representation. We used these two approaches as comparison partners where a representative corresponds to a sample value. Figure 12 shows again that our medoid clustering approach outperforms the union and intersection method by far.

6.2.4. Efficiency. In a last set of experiments, we investigated the efficiency of our approaches. In all tests we did not use any index structure and all data was kept in main memory. Figure 13 shows clearly that if only one slave is available the single sampling approach is by far the most efficient approach. Obviously, the distance expectation approach is much slower due to the much more expensive distance computation between two objects. Note that the runtimes of the union/intersection approach are similar to the ones of the expectation approach. When using only one slave, the medoid

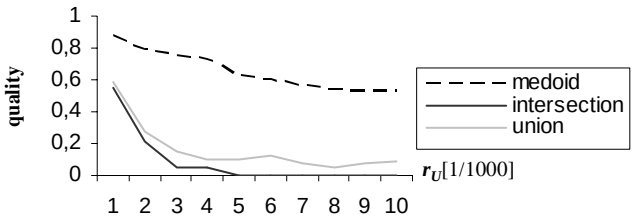


Figure 12. Union / Intersection Approach (DBSCAN).

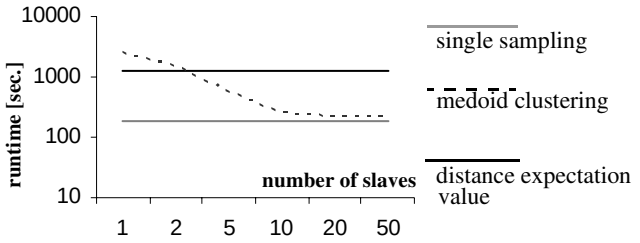


Figure 13. Efficiency Evaluation (DBSCAN). sampling rate s = 10, uncertainty radius $r_U = 0.001$

approach is even slower than the distance expectation approach because we have to determine the medoid clustering from the sample clusterings. The more slave computers are available, the more benefits our medoid approach. If *s* (=sample rate) slave computers are available, we can carry out a sample clustering on each slave. Therefore, we have an almost linear speed-up until *s* slaves are used. For a higher number of slaves, we can only parallelize the computation of the medoid clusterings from the sample clusterings, but not the generation of the sample clusterings. Therefore, we suggest to use *s* slaves for the computation of the medoid clustering.

In all our tests, we noticed that the heuristic introduced in Section 4.3.1 saves on average 12% of all distance computations between two clusterings. The ratio between the runtimes needed for the determination of the sample clusterings and the runtimes needed for the determination of the medoid clustering from these sample clusterings depends on the ratio of objects to be clustered and on the detected number of clusters. If we detect only a small number of clusters, the computation of the distances between two clusterings can be done efficiently when using the distance measures introduced in Section 5. On the other hand, distance computations between clusterings containing many clusters are rather expensive.

To sum up, the medoid approach is the method of choice for clustering fuzzy moving objects, especially if several slaves are available.

7. Conclusions

In this paper, we tackeled the complex problem of clustering moving object with uncertain positions. In order to do this effectively, we introduced the concept of *medoid clusterings*. We showed that these medoid clusterings are more suitable to cluster fuzzy moving objects than other approaches which are purely based on sampling or which are based on the distance expectation values between the fuzzy objects.

In our future work, we will show that density probability functions describing the positions of fuzzy moving objects can also beneficially be used in the context of location-based services.

References

[1] Ankerst M., Breunig M., Kriegel H.-P., Sander J.: *OPTICS: Ordering Points To Identify the Clustering Structure*. SIGMOD'99, pp. 49-60.

[2] Bracewell, R. *The Impulse Symbol*. Ch. 5 in The Fourier Transform and Its Applications, 3rd ed. : McGraw-Hill, 1999.

[3] Behr T., Güting R.H.: *Fuzzy Spatial Objects: An Algebra Implementation in SECONDO*. To appear at ICDE 2005.

[4] Brecheisen S., Kriegel H.-P., Kröger P., Pfeifle M.: *Visually Mining Through Cluster Hierarchies*. Proc. SIAM Int. Conf. on Data Mining (SDM'04), 2004, pp. 400-412.

[5] Banerjee A., Langford J.: *An Objective Evaluation Criterion for Clustering*. Proc. 10th ACM SIGKDD, 2004, pp. 515-520.

[6] Chudova D., Gaffney S., Mjolsness E., Smyth P.: *Translation-invariant mixture models for curve clustering*. Proc. 9th ACM SIGKDD, 2003.

[7] Ester M., Kriegel H.-P., Sander J., Xu X.: *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. KDD'96, pp. 226-231.

[8] Eiter, T., Mannila, H.: *Distance Measures for Point Sets and Their Computation*. Acta Informatica 34 (1997), pp. 103–133.

[9] Fowlkes E., Mallows C.: *A method for comparing two hierarchical clusterings*. Journal of American Statistical Association, 78, 1983, pp.553-569.

[10] Har-Peled S.: *Clustering Motion*. Discrete and Computational Geometry, 31(4):545-565, 2003.

[11] Höppner F., Klawonn F., Kruse R., Runkler T: *Fuzzy Cluster Analysis*. Wiley (1999).

[12] Jain A. K., Murty M. N., Flynn P. J.: *Data Clustering: A Review*. ACM Computing Surveys, Vol. 31, No. 3, Sep. 1999, pp. 265-323.

[13] Kriegel H.-P., Pfeifle M.: *Measuring the Quality of Approximated Clusterings*. BTW 2005.

[14] Kriegel H.-P., Kailing K., Pryakin A., Schubert M.: *Clustering Multi-Represented Objects with Noise*. Proc. 8th PAKDD 20004, pp. 394-403.

[15] Li Y., Han J., Yang J.: *Clustering Moving Objects*. Proc. 10th ACM SIGKDD, 2004.

[16] McQueen J.: *Some Methods for Classification and Analysis of Multivariate Observation*. Proc. 5th Berkeley Symp. on Math. Statist. and Prob., Vol. 1, 1965.

[17] Meila M.: *Comparing Clusterings by the Variation of Information*. Proc. 16th Annual Conference on Computational Learning Theory (COLT'03), pp. 173-187.

[18] Munkres, J.: *Algorithms for the assignment and transportation problems*. Journal of the SIAM 6 (1957) 32–38.

[19] Ramon J., Bruynooghe M.: *A polynomial time computable metric between point sets*. Acta Informatica 37 (2001), pp. 765–780.

[20] Sander J., Qin X., Lu Z., Niu N., Kovarsky A.: *Automatic Extraction of Clusters from Hierarchical Clustering Representations*. Proc. 7th PAKDD, 2003, pp 75-87.

[21] Yiu M. L., N. Mamoulis N.: *Clustering Objects on a Spatial Network*. Proc. 23th ACM SIGMOD, 2004, pp. 443-454.

[22] Zhang Q., Lin X.: *Clustering Moving Objects for Spatio-Temporal Selectivity Estimation*. Proc 15th Australasian Database Conference (ADC), 2004 pp. 123-130.

[23] Zhang T., Ramakrishnan R., Livny M.: *BIRCH: An efficient data clustering method for very large databases*. Proc. 15th ACM SIGMOD, 1996, pp. 104-114.

[24] Zhang K., Wang J., Shasha D.: *On the editing distance between undirected acyclic graphs*. International Journal of Foundations of Computer Science, 7(1):43–57, 1996.

Detection and Tracking of Discrete Phenomena in Sensor-Network Databases *

M. H. Ali[1] Mohamed F. Mokbel[1] Walid G. Aref[1] Ibrahim Kamel[2]

[1]Department of Computer Science, Purdue University, West Lafayette, IN
{mhali,mokbel,aref}@cs.purdue.edu
[2]College of Information Systems, Zayed University, U.A.E.
Ibrahim.Kamel@zu.ac.ae

Abstract

This paper introduces a framework for Phenomena Detection and Tracking (PDT, for short) in sensor network databases. Examples of detectable phenomena include the propagation over time of a pollution cloud or an oil spill region. We provide a crisp definition of a phenomenon that takes into consideration both the strength and the time span of the phenomenon. We focus on discrete phenomena where sensor readings are drawn from a discrete set of values, e.g., item numbers or pollutant IDs, and we point out how our work can be extended to handle continuous phenomena. The challenge for the proposed PDT framework is to detect as much phenomena as possible, given the large number of sensors, the overall high arrival rates of sensor data, and the limited system resources. Our proposed PDT framework uses continuous SQL queries to detect and track phenomena. Execution of these continuous queries is performed in three phases; the joining phase, the candidate selection phase, and the grouping/output phase. The joining phase employs an in-memory multi-way join algorithm that produces a set of sensor pairs with similar readings. The candidate selection phase filters the output of the joining phase to select candidate join pairs, with enough strength and time span, as specified by the phenomenon definition. The grouping/output phase constructs the overall phenomenon from the candidate join pairs. We introduce two optimizations to increase the likelihood of phenomena detection while using less system resources. Experimental studies illustrate the performance gains of both the proposed PDT framework and the proposed optimizations.

* This work was supported in part by the National Science Foundation under Grants IIS-0093116, IIS-0209120, and 0010044-CCR.

1. Introduction

The wide spread of sensor network applications calls for new online query processing techniques to deal with the continuous arrival of sensor data. Examples of these applications include surveillance [25] and environmental monitoring [26]. Within a sensor network, each individual sensor sends a stream of data to a sensor network database. Although the individual readings of each sensor is useful by itself, the overall processing of the data in the sensor network database as one unit provides a global view of the underlying environment.

Recent research literature focuses on leveraging database and data stream management systems to handle the massive amount of received data from sensor networks, e.g., see [5, 8, 9, 10, 12, 17, 19, 30]. The main goal is to provide efficient query processing techniques for sensor data. In this paper, we focus on extending data stream management systems to support sensor network applications. In particular, we focus on *Phenomena Detection* and *Tracking*, (*PDT*, for short). We propose a framework that can be plugged into any data stream management system to provide an online and efficient phenomena detection and tracking.

As a first step towards *phenomena detection*, we propose a *crisp* definition of a *phenomenon*. Then, we simplify the definition by considering the discrete case of the phenomenon. The proposed definition relies on two main parameters; *strength* (α) and *time span* (w). A phenomenon is of strength α and time span w when it occurs α times in the last w time units. The main idea of our proposed *phenomena detection* and *tracking* framework (*PDT*) is to join different readings from various sensors using a *multi-way* join algorithm for data streams. The output of the multi-way join algorithm feeds a *connectivity* graph that takes into consideration both the *strength* and *time span* of the required phenomenon. Continuously maintaining the *connectivity* graph *tracks* the sensor network phenomena. Moreover, we furnish our proposed *PDT* framework with a *phenomenon-*

aware optimizer where the execution of the *PDT* framework is tuned based on the received feedback from the query result.

In general, the proposed *phenomena detection* and *tracking* (*PDT*) framework has three phases; the *joining* phase, the *candidate selection* phase, and the *grouping/output* phase. The *joining* phase takes the raw data from the sensor network as its input and produces as output a set of sensor reading pairs that have similar values. The output of the *joining* phase is input to the *candidate selection* phase. The *candidate selection* phase strictly enforces the phenomena definition by filtering the input to produce only sensor pairs with the specified strength (α) and the time span (w). Finally, the *grouping/output* phase constructs the overall phenomenon from the candidate join pairs produced by the candidate selection phase. Moreover, the *candidate selection* phase gives a feedback on the query result to the *joining* phase. Based on the query feedback, we introduce two *phenomenon-aware* optimizations that aim to tune the performance of the *PDT* framework.

All the proposed ideas and algorithms in this paper are implemented inside the *Nile* data stream management system [14]. *Nile* is a research prototype that is currently being developed at Purdue University. In general, the contributions of this paper can be summarized as follows:

1. We introduce a *crisp* definition of a phenomenon that takes into consideration both the strength and the time span of the phenomenon.
2. We propose an efficient technique for *phenomena detection* and *tracking* (*PDT*). The proposed technique adheres to the proposed phenomenon definition.
3. We propose two *phenomenon-aware* optimizations where the query result, i.e., the detected phenomenon tunes the execution of the *PDT* framework.
4. We provide , based on a real implementation inside a prototype data stream management system, an experimental evidence of the efficiency and performance gains of the *PDT* framework.

The rest of the paper is organized as follows: Section 2 introduces the *phenomenon* definition. The SQL queries that initiate the processing of the *PDT* framework are presented in Section 3. Section 4 introduces our proposed framework for *phenomena detection* and *tracking* (*PDT*). The *phenomenon-aware* optimization techniques are presented in Section 5. Experimental results that are based on a real implementation of the proposed *PDT* framework inside a data stream management system are presented in Section 6. Section 7 highlights related work. Finally, Section 8 concludes the paper.

2. Phenomena Definition and Applications

In this section, we introduce the definition of a phenomenon along with some applications that can benefit from our proposed definition.

Definition 1 *In a sensor network SN, a phenomenon P takes place only when a set of sensors $S \subset SN$ report similar reading values more than α times within a time window w.*

Two parameters control the *phenomenon* definition, the *strength* (α) and the *time span* (w). The *strength* of a phenomenon indicates that a certain phenomenon should occur at least α times to qualify as a phenomenon. (This measure is similar to the notion of support in mining association rules, e.g., see [3].) Reading a value less than α times is considered noise, e.g., impurities that affect the sensor readings. The time span w limits how far a sensor can be lagging in reporting a phenomenon. w can be viewed as a time-tolerant parameter, given the common delays in a sensor network. (This measure is similar to the notion of gaps in mining generalized sequential patterns [24].)

In this paper, we focus on discrete phenomena that are produced by sensors whose reading values are discrete. In this case, the notion of similarity among sensor readings reduces to equality. Several applications benefit from the detection of discrete phenomena. Examples of these applications include:

- Tracing pollutants in the environment, e.g., oil spills in the ocean, or gas leakage out of a container. To be considered a phenomenon, the sensor should report the pollutant ID at least α times per w time units.
- Reporting the excessive purchase of a certain item at different branches of a retail store in the same day. The purchase of an item is considered a phenomenon when the number of purchases exceeds α times in the last w time units, e.g., in the last day.
- Detecting computer worms that strike various computer sub-networks over a certain period of time. When at least α computers are infected within a certain time window w, a phenomenon is reported.

Our work can be extended to detect continuous phenomena where sensors read values from a continuous range, e.g., temperature or density values, through a pre-processing phase. The pre-processing phase quantizes the sensor readings into a discrete set of value based on a user-defined function. Handling continuous phenomena is beyond the scope of this paper.

In general, a phenomenon may move in space. For example, an oil spill may surf the ocean according to the movement of the wind. A phenomenon may appear, disappear, move, expand, or shrink as time proceeds. In addition, a

SELECT SN1.VALUE, SN1.ID, SN2.ID
FROM SN SN1, SN SN2
WHERE SN1.VALUE=SN2.VALUE
AND SN1.ID <> SN2.ID
AND <*other conditions* >
GROUP BY $SN1.VALUE, SN1.ID, SN2.ID$
HAVING COUNT $(*) >= \alpha$
WINDOW W

Figure 1. PDT SQL queries

phenomenon may have spatial properties. For example, an oil spill is a contiguous portion of the ocean surface. In this case, the spatial phenomenon is termed a *"cloud"*.

3. PDT SQL-Queries

To support sensor network operations, we extend data stream management systems with an abstract data type (*ADT*), called *SensorNetwork-ADT*. *SensorNetwork-ADT* handles the extraction of sensor readings from the sensor network. Sensor readings are of the form $(ID, value, loc, ts)$, where ID is the identifier of the sensor that emitted the reading while $value$ and loc indicate the reading value and the location of that sensor at timestamp ts, respectively.

Figure 1 gives the general form of SQL queries that continuously detect phenomena in a sensor network database. Basically, the sensor network *SN* is joined with itself. Any sensor $S_i \in SN$ is eligible to join with any other sensor $S_j \in SN$, $(S_i \neq S_j)$, based on an equality join of *SN.value*. Based on the application semantics, the `where` clause specifies other conditions, e.g., the spatial and/or temporal clustering of the phenomenon. The phenomenon *strength* (α) is checked by grouping the query result by $(SN1.VALUE, SN1.ID, SN2.ID)$ and the *count* is calculated to report only sensors that join on the same value more than α times within window w. The phenomenon *time span* (w) is presented within the `window` clause.

Figure 2 gives an example of an SQL query that detects and tracks pollutants in the ocean, e.g., oil spills. OC represents a set of sensors distributed in the ocean. The sensor network OC is joined with itself based on the *liquid* value reported from each sensor. Only sensors that report a *liquid* value other than *"water"* are considered in the join. To reflect the *spatial* clustering of the detected pollutants, each sensor is restricted to join with other sensors that are at a maximum distance of ten meters. The *strength* (α) and *time span* (w) of the detected phenomena are set to five and one minute, respectively.

SELECT OC1.LIQUID, OC1.ID, OC2.ID
FROM OC OC1, OC OC2
WHERE OC1.LIQUID=OC2.LIQUID
AND OC1.ID <> OC2.ID
AND LIQUID<>"WATER"
AND DISTANCE(OC1.LOC,OC2.LOC)<= 10
GROUP BY $OC1.LIQUID, OC1.ID, OC2.ID$
HAVING COUNT $(*) >= 5$
WINDOW 1 minute

Figure 2. An example SQL query for pollution detection

4. PDT Query Processing

The process of *phenomena detection* and *tracking* (*PDT*) is initiated by issuing the SQL-query given in Figure 1. *PDT* query processing is divided into three phases as illustrated in Figure 3. The first phase, the *joining* phase, accepts the input tuples streamed out of the sensors and applies an in-memory *multi-way* join over the entire sensor network to detect sensors with the same value within a time frame of length w from each other. The second phase, the *candidate selection* phase, receives the joined sensor pairs and checks the sensors that qualify to be phenomena candidate members. Based on our definition of a phenomenon, the *candidate selection* phase checks the density of the phenomenon based on the user-specified strength (α) and time span (w). Sensors that join at least α times over a time-window w are reported to the *grouping/output phase*. The third phase, *the grouping/output phase*, groups the pairs of phenomena candidate members and investigates the application semantics to form and report the phenomena to the user.

Guided by the detected phenomena candidate members in the *candidate selection* phase, the processing is tuned to increase the likelihood of phenomena detection while using less resources. A phenomenon-aware feedback is provided to the *joining phase* to draw the attention to regions where phenomena tend to be active. For example, the input buffers that are associated with sensors contributing to phenomena are given higher priorities than those that do not contribute to any phenomena. Similarly, in the *joining phase*, the join probing sequence is tuned to favor the joins that affect the appearance or the disappearance of a phenomenon. The rest of this section is dedicated to the three phases of the proposed *PDT* framework. *phenomenon-aware* optimizations are presented in Section 5.

4.1. Phase I: Joining

Two alternative approaches exist for implementing the multi-way join operator for N streams: as a series of cas-

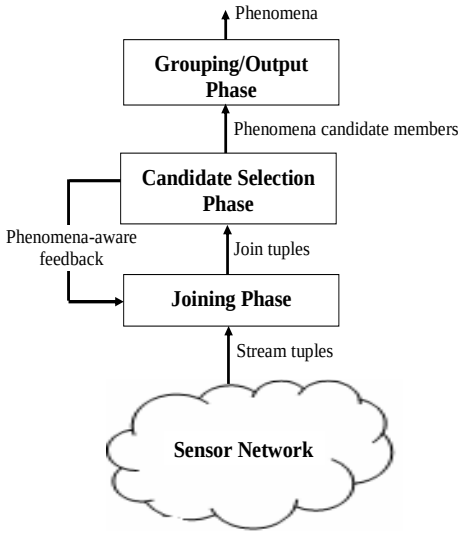


Figure 3. PDT query processing phases

caded $N-1$ binary join operators where only two streams are joined at a time, or as a single operator that takes N streams as its input. The MJoin operator [28] employs the second approach where it produces join results with a faster rate than the tree of binary joins. Thus, MJoin [28] is more suitable for data streaming applications. The main idea of MJoin is to maintain a hash table for each stream, i.e., sensor. Once a tuple arrives from one stream, it is inserted into the stream's corresponding hash table. Then, the incoming tuple probes the hash tables of other streams. Since a joined tuple is reported only if it appears in *ALL* streams, the MJoin algorithm stops probing hashing tables once the probed value is missing in one of the streams. To avoid unnecessary processing, the probing sequence is chosen based on the join selectivity among the streams.

To support *phenomena detection* and *tracking* (*PDT*), we employ three main modifications to the original MJoin algorithm [28]. These modifications are summarized as follows:

1. The modified multi-way join algorithm does not stop once the join value is missing in one of the streams. Instead, it continues to examine the remaining streams to produce partial results. Notice that a phenomenon need not span all the sensors in the sensor network.

2. The notions of positive and negative tuples [13] are utilized. A positive tuple is reported when a join occurs. A negative tuple is reported when one of the previously-reported join tuple components expires, i.e., becomes old enough to get outside of the most recent time-window w. The negative tuple is important to invalidate the candidate members of a phenomenon, if the sensors stop showing the same behavior over a time-window w.

3. The probe sequence and the stream sampling rate are guided by the detected phenomena to favor the probe sequence and the streams that participate in a phenomenon. This phenomenon-aware optimizations are discussed in detail in Section 5.

4.2. Phase II: Candidate Selection

The *joining* phase produces a tuple if the same reading is observed by two streams within the specified time-window. These two streams are considered phenomena candidate members if they persist to join with each other α times within the same time-window. The *candidate selection* phase employs a *connectivity graph* that is used to record the number of joins between each pair of sensors. Each sensor S_i is represented by a node in the *connectivity graph*. For any two sensors S_i and S_j, $(i \neq j)$, an edge $E(v,i,j)$ is added to the connectivity graph only if S_i and S_j are joined together at least once in the last w time units over value v. The weight of the Edge E_{ij} is the number of times that S_i and S_j are joined together in the last w time units, i.e., the strength of the phenomenon.

Figure 4 gives the processing of input pairs received from the *joining phase*. The input is either a positive or a negative tuple with the format $\pm(SN1.VALUE, SN1.ID, SN2.ID)$. The tuple represents the join value and two joining sensors. This tuple updates the weights of the edges in the connectivity graph. The weight of each edge is monitored. If the weight of an edge increases to reach α (i.e., $weight = \alpha$), a positive tuple is reported to denote the appearance of the candidate member $(SN1.VALUE, SN1.ID, SN2.ID)$. If the weight of an edge drops below α (i.e., $weight = \alpha - 1$), a negative tuple is reported to denote the disappearance of that candidate member.

Figure 4.2 gives an example of the connectivity graph for five sensors over a window of 5 time units. The connectivity graph starts from scratch and records each join tuple by increasing the weight of the edge between the two joining sensors. Edges that exceed α, which is set to four, are marked as bold lines to denote phenomenon candidate members. Notice that, as the window slides, the value 10 from sensor $S1$, that came at $t1$, will expire, and consequently the edge between $S1$ and $S3$ will drop below α generating a negative tuple to invalidate that candidate member.

4.3. Phase III: Grouping/Output

The *grouping/output* phase receives phenomena candidate members on the form of a tuple that consists of the IDs of the two joining sensors and the join value. Each tuple

INPUT: the join tuple (SN1.VALUE, SN1.ID, SN2.ID) OUTPUT: the phenomena candidate members

Upon receiving a positive tuple,

if CheckEdge(SN1.VALUE, SN1.ID, SN2.ID)
// if edge exists increase its weight
Edge(SN1.VALUE, SN1.ID, SN2.ID).weight++
// check the appearance of a candidate
if (Edge(SN1.VALUE, SN1.ID, SN2.ID).weight=α)
Output +(SN1.VALUE, SN1.ID, SN2.ID)
else
// create a new edge with weight=1
CreateEdge(SN1.VALUE, SN1.ID, SN2.ID)
Edge(SN1.VALUE, SN1.ID, SN2.ID).weight=1;
endif

Upon receiving a negative tuple,

// Decrease the weight of the edge by 1 and
Edge(SN1.VALUE, SN1.ID, SN2.ID).weight−−
// check the disappearance of a candidate
if (Edge(SN1.VALUE, SN1.ID, SN2.ID).weight=$\alpha - 1$)
Output −(SN1.VALUE, SN1.ID, SN2.ID)
// remove the edge if its weight becomes zero
if (Edge(SN1.VALUE, SN1.ID, SN2.ID).weight=0)
RemoveEdge(SN1.VALUE, SN1.ID, SN2.ID)
endif

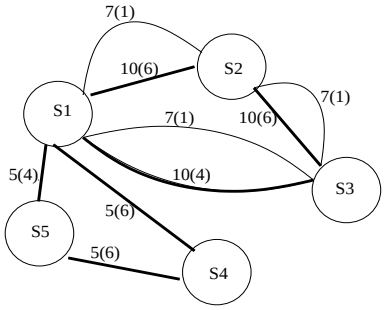
Figure 4. Pseudo code of the candidate selection phase

can be positive or negative to denote the appearance or disappearance of a candidate member. A positive/negative tuple indicates that the number of joins between the two sensors over the last w becomes above/below α. Upon receiving a positive tuple, based on the application semantics, the *grouping/output* phase may start a new phenomenon, add one sensor to an existing phenomenon, or merge two phenomena together. Similarly, upon receiving a negative tuple, the *grouping/output* phase may delete a phenomenon, remove a sensor from an existing phenomenon, or split one phenomenon into two separate phenomena.

In addition, the *grouping/output* phase has the flexibility to apply application-dependent semantics. For example, forcing a minimum number of sensors to form a phenomenon, determining the outer contour or the convex hull of the phenomena . Also, the application may consider the spatial clustering of sensors. The clustering or spread of the phenomenon implies whether there is a single source or multiple sources for the phenomenon. For example, multiple disconnected oil spills implies leakage out of more than one container. Application-dependent semantics can include the density of the phenomena as well. The density is measured by the ratio of the number of sensors reading

	t1	**t2**	**t3**	**t4**	**t5**
Sensor 1	**10**	**7**	**10**	**5**	**5**
Sensor 2	15	**10**	**7**	**10**	**10**
Sensor 3	19	**7**	3	**10**	**10**
Sensor 4	22	**5**	9	**5**	**5**
Sensor 5	18	**5**	**5**	4	23

(a) Sensors' input over 5 time instants



(b) The connectivity graph

Figure 5. An example of the connectivity graph

the same phenomenon to the number of sensors not reading that phenomenon in a specified region. All these issues are application-dependent and are addressed by the *grouping/output* phase.

5. Phenomenon-Aware Query Optimization

This section proposes two *phenomenon-aware* optimizations that aim to provide a scalable execution for the proposed *phenomena detection* and *tracking* (*PDT*) framework. These optimizations tune the processing towards tuples that contribute in producing the phenomena. The main idea is to utilize the processing of the *candidate selection* phase to provide feedback to the *joining* phase. The feedback contains information about the sensors that contribute to the currently tracked phenomena. The two phenomenon-aware optimizations are: (1) Controlling the sampling rate of each sensor, and (2) Choosing the join probing sequence.

5.1. Controlling the Sampling Rate of Sensors

The number of sensors in a sensor network can grow large. These sensors may be generating stream data with

high rates. They may show a bursty behavior as well. As a result of all of the above reasons, the query processing engine faces periods of heavy load. In these periods, it will not be possible to cope with every sensor reading. To overcome this problem, a sampler for each sensor is employed to control the input rate of its generated stream. Instead of a random sampler, a *phenomenon-aware* sampler is preferred to favor sensors that contribute heavily to the phenomena. The *phenomenon-aware* sampler gets feedback from the *candidate selection* phase about the phenomena that are currently monitored.

Notice that if the weight of an edge between two sensors is highly below or highly above α, it is less likely to provide new information. If it is highly below α, it is less likely to increase instantly to α and form a phenomenon candidate member. Similarly, if it is highly above α, it is less likely to decrease instantly below α and eliminate a candidate member. Sensors with edges that are close to α are considered strong candidates to form or eliminate a phenomenon. Hence, as given in Equation 1, the *edge strength (ES)* between two sensors i, j is inversely proportional to the absolute difference between the edge weight and α ($|Edge(v,i,j).weight - \alpha|$), where $Edge(v,i,j).weight$ is the weight of the edge between sensors i and j based on the value v (or zero if no edges at all). However, there may be more than one edge between the two sensors if they join over multiple values. To be conservative, the edge gets the maximum strength over all of these values and the *sensor strength (SS)* is considered to be the maximum strength over all of the edges connecting that sensor to its neighbors (Equation 2). To favor sensors that are involved in phenomena, the *sampling factor (SF_i)* of each sensor i is proportional to its strength as given in Equation 3.

Let R^* be the global desired sampling rate over all sensors, and let R_i be the sampling rate of each sensor. R^* is formed by adding the sensor rates R_i after they are being adjusted by SF_i (Equation 4). Equation 4 is rewritten again in the form of a summation in Equation 5. In Equation 6, we substitute for SF_j by Equation 3. From Equation 7, we can obtain SF_i given the rate of each sensor, the strength of each sensor and the desired rate R^*. These parameters are continuously updated as the streams are running. The stream rates ($R_i s$) and the desired rate (R^*) are updated periodically to avoid unnecessary fluctuations and bursty behaviors of sensors. The sensor's strength (SS_i) is updated with the arrival of each tuple to eagerly detect the new phenomena.

$$ES(i,j) = MAX_v\{\frac{1}{1+|Edge(v,i,j).weight - \alpha|}\} \quad (1)$$

$$SS_i = MAX_j(ES(i,j)) \quad (2)$$

$$\frac{SF_i}{SF_j} = \frac{SS_i}{SS_j} \quad (3)$$

$$SF_1 \cdot R_1 + SF_2 \cdot R_2 + \cdots + SF_n \cdot R_n = R^* \quad (4)$$

$$SF_i \cdot R_i + \sum_{j=1, j\neq i}^{n} SF_j \cdot R_j = R^* \quad (5)$$

$$SF_i \cdot R_i + \sum_{j=1, j\neq i}^{n} \frac{SF_i \cdot SS_j}{SS_i} \cdot R_j = R^* \quad (6)$$

$$SF_i \cdot (R_i + \frac{\sum_{j=1, j\neq i}^{n} SS_j \cdot R_j}{SS_i}) = R^* \quad (7)$$

Reducing the stream sampling rate of a sensor may lead to delaying the discovery or to entirely missing new phenomena because they will take a longer time to increase the edge weights between participating sensors till they reach to the desired α. A sensor needs to be persistent in producing the phenomenon and to increase its strength gradually till the phenomenon is discovered. The difference between the time at which the phenomenon is formed and the time at which it is reported is known as the response time. We trade the response time of discovering new phenomena for the sake of monitoring already existing phenomena efficiently. Otherwise, monitoring all sensors with the same quality would degrade the whole system's performance and may result in losing phenomena.

5.2. Choosing the Join Probing Sequence

Once a tuple arrives from one sensor, it is used to probe the hash tables of other sensors looking for matches. The sequence in which the tuple probes other hash tables affects the performance of the join operator. In the original MJoin [28] algorithm, the selectivity factors among the joins are taken into consideration. The least selective join is evaluated first. Consequently, the number of partial output tuples is reduced at early steps. In our context, choosing the join order based on the selectivity is not of great benefit where we are interested in partial results as well. Moreover, in large sensor networks, probing hundreds or thousands of sensors may be prohibitive, specially when the sensor joins with a very small subset of the sensors. Instead, we choose a probing sequence in which probing is based on the likelihood of producing a tuple that contributes to a phenomenon.

If a tuple arrives at sensor i, it probes the hash table of sensor j with probability P, where

$$P = \frac{1}{1+|Edge(v,i,j).weight - \alpha|} \quad (8)$$

Equation 8 adjusts the probing probability based on how close the edge between sensors i and j on value v to α. The

INPUT: a tuple from sensor S_i with value v
OUTPUT: the join probing sequence

for j=1 to NoOfSensors
begin
$P = min(BaseProb, \frac{1}{1+|Edge(v,i,j).weight-\alpha|})$
Generate a random variable U *between* $0, 1$
if $(U \leq P)$
ProbeSensor(j)
end

Figure 6. The join probing sequence

join probing sequence is evaluated as given in Figure 6. On the arrival of a tuple with value v from sensor i, a complete traversal over all sensors is performed. Because the probing operation is costly, we decide to either probe or skip sensor j based on the probability P. Notice that P should not go below a minimum $BaseProb$ to avoid the zero join probability among sensors with no edges in between. This policy reduces the number of joins dramatically and focuses on joins that contribute in phenomena. Eliminating the cost of unnecessary joins allows our technique to be scalable with respect to the number of sensors. Similar to the case of Section 5.1, a delay may be observed in detecting new phenomena. A sensor needs to strengthen the edge between itself and other sensors gradually to get a higher probability in the join operation.

6. Experiments

In this section, we conduct an experimental study of the proposed *phenomena detection* and *tracking PDT* technique. Three sets of experiments are conducted. The first set of experiments (Section 6.1) is concerned with the effect of the *PDT* parameters; the strength α and the time span w on the number of detected phenomena. The second (Section 6.2) and the third sets (Section 6.3) of experiments study the performance of the *PDT* optimization techniques with the change of the number of sensors and data arrival rates, respectively. For the last two sets of experiments, we compare the performance of the following four versions of the proposed *PDT* framework:

1. *Simple PDT*, where processing is not guided by the detected phenomena.
2. *PDT+1*, where the sensor's sampling rate is controlled based on the detected phenomena as discussed in Section 5.1.
3. *PDT+2*, where the join probing sequence is controlled based on the detected phenomena as discussed in Section 5.2.

α	w=5	w=10	w=15	w=20
3	1010	4350	7150	11150
4	121	498	815	1256
5	20	220	270	320
6	4	19	56	140
7	0	5	11	18
8	0	0	2	5

Table 1. The effect of application-dependent parameters

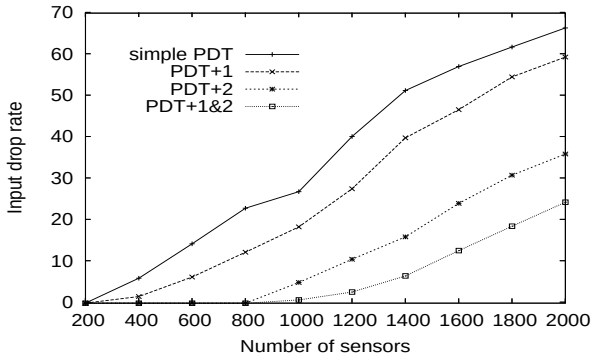
4. *PDT+1&2*, where both of the optimizations in (2) and (3) are applied together.

Various *PDT* techniques are compared with respect to three performance measures: (1) The *input drop rate* where some of the input sensor data tuples have to be dropped due to the scarcity of system resources. (2) The *response time*, which is measured by the difference between the time in which a phenomenon is reported by the system and the actual time in which it took place. (3) The *output loss rate* where some phenomena are lost as a result of losing some of the input tuples. A smart technique tries to minimize all these measures. rate.

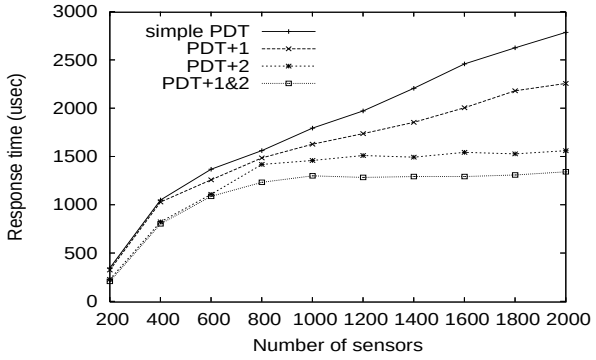
All the experiments are triggered by the execution of the continuous query given in Figure 2. A synthesized data set is used to simulate the sensor network readings. Unless mentioned otherwise, we maintain 1000 sensors uniformly distributed over a 100×100 meters rectangular space. Each sensor generates a stream of 10,000 tuples where the tuple values follow the zipfian distribution. The interarrival time of sensor data follows an exponential distribution with an average of one second. All the experiments in this section are based on a real implementation of the *PDT* framework inside the *Nile* data stream management system [14]. The Nile engine executes on a machine with Intel Pentium IV, CPU 2.4GHZ with 512MB RAM running Windows XP.

6.1. PDT parameters

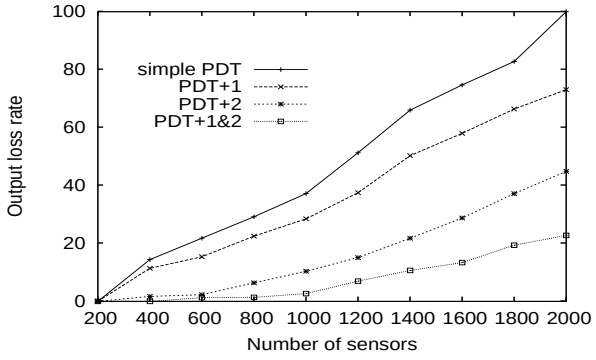
Table 1 gives the number of detected phenomena for various values of the strength (α) and time span (w). As given in the table, α and w have opposite effects. The increase in α results in less detected phenomena as the condition for detecting a phenomenon becomes more restrictive. On the other side, the increase in the time span w relaxes the condition for the detected phenomena. Thus, more phenomena can be detected. The largest number of detected phenomena is obtained for $w = 20$ and $\alpha = 3$. At this point, *PDT* tracks up to 11,150 different phenomena over the lifetime of the experiment.



(a) The input drop rate



(b) The response time



(c) The output loss rate

Figure 7. The effect of the number of sensors

6.2. The effect of the number of sensors

Figure 7 studies the scalability of the four variations of the *PDT* framework with respect to increasing the number of sensors from 200 to 2000. The *PDT* parameters α and w are set to 5 and 10 seconds, respectively. *PDT-1* decreases the input drop rate over the *simple PDT* because it reduces the sampling rate of sensors that do not contribute to any phenomenon and, hence, keeps the input buffers less occupied (Figure 7a). The response time of the *PDT+1* is less than that of the *simple PDT* because the size of the hash structures gets smaller after reducing the sampling rate of irrelevant sensors (Figure 7b). As a result of controlling the sampling rates, the output loss rate of the *PDT+1* is reduced (Figure 7c). Notice that as the number of sensors increases, more load is posed against the system and the difference in the output loss rate becomes significant (up to 25% for 2000 sensors). *PDT+2* reduces the response time (Figure 7b) because it favors the join over sensors that contribute to phenomena and leaves other joins to be performed with a lower probability. This behavior increases the processing time availability and reduces the input drop rate (Figure 7a). The controlled join probing sequence reduces the output loss rate through the efficient management of the time budget in useful joins (Figure 7c). *PDT+1&2* combines the features of both of the *PDT+1* and *PDT+2* techniques. *PDT+1&2* decreases the input drop rate (Figure 7a), the response time (Figure 7b), and the output loss rate (Figure 7c). There is a reduction of up to 78% in the output loss rate over the *simple PDT* for 2000 sensors.

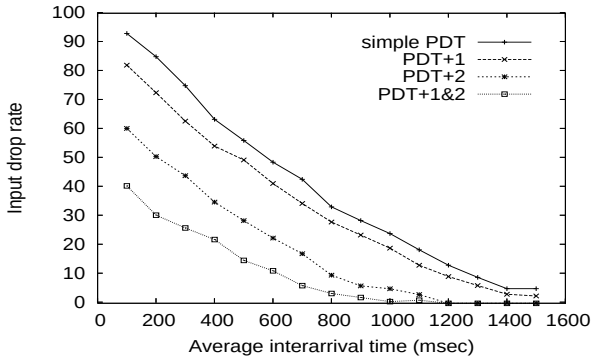
6.3. The effect of the stream rates

Figure 8 compares the performance of the four variations of the *PDT* framework with respect to varying the sensor data interarrival rates from 0.1 to 1.5 seconds. The *PDT* parameters α and w are set to 5 and 10 seconds, respectively. Small interarrival times imply scarcity in resources. Large interarrival times imply an increased availability of resources, where all curves approach each other and the drop rate approaches zero. For the same reasons, as discussed in Section 6.2, both *PDT+1* and *PDT+2* decrease the input drop rate over the *simple PDT* (Figure 8a), the response time (Figure 8b), and the output loss rate (Figure 8c). The *PDT+1&2* combines the benefits of both optimizations and reduces the output loss rate by up to 45% over the *simple PDT* (for a 0.1 second average interarrival time).

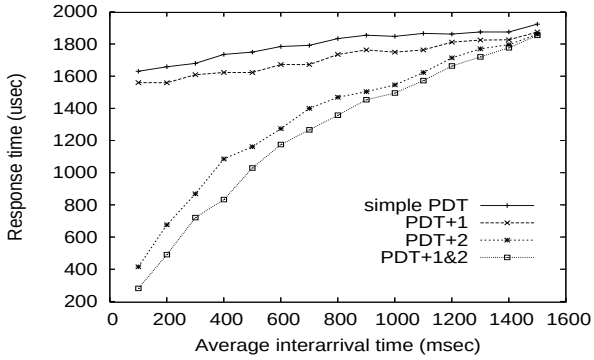
Notice that the response times of the *simple PDT* and *PDT+1* are constant over time because the join probing sequence spans all of the 1000 sensors for all stream rates. However, the *PDT+2* and *PDT+1&2* increase the length of the probing sequence with the increase of the time availability. The response time increases because the technique traverses a larger probing sequence per tuple for the sake of decreasing the output loss rate.

7. Related Work

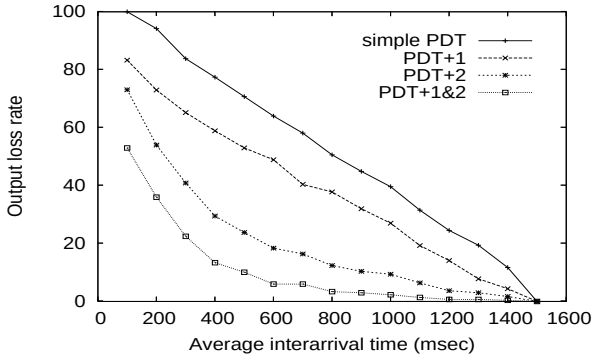
A lot of research interest has been directed recently to data stream processing. Data stream systems, e.g., Stanford STREAM [22], AURORA [1], NiagaraCQ [7], Telegraph [6], are developed to cope with the new challenges imposed by the nature of data streams [4]. The COUGAR system [5, 30] introduces a new abstract data type for sen-



(a) The input drop rate



(b) The response time



(c) The output loss rate

Figure 8. The effect of the stream rates

sors to facilitate the extraction of data and to process queries over sensor networks. The Borealis engine [2] proposes a scalable QoS-based optimization model to operate across sensor networks. The Fjords architecture [17] proposes an infrastructure for query processing over sensor data.

The physical acquisition or sampling of sensor data is explored in [19, 11, 16, 23] to extract the sensor data with low cost but still accurate methods. The work in [8, 18, 20] proposes the aggregation of sensor data tuples before reaching the data management system. Aggregates reduce the data size and, consequently, consume less power in the transmission process.

Some prior work has been conducted to track moving objects in sensor networks [12]. Similar to our work, the join operation is used to detect similar readings over the sensor network. It proposes a new non-blocking multi-way join operator over a sliding window, the W-join operator, to track the readings of the same object-ID that appears in different locations over the sensor network. Our work tracks moving phenomena, where each phenomenon is a group of sensors producing the same value, rather than tracking individual moving point objects.

The join operation over data streams has been explored in the literature. Symmetric Hash Join [29] is proposed to take care of the infiniteness of the data source. XJoin [27] provides disk management to store overflowing tuples on disk for later processing. An asymmetric window join over two data streams with different arrival rates is discussed in [15]. The Hash-Merge Join (*HMJ*) [21] is a non-blocking join algorithm that produces early join results. In our work, a modified version of the M-Join [28] is used to detect streams with similar behavior over a window of time.

8. Conclusions

In this paper, we proposed a framework for *phenomena detection* and *tracking* (*PDT, for short*) in sensor network databases. To identify a phenomenon, we provided a *crisp* definition for the phenomenon that takes into consideration both the strength (α) and the time span (w). A *phenomenon* of strength (α) and time span w occurs at least α times in the last w time units.

The proposed *PDT* framework has three phases: The *joining* phase, the *candidate selection* phase, and the *grouping/output* phase. The *joining* phase employs a multi-way join algorithm that joins the raw data from the sensor network and produces a set of sensor pairs with similar values. The *candidate selection* phase takes the output of the joining phase as input and applies a filter to enforce the strength and time span of the phenomena. Finally, the *grouping/output* phase is application-specific where it enforces the application semantics. Furthermore, we provided two *phenomenon-aware* optimizations that aim to : (1) Increase the sampling rate of the sensors that are part of any phenomenon, and (2) Choose the join order of the multi-way join algorithm to increase the likelihood of detecting the phenomena.

Experimental study based on a real implementation inside a research prototype for data stream management systems shows that the proposed *PDT* technique is scalable in terms of the number of streams, the stream rates, and the number of detected phenomena. The *optimized PDT* reduces the output loss rate over the *simple PDT* by up to 78% for a network of size 2000 sensors.

References

[1] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 2:120–139, August 2003.

[2] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *Proceedings of the Second Conf. on Innovative Data Systems Research (CIDR)*, Jan. 2005.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of VLDB*, pages 487–499, Sept. 1994.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of PODS*, pages 1–16, June 2002.

[5] P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proceedings of the Intl. Conf. on Mobile Data Management*, pages 3–14, Jan. 2001.

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, and et al. Telegraphcq: Continuous dataflow processing for an uncertain world. In *Proceedings of the First Conf. on Innovative Data Systems Research (CIDR)*, Jan. 2003.

[7] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagracq: A scalable continuous query system for internet databases. In *Proceedings of ACM SIGMOD*, pages 379–390, May 2000.

[8] J. Considine, F. Li, G. Kollios, and J. W. Byers. Approximate aggregation techniques for sensor databases. In *Proceedings of ICDE*, pages 449–460, April 2004.

[9] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of VLDB*, pages 588–599, August 2004.

[10] A. Deshpande, S. Nath, P. B. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *Proceedings of ACM SIGMOD*, pages 503–514, June 2003.

[11] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin. Coping with irregular spatio-temporal sampling in sensor networks. *Computer Communication Review*, 34(1):125–130, 2004.

[12] M. A. Hammad, W. G. Aref, and A. K. Elmagarmid. Stream window join: Tracking moving objects in sensor-network databases. In *Proceedings of the Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, pages 75–84, July 2003.

[13] M. A. Hammad, T. M. Ghanem, W. G. Aref, A. K. Elmagarmid, and M. F. Mokbel. Efficient execution of sliding-window queries over data streams. Technical Report CSD-03-035, Department of Computer Science, Purdue University, June 2004.

[14] M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, A. C. Catlin, A. K. Elmagarmid, M. Eltabakh, M. G. Elfeky, T. Ghanem, R. Gwadera, I. F. Ilyas, M. Marzouk, and X. Xiong. Nile: A query processing engine for data streams. In *Proceedings of ICDE*, page 851, April 2004.

[15] J. Kang, J. F. Naughton, and S. D. Viglas. Evaluating window joins over unbounded streams. In *Proceedings of ICDE*, pages 341–352, Feb. 2003.

[16] I. Lazaridis, Q. Han, X. Yu, S. Mehrotra, N. Venkatasubramanian, D. V. Kalashnikov, and W. Yang. Quasar: quality aware sensing architecture. *SIGMOD Record*, 33(1):26–31, 2004.

[17] S. Madden and M. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proceedings of ICDE*, pages 555–566, Feb. 2002.

[18] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. In *USENIX OSDI*, 2002.

[19] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of ACM SIGMOD*, pages 491–502, June 2003.

[20] S. Madden, R. Szewczyk, M. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks.

[21] M. Mokbel, M. Lu, and W. Aref. Hash-merge join: A non-blocking join algorithm for producing fast and early join results. In *Proceedings of ICDE*, pages 251–262, April 2004.

[22] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *Proceedings of the First Conf. on Innovative Data Systems Research (CIDR)*, Jan. 2003.

[23] J.-Y. Pan, S. Seshan, and C. Faloutsos. Fastcars: Fast, correlation-aware sampling for network data mining. In *GLOBECOM 2002 - IEEE Global Telecommunications Conf.*, pages 2167 – 2171, November 2002.

[24] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of EDBT*, pages 3–17, March 1996.

[25] S. Srinivasan, H. Latchman, J. Shea, T. Wong, and J. McNair. Airborne traffic surveillance systems: video surveillance of highway traffic. In *The 2nd ACM international workshop on Video surveillance & sensor networks*, pages 131–135, 2004.

[26] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Communications of ACM*, 47(6):34–40, 2004.

[27] T. Urhan and M. J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Eng. Bull.*, 23(2):27–33, 2000.

[28] S. Viglas, J. F. Naughton, and J. Burger. Maximizing the output rate of multi-way join queries over streaming information sources. In *Proceedings of VLDB*, pages 285–296, Sept. 2003.

[29] A. N. Wilschut and E. M. G. Apers. Pipelining in query execution. In *Proceedings of the International Conference on Databases, Parallel Architectures and their Applications*, 1991.

[30] Y. Yao and J. Gehrke. Query processing in sensor networks. In *Proceedings of the First Conf. on Innovative Data Systems Research (CIDR)*, pages 233–244, Jan. 2003.

A Query Language for Moving Object Trajectories*

(Extended Abstract)

Hoda M. O. Mokhtar
Department of Computer Science
University of California at Santa Barbara
hmokhtar@cs.ucsb.edu

Jianwen Su
Department of Computer Science
University of California at Santa Barbara
su@cs.ucsb.edu

Abstract

Trajectory properties are spatio-temporal properties that describe the changes of spatial (topological) relationships of one moving object with respect to regions and trajectories of other moving objects. Trajectory properties can be viewed as continuous changes of an object's location resulting in a continuous change in the topological relationship between this object and other entities of interest. In this paper we develop a query language TQ for expressing trajectory properties. Our model and query language are based on the framework of constraint query languages. We present some preliminary complexity and expressive power results for the proposed language.

1 Introduction

Rapid technology advancement is revolutionizing the modern society in almost every possible ways. In particular, the ever shrinking computing devices and wireless communication devices are making it easier to collect, transmit, and process data. For example, wireless applications are now seen in everyday activities ranging from mobile phones applications to military and navigational applications. Today's market already has cellular phones (GSM), global positioning systems (GPS), traffic navigational systems, sensor networks, digital assistants (PDA); more innovations are on their way. Such an explosion of technology not only brings new problems but also presents serious challenges in areas such as data management.

Moving object databases (spatio-temporal databases) are one of those recent evolutions that emerged to fulfil some of the new urging requirements. Moving object databases integrate traditional spatial and temporal databases and studies managing and querying objects whose location and/or shape change over time. Moving object databases appear in numerous applications including emergency services (E911), navigational and military services, flight management and tracking, m-commerce, and various location based services (LBS) as fleet management, vehicle tracking, mobile advertisements, etc. These advancements demand new techniques for managing and querying changing location information.

The aim of this paper is to study time varying topological properties for moving objects that we refer to as trajectory properties. In general, both moving object databases and topological properties have been of interest in a number of research work. For moving objects databases many research work was conducted to examine some problems including modeling and query languages [33, 38, 14, 12, 18, 35, 26, 7], handling large volume of location information through the use of efficient index structures [29, 21, 1, 32, 36, 5, 24], efficient data management, specifically, processing queries and handling updates [33, 34], query evaluation [20, 34, 3], and uncertainty management [28, 30, 37, 4, 25]. On the other hand topological predicates where studied in several works including [10, 11, 22, 27].

Nevertheless, designing efficient trajectory query languages continues to be an interesting problem. Prior approaches to query languages are mostly based on extending known languages or frameworks for spatial or spatio-temporal databases to allow expressing properties concerning trajectories (e.g., [33, 7, 18, 35, 15]). While general purpose spatial query languages could be used, the uniqueness of trajectories representing objects moving in space makes these general-purpose spatio-temporal languages unfit. For example, [26] exhibits difficulties for the general constraint query language to express some trajectory specific queries. In fact, querying trajectories seems to demand new techniques in the query languages. The recent effort in [13, 9] made a significant step towards understanding some of these issues. In [13] Erwig and Schneider elevate topological predicates to spatio-temporal predicates by aggregating a

*Support in part by NSF grant IIS-0101134.

temporal dimension to spatial predicates. They present a framework for expressing spatio-temporal predicates. The framework is based on aggregating time to elementary spatial predicates and sequential composition of predicates. In [9] du Mouza and Rigaux introduced the notion of mobility patterns which describe the motion pattern of a moving object. They model a moving object trajectory as a sequence of spatial zones and time spent at each zone. Their focus is on continuous evaluation of pattern matching queries and how to incrementally maintain the results.

Following those efforts we propose in this paper a constraint based query language (TQ) for reasoning about trajectory properties. We use the prevailing model of linear motions for moving object trajectories and allow a database to consist of a finite set of trajectories and a finite set of regions. We adopt constraint databases techniques [19, 23, 6, 16] to represent the trajectories and regions as "generalized relations", i.e., boolean combination of linear constraints interpreted over the structure of real numbers. By basing our design on constraint databases, many techniques of constraint query languages and evaluations are immediately available to use.

Our language TQ consists of two components: a spatial component (called SQ) focussing on expressing spatial relationships at a time instant and a temporal component. SQ generalizes the classical CQL of [19] by allowing variables to represent regions and trajectories. This permits expressing spatial properties among trajectories and regions. SQ queries are then generalized to allow existential and universal quantification on a time interval, i.e., interval based spatial properties. The instant/interval properties are then put together in the temporal component as regular (formal) languages to model properties of trajectories. While such a method of "gluing" spatial properties along the time line is similar to [13, 9], it turns out that the use of constraint query languages in SQ significantly enhances the query language. We provide an expressive power comparison of our language with the languages in [13, 9] in the paper.

We study the expressive power of TQ and the complexity of evaluating queries in TQ. In this paper we establish the following technical results.

1. SQ (i.e. snapshot query language) has polynomial time data complexity and exponential space combined complexity (i.e. query expression and database complexity).

2. TQ queries can be effectively evaluated.

3. TQ is more expressive than the language of [13] and the variable free queries of [9].

The paper is organized as follows. Section 2 defines the model for the moving object trajectories. Section 3 introduces the language TQ for trajectories. Section 4 presents the complexity results for TQ and a sublanguage of TQ (star-free). Section 5 studies expressive power of TQ. Section 6 concludes the paper.

2 A Data Model for Moving Objects

In this section we present necessary concepts for spatio-temporal objects to be used in the paper. Key notions include that of a "moving object" and its "trajectory." Roughly speaking, moving objects are spatio-temporal objects whose location and/or extent change over time. In general, such spatio-temporal objects can be points or regions. Moving points are suitable to model planes, cars, buses, trains, people, etc. whose locations and movements are important. On the other hand, applications concerning oil spills, fires, forests, pollution, etc. are very dependent on both shapes and locations of such moving objects. In this paper, we assume moving points and static regions to focus on query languages for trajectories. With some proper modeling of regions that changing over time, the results may be generalized to include such regions.

Our data model is based on "linear constraints" that are logical formulas over the real closed field. Kanellakis, Kuper, and Revesz in their seminal paper [19] demonstrated that such logical formulas can represent spatial and spatio-temporal information in an abstract manner independent of the underlying storage mechanisms (cf [23]).

Specifically, our model is an extension of the constraint data model [19] to represent a finite set of regions[1] and moving object trajectories in the database. Some key techniques of the results concerning constraint formulas in this paper are also extended from the constraint query evaluation techniques [23]. More details will be discussed in Sections 3 and 4.

We now proceed with the technical presentation, starting with constraints. Let $\mathbb{R}, \mathbb{N}$ be the set of real and natural numbers (respectively). Consider a first order language L for $\mathbb{R}$ that includes equality and order predicates $(=, <, \leqslant, >, \geqslant)$, a binary function for addition $(+)$, and a unary *coefficient* function "$c\cdot$" for each real number $c \in \mathbb{R}$. Intuitively, the unary coefficient functions are used to represent real coefficients. For simplicity, we will denote "$c\cdot(x)$" as "cx" for each $x \in \mathbb{R}$.

Let $n \in \mathbb{N}$ and $n > 0$. An *atomic linear constraint* over variables $x_1, ..., x_n$ is an expression of the following form:

$$(\Sigma_{i=1}^{n} c_i x_i)\ \theta\ c_0 \quad \text{or} \quad c_1x_1 + c_2x_2 + \cdots + c_nx_n\ \theta\ c_0$$

where $c_0, c_1, ..., c_n$ are real numbers in $\mathbb{R}$ and θ is a predicate in L. Constraints are interpreted over the real numbers in the natural manner, i.e., if $\varphi(x_1, ..., x_n) =$

[1] Regions in this paper are non-changing and spatial. Generalization to time-dependent regions can be done easily.

$(\Sigma_{i=1}^{n} c_i x_i)\theta c_0$ is an atomic constraint, and $a_1, ..., a_n$ are real numbers in $\mathbb{R}$, $\varphi(a_1, ..., a_n)$ is true if $(\Sigma_{i=1}^{n} c_i a_i)\theta c_0$ is satisfied (interpreted over the real numbers).

A *linear constraint* over variables $x_1, ..., x_n$ is a boolean combination of atomic linear constraints over variables $x_1, ..., x_n$. An advantage of (linear) constraints is their ability to *finitely* represent potentially *infinite* sets of points (i.e., regions).

Definition: Let $n > 0$ be a natural number. An n-*dimensional region* is a linear constraint in disjunctive normal form over n variables. Let Reg^n denote the set of all n-dimensional regions.

In the remainder of the paper, we will fix the dimensionality of the space to be n for some $n > 0$ and may simply use Reg instead of Reg^n.

We now consider moving objects and regions in the n-dimensional real space $\mathbb{R}^n$. Object movements are viewed as location changes over "time". We model the time domain as a domain isomorphic to $\mathbb{R}$ (with a dense total order, addition, and multiplication by a real number). As we shall see in the technical presentation, we will introduce some restrictions on time instants in both the data model and query languages. To make the presentation clear, we use $\mathbb{T}$ to denote the densely ordered domain of *time instants*. We only allow one variable for the time domain, denoted as t.

A moving object trajectory basically defines the motion of an object. Different alternatives for modeling trajectories have been studied in [33, 38, 14, 12, 18, 35, 26]. In this paper we use the standard model for trajectory as a sequence of line segments in the n-dimensional space. Instead of keeping the segments' endpoints to define a trajectory as in [14, 29], we use constraints to define a trajectory. Constraints allow us to focus on logical properties in querying trajectories. Technically, a trajectory is a sequence of linear motions defined as follows.

Let Flin be a set of expressions of the form "at$+b$" where $a, b \in \mathbb{R}$. Each element in Flin represents a linear function from $\mathbb{T}$ to $\mathbb{R}$.

Definition: An n-dimensional *motion* is an n-tuple $m \in \mathsf{Flin}^n$. For each $1 \leqslant i \leqslant n$, m_i denotes the i-th value of m.

Intuitively, an n-dimensional motion m defines a linear function from $\mathbb{T}$ to $\mathbb{R}^n$; m_i represents the location change along the i-th dimension. For example, for a given time instant a and a motion m, $m(a)$ represents the point (location) in $\mathbb{R}^n$ at time a, while $m_i(a)$ is the position on the i-th dimension.

Definition: A *trajectory* is a sequence $(z_0, m_0, z_1, m_1, ..., z_k, m_k)$, where $k \in \mathbb{N}$ and for each $0 \leqslant i \leqslant k$,

- z_i is a time instant in $\mathbb{T}$ such that for each $0 \leqslant i < k$, $z_i < z_{i+1}$, and
- m_i is a motion such that for $0 \leqslant i < k$,

$$m_i(z_{i+1}) = m_{i+1}(z_{i+1}).$$

Let Traj denote the set of all trajectories.

Intuitively, in a trajectory $(z_0, m_0, z_1, m_1, ..., z_k, m_k)$, the motion m_i (for $1 \leqslant i < k$) defines the object location from time z_i to z_{i+1}, m_k defines the object location for all times after z_k, and the object does not exist before time z_0. It is easy to see that a trajectory is a continuous piecewise linear function from $\mathbb{T}$ to $\mathbb{R}^n$ which always has a starting time but no ending time.

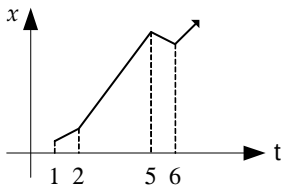


Figure 1. A 1-dimensional Trajectory

Example 2.1 Consider the 1-dimensional space and a moving object whose trajectory is defined as:

$$(1, (\mathrm{t}+4), 2, (2\mathrm{t}+2), 5, (-\mathrm{t}+17), 6, (2\mathrm{t}-1)).$$

Figure 1 shows the beginning part of the trajectory. The trajectory consists of a sequence of four motions h, p, q, r where $h = (\mathrm{t}+4)$ defined the object location for the time interval $[1, 2]$, $p = (2\mathrm{t}+2)$ for the time interval $[2, 5]$, $q = (-\mathrm{t}+17)$ for the time interval $[5, 6]$, and $r = 2\mathrm{t}-1$ for all times after 6. ∎

We assume that $\mathbf{U}_\mathrm{R}$ and $\mathbf{U}_\mathrm{T}$ are two disjoint countably infinite set of names (identifiers) for regions and trajectories, respectively.

Definition: A *(moving object) database* (or *MOD*) is a quadruple $d = (N_R, N_T, f_R, f_T)$ where $N_R \subseteq \mathbf{U}_\mathrm{R}$ and $N_T \subseteq \mathbf{U}_\mathrm{T}$ are finite subsets, f_R is a mapping from N_R to Reg, and f_T is a mapping from N_T to Traj.

The data model (trajectories and regions) are similar to the ones in the literature (e.g., [18, 38, 35]), except that regions do not change over time. The trajectory model was originally defined in [26], it is different from the model used in [9] where a trajectory is a sequence of spatial zones and time values representing the time spent at each zone. In [13] the authors also discuss spatio-temporal properties but no model for trajectories is provided in their framework.

Finally, the model ignores the usual semantic modeling, i.e., separating the set of "cars" from that of "trucks". This is to simplify the technical presentation and to allow us focussing on the querying aspect of trajectories.

3 A Query Language for MOD

In this section we propose a trajectory query language (TQ) for expressing spatio-temporal properties of moving object trajectories. TQ is based on constraint query languages and is powerful enough to express properties between objects and regions and between different objects.

Definition: A *(spatial) term* is an expression of one of the following forms:

- a where $a \in \mathbb{R}$,
- x where x is a spatial (i.e., real) variable,
- cx where $c \in \mathbb{R}$ is a coefficient function and x a spatial variable, and
- $s_1 + s_2$ where s_1 and s_2 are spatial terms.

We now define the central notion of a "spatial formula". Intuitively, a spatial formula can express properties concerning positions with the time variable t as a parameter. Spatial formulas resemble formulas in constraint query languages of [19]. However, the primary difference is that in constraint query languages, only named regions can be used, while in our language, spatial formulas can reference named regions (i.e., region names), named trajectories, and also variables that represent regions and trajectories. This is natural since a moving object database may have arbitrary number of regions and trajectories.

Recall that n is the number of dimensions and t is the time variable.

Definition: The set of *spatial formulas* is defined recursively as follows.

- $s_1 \theta s_2$ is an *atomic* spatial formula if s_1 and s_2 are spatial terms and $\theta \in \{=, <, >, \leqslant, \geqslant\}$,
- $v(s_1, ..., s_n)$ is an *atomic* spatial formula if v is a region variable or a region name in $\mathbf{U}_\mathrm{R}$ and $s_1, ..., s_n$ are spatial terms,
- $v(\mathsf{t}, s_1, ..., s_n)$ is an *atomic* spatial formula if v is a trajectory variable or a trajectory name in $\mathbf{U}_\mathrm{T}$ and $s_1, ..., s_n$ are spatial terms,
- $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, and $(\neg\varphi)$ are spatial formulas if φ and ψ are spatial formulas,
- $\exists x\varphi$, $\forall x\varphi$ are spatial formulas if x is a spatial variable and φ a spatial formula.

The semantics of spatial formulas is defined in the straightforward manner. Technically, a *valuation* is a mapping that maps

- each spatial variable to $\mathbb{R}$,
- the time variable t to $\mathbb{T}$,
- each region name and each region variable to a region (a generalized relation) in Reg, and
- each trajectory name and each trajectory variable to Traj.

A valuation can be naturally extended to all spatial terms. Then the *truth* value of a spatial formula φ under a valuation is identical to a constraint formula in a constraint query language with the following exception:

- We will represent the time domain $\mathbb{T}$ as real numbers in $\mathbb{R}$, and consequently, each trajectory in Traj is treated as an $(n + 1)$-ary generalized relation with the first dimension representing the time.

Spatial formulas are then used to construct "query formulas" defined below.

Definition: Let $d = (N_R, N_T, f_R, f_T)$ be a MOD. The set of *query formulas* over d is defined below.

- A spatial formula φ is a query formula if φ has no spatial variables occurring free and each region (respectively trajectory) name occurring in φ is in N_R (respectively N_T).
- $z_1{=}z_2$ is a query formula if both z_1, z_2 are region variables/names, or trajectory variables/names.
- $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\neg\varphi)$, $(\exists z\varphi)$, $(\forall z\varphi)$ are query formulas if z is a region variable or trajectory variable and $\varphi, \varphi_1, \varphi_2$ are query formulas.

A *snapshot (trajectory) query* is a query formula with exactly two variables occurring free: the time variable t and a trajectory variable. Let SQ denote the set of all snapshot queries.

The semantics of the query language SQ is defined in the standard way based on the semantics of spatial formulas. Let $d = (N_R, N_T, f_R, f_T)$ be a moving object database and $\varphi(\mathsf{t}, \mathsf{z})$ a snapshot query. A mapping is said to be an *assignment* with respect to d if it maps each variable to its domain, i.e., spatial variables to $\mathbb{R}$, t to $\mathbb{T}$, region variables to N_R, and trajectory variables to N_T. Indeed, let α be an assignment, we can derive a valuation by simply taking the composition $\alpha \circ (f_R \cup f_T)$. The semantics for quantifiers on region and trajectory variables is standard, here we take the "active domain" semantics for these variables, i.e., only region and trajectory names in the database are considered. Finally, we say that the database *satisfies* φ at time $c \in \mathbb{T}$ for some trajectory (name) $\tau \in N_T$, written as $(d, c) \models \varphi[\tau]$, if the query formula returns true.

Example 3.1 Let τ be the name of an object trajectory and r_1, r_2 be names of two regions in the 2 dimensional space. The property that τ is in the intersection of r_1, r_2 can be expressed as the following spatial formula (snapshot query).

$$\exists x \exists y\ \tau(\mathsf{t}, x, y) \wedge r_1(x, y) \wedge r_2(x, y)$$

The property that no regions intersecting the rectangle of size $2a$ centered at τ can be expressed by the following snapshot query:

$$\exists x \exists y\ \tau(\mathsf{t}, x, y) \wedge \forall r \forall x' \forall y' (r(x', y') \rightarrow \neg(x{-}a \leqslant x' \leqslant x{+}a \wedge y{-}a \leqslant y' \leqslant y{+}a))$$ ∎

Snapshot queries provide a basis for expressing trajectory properties. In [13, 9] trajectory properties are considered as spatial properties varying along time. Similar to their approaches, we will develop a "temporal" portion on top of the SQ language to express trajectory properties. Different from their approaches, our language allows explicit definition of transition time (from one spatial property to the next) through querying the database explicitly, and the use of SQ significantly enriches the expressiveness of spatial properties.

Trajectory properties maybe expressed as snapshot properties or interval properties. As in SQ, snapshot properties focus on the properties that a moving object trajectory would satisfy at a given time instant, whereas interval properties require a moving object trajectory to satisfy the property during a given interval.

Since trajectory properties depend on either the time instant or the time interval it is checked against, expressing time in a trajectory property is a key issue. In many applications, a query checks a reoccurring property that repeats itself at different time instants. Such queries require the language to support expressing both absolute and relative times. The following example illustrates the difference between queries that require absolute and relative time resp.

Example 3.2 Consider the following queries on trajectories:

Q_1: Retrieve all delivery trucks that entered the Santa Barbara area at 2:00pm and stayed there until 5:00pm.

Q_2: Retrieve all delivery trucks that entered the Santa Barbara area at 2:00pm and stayed there for 35 minutes.

The main difference between Q_1 and Q_2 is that the ending time for Q_1 is fixed. On the other hand, Q_2 requires the trajectory property of staying inside Santa Barbara to be checked until 35 minutes after the time when the object entered. Q_2 thus requires the language to express things like "entering time" $+35$. ∎

In our design, we capture this requirement through (1) the use of "time expressions" to define time instants, (2) the use of a "relative time expressions" to refer to the difference between the current time and the previous time instant of interest.

To facilitate the development, we start with some necessary technical notions. Note that t is the time variable,

Definition: Let φ be a query formula with only t occurring free. Then, a *time expression* is one of the following:

- c, if $c \in \mathbb{T}$,
- $\min(\varphi)$, or
- $\max(\varphi)$.

A *relative time expression* is an expression "$+e$" where e is a time expression and "$+$" is a special symbol.

Intuitively, a (relative) time expression defines a time instant either explicitly or through querying the database. The functions $\min, \max$ returns the smallest, respectively largest time instant that satisfies the query formula. In case the smallest/largest instant does not exist, the expression is not *well-formed*. Let e be a time expression and d a database. We denote by $e(d)$ the *result* of e under d. The following property shows that well-formedness can be checked if a database is given; undecidable if it is well-formed independent of databases.

Proposition 3.3 For a given time expression e and a MOD d, it can be decided where e is well-formed. However, it is undecidable if e is well-formed for all databases.

The positive result can be proved using an argument similar to the proof of Lemma 4.1 in Section 4 about complexity. The negative results is a direct consequence of undecidability results of constraint query properties [17].

Combining snapshot queries with (relative) time expressions we can define a language for expressing trajectory properties.

Definition: Let z be a designated trajectory variable. If ϵ is a (relative) time expression and φ is a snapshot query in SQ, then the following are *startless atomic trajectory queries*:

- $\varphi(\mathsf{t}, \mathsf{z})$ is a startless *snapshot* trajectories query,
- $\epsilon^{\exists}.\varphi(\mathsf{t}, \mathsf{z})$ is a startless *existential-time* trajectory query, and
- $\epsilon^{\forall}.\varphi(\mathsf{t}, \mathsf{z})$ is a startless *universal-time* trajectory query.

A *startless trajectory query* is a regular expression over the set of startless atomic trajectory queries, i.e., composed from startless atomic trajectory queries using concatenation ($q_1 q_2$), union ($q_1 + q_2$), and closure (q^*), where q, q_1, q_2 are

startless trajectory queries. If the regular expression does not use the closure operator, the startless trajectory query is called *star-free*.

A startless trajectory query q defines a formal language (i.e., set of words) over startless atomic trajectory queries. We denote the language as $\text{SEQ}(q)$. Each word expresses a sequence of atomic properties. There are three types of atomic properties in the language. A snapshot trajectory query examines the spatial properties at a time instant. An existential-time trajectory query checks if the spatial properties hold at some time instant during a time interval. Finally, a universal-time trajectory query insists that the spatial properties should be true for all time instants during the time interval. The word "startless" indicates that the start time is not given. The end of a time interval for an existential- or universal-time trajectory query is given by the time expression ϵ preceding the snapshot query formula.

Definition: An (*atomic, snapshot, existential-time, universal-time, star-free*) *trajectory query* is a pair (ϵ, q), where ϵ is a time expression and q a startless (resp., atomic, snapshot, existential-time, universal-time, star-free) trajectory query. Let TQ denote the set of all trajectory queries.

We outline the semantics for TQ below.

Let $c_0 \in \mathbb{T}$ be a time instant, $q_1 \cdots q_\ell$ be a sequence of atomic trajectory queries, and d a MOD database. We define a time instant sequence $c_0, c_1, ..., c_\ell$ as follows: for each $1 \leqslant i \leqslant \ell$,

- if q_i is a snapshot query, $c_i = c_{i-1}$,
- otherwise, let e_i be the time expression in q_i. Let $c_i = c_{i-1} + e_i(d)$ if q_i has a relative time expression, otherwise let $c_i = e_i(d)$.

Let $\tau \in \mathbf{U}_\text{T}$ be a trajectory name. We say that the database d *satisfies* the atomic query sequence $q_1 \cdots q_\ell$ for τ at c_0, denoted as $(d, c_0) \models q_1 \cdots q_\ell[\tau]$, if the sequence $c_0, c_1, ..., c_\ell$ is monotonically increasing (not necessarily strict) and for each $1 \leqslant i \leqslant \ell$, the following are true:

1. If $q_i = \varphi(\mathsf{t}, \mathsf{z})$ is a snapshot query, then $(d, c_i) \models \varphi[\tau]$,
2. If $q_i = \epsilon^{\exists}.\varphi(\mathsf{t}, \mathsf{z})$ is an existential-time query, then $(d, c) \models \varphi[\tau]$ for some $c_{i-1} < c < c_i$, and
3. If $q_i = \epsilon^{\forall}.\varphi(\mathsf{t}, \mathsf{z})$ is a universal-time query, then $(d, c) \models \varphi[\tau]$ for each $c_{i-1} < c < c_i$.

Let $q = (\epsilon, q_1 \cdots q_\ell)$ be a trajectory query where each q_i ($1 \leqslant i \leqslant \ell$) is atomic, and $d = (N_R, N_T, f_R, f_T)$ a database. A trajectory (name) $\tau \in N_T$ is in the *answer* to q over d, if $(d, \epsilon(d)) \models q_1 \cdots q_\ell[\tau]$. Denote by $q(d)$ the set of all trajectories in the the answer to q over d.

Let $q = (\epsilon, q')$ be a trajectory query and $d = (N_R, N_T, f_R, f_T)$ be a MOD database. The *answer* of q over d is the set $q(d) =$

$$\{\tau \mid \tau \in N_T, \text{and } \exists q'' \in \text{SEQ}(q'), (d, \epsilon(d)) \models q''[\tau]\}$$

4 Complexity Results

In this section we present complexity results for the trajectory query language TQ. In the literature, data and combined complexity are used to measure the complexity of query languages. The main results in the section show that SQ and TQ have polynomial time data complexity and exponential space combined complexity.

The *complexity* of a query is the time/space needed to compute the answer. We consider two complexity measures. *Data complexity* of a query measures the complexity in terms of the database size, i.e., the query expression is considered fixed; *combined complexity* of a query measures the complexity in terms of both the database size and query expression size. We use PTIME, EXPSPACE to denote the polynomial time, (respectively) exponential space complexity classes. A query language has *complexity* C if every query in the language has complexity C.

Lemma 4.1 The snapshot query language SQ has PTIME data complexity and EXPSPACE combined complexity.

We note here that the main source of combined complexity is from the number of variables. Indeed, we can show that the space complexity is exponential in the number of variables in a query expression φ.

Proof: (Sketch) Technically, the result states that for each time instant $c \in \mathbb{T}$, each SQ query $\varphi(\mathsf{t}, \mathsf{z})$, each database $d = (N_R, N_T, f_R, f_T)$, and each trajectory (name) $\tau \in N_T$, $(d, c) \models \varphi[\tau]$ can be decided in (1) polynomial time in the size of d, and (2) exponential space in the size of d and φ.

The main idea of the proof is as follows. We consider a fixed mapping γ that assigns a region name in N_R to each region variable and a trajectory name in N_T to each trajectory variable including τ. Let $Q[\gamma]$ denote the snapshot query obtained from Q by replacing each region/trajectory variable z by $\gamma(z)$. Note that $Q[\gamma]$ is a snapshot query without any region and trajectory variables, it does have, however, spatial variables. Mapping the time domain $\mathbb{T}$ to real numbers $\mathbb{R}$ and t to a spatial variable, we can now view $Q[\gamma]$ as a constraint query in the model of [19, 17]. It can be concluded that $Q[\gamma]$ can be evaluated in polynomial time in the size of the image of the composed mapping $\gamma \circ (f_R \cup f_T)$ which is a subset of d. From the results in [19], the result of evaluating $Q[\gamma]$ is a set of intervals for t that makes $\varphi[\gamma]$ true. In particular, the set can be computed effectively since $\varphi[\gamma]$ has only linear constraints.

Now let $\tau, z_1, ..., z_\ell$ be an enumeration of region and trajectory variables occurring in φ. We can consider all possible assignments from $\{\tau, z_1, ..., z_\ell\}$ to $N_R \cup N_T$ that preserve the type. For each such assignment γ, we can repeat the above process. Since the total number of assignments is $(|N_R| + |N_T|)^{\ell+1}$, and ℓ depends on the query expression φ that is considered as fixed, the number of assignment is a polynomial in the size of d.

Finally the combined complexity is due to fact that the complexity of evaluating queries is double exponential time in the number of quantifiers [31]. ∎

Theorem 4.2 Atomic trajectory queries have PTIME data and EXPSPACE combined complexity.

Proof: (Sketch) Clearly, snapshot queries have PTIME data and EXPSPACE combined complexity, by Lemma 4.1. From constraint query evaluation algorithms, we can see that the result of evaluating a snapshot query is a set of time instants (possibly infinite) represented in constraint form as a generalized relation. Therefore, checking the existential-time and universal-time property simply becomes checking the intersection and (respectively) containment of interval, which can be done efficiently. ∎

We now consider general trajectory queries. The idea is to use the evaluation procedure for atomic queries as the basis and extend for "path queries" defined in regular expressions. Technically, we will construct from a trajectory query a "query graph" and then develop a generic iterative process for query evaluation. We show that the iterative process ends in finite number of steps. The number of steps, however, may depend on the properties of the constraints used in defining the regions and trajectories in the database.

Let $q{=}(\epsilon, q')$ be a trajectory query. Since q' is a regular expression, let G_q be a finite automaton that accepts the same language as q' (see [2]). Without loss of generality, we assume that G_q does not have *empty* moves. In fact, G_q can be viewed as a *(query) graph* (V, E, v_0, F, Q) where V is a finite set of nodes (states), $E \subseteq V \times V$ is a set of edges between nodes, $v_0 \in V$ and $F \subseteq V$ are the starting and final nodes respectively, and Q is a mapping from E to atomic trajectory queries in q'.

A path $u_0, u_1, ..., u_\ell$ in a query graph G_q is called an *advancing edge* if (i) $\ell > 0$, and (ii) either $u_\ell \in F$ is a final node, or $Q(u_{\ell-1}, u_\ell)$ is not snapshot but $Q(u_{i-1}, u_i)$ is a snapshot for all $1 \leqslant i < \ell$.

Let $d = (N_R, N_T, f_R, f_T)$ be a database. We define an (infinite) relation $E \subseteq \mathbb{T}^2 \times V^2 \times N_T$ as follows:

- $(c_1, c_2, u_0, u_\ell, \tau) \in E$ if there is an advancing edge $u_0, u_1, ..., u_\ell$ such that if $Q(u_{\ell-1}, u_\ell)$ is not snapshot, $(d, c_1) \models \wedge_{i=1}^{\ell-1} Q(u_{i-1}, u_i)[\tau]$, c_2 is the result of (relative) time expression in $Q(u_{i-1}, u_i)$, and τ satisfies $Q(u_{\ell-1}, u_\ell)$ in d from time c_1 to c_2, otherwise, $(d, c_1) \models \wedge_{i=1}^{\ell} Q(u_{i-1}, u_i)[\tau]$ and $c_1 = c_2$.

By Theorem 4.2, $(c_1, c_2, u_0, u_\ell, \tau) \in E$ can be decided in polynomial time in the size of d.

Finally, we define a sequence of relations $A_i \subseteq \mathbb{T} \times V \times N_T$ as follows:

(i) $A_0 = \{c_0\} \times \{v_0\} \times N_T$ where c_0 is the value of ϵ, and

(ii) $A_{i+1} = A_i \bowtie E$, where the join is defined as $(c_2, u_2, \tau) \in A_i \bowtie E$ if both $A_i(c_1, u_1, \tau)$ and $E(c_1, c_2, u_1, u_2, \tau)$ hold.

Proposition 4.3 Given the above context, $q(d) = \pi_3 \sigma_{2 \in F} \cup_{i \geqslant 0} A_i$. ∎

The procedure outlined above can semi-compute the query answer, i.e., if a trajectory $\tau \in \pi_3 A_i$ for some i, then τ is in the answer. However, it is not clear when we can decide if τ is not in the answer. The following result implies that there is a time upper bound to decide if τ will never be in the answer.

Theorem 4.4 Given a trajectory query q, a MOD d, and a trajectory name in d, there exists $c \in \mathbb{T}$ such that for all $h, h' \in \mathbb{T}$ if $h > c$ and $h' > c$, $(d, h) \models q'[\tau]$ iff $(d, h') \models q'[\tau]$ where q' is an atomic query in q.

The proof of the result is rather involved and will be given in the full paper. The key idea is based on algebraic cell decomposition (e.g., [31, 8]). Specifically, for a given finite set of constraints, there is a partition of space into a finite number of "cells", each of which preserves the constraints. This intuitievly would indicate that after some point in time, the constraint (spatial) properties will not change.

Theorem 4.5
1. Let q be a trajectory query and d a MOD. It can be decided if a trajectory is in $q(d)$.
2. Star-free TQ has PTIME data and EXPSPACE combined complexity. ∎

In the remainder of the section we introduce a restricted sublanguage of TQ, called "intermediate-variable free" trajectory queries. Roughly, these queries use neither region nor trajectory variables, and have only n quantifiers over the spatial variables for the n dimensions.

Definition: A query formula is *intermediate-variable free*, (or *int-var-free*), if it only has n quantifiers, one for each spatial variable for a dimension. A trajectory query in TQ (snapshot query in SQ) is *intermediate-variable free* if all query formulas in it are int-var-free.

Example 4.6 The query "was the United Flight UA80 inside the Los Angeles airport between 5pm and 5:10pm?" can be expressed by in int-var-free TQ as

$$(e_0, e_1^{\forall}.\exists x_1 \exists x_2 \text{UA80}(\text{t}, x_1, x_2) \wedge \text{LAX}(x_1, x_2)),$$

where e_0, e_1 are time expressions. ∎

We argue that many queries can be expressed in this set. For technical results, we show that star-free trajectory queries in this subclass have PTIME combined complexity.

Theorem 4.7 Int-var-free SQ and star-free, int-var-free TQ have PTIME combined complexity. ∎

5 Expressive Power

In this section we study the expressive power of trajectory languages developed in this paper. Our main goal is to compare our languages with the trajectory query languages in [13, 9]. Our results show that TQ is quite expressive, and even int-var-free TQ is sufficient to express many queries in the languages of [13, 9].

Since we will compare different languages that are defined over different data models, we need to formulate the technical basis for comparison. We outline the comparison framework informally below, the precise definitions are provided in the full paper.

In the "mobility patterns" language developed by du Mouza and Rigaux [9], a database consists of (1) a set of regions that form a partition of $\mathbb{R}^2$, and (2) a set of objects each of which is a sequence of regions the object goes through (in that order) and the length of time it stays in each region. The time domain used in their model is discrete. For the sake of comparisons, we extend their time domain to a continuous one isomorphic to $\mathbb{T}$. Although the change may affect query evaluation, the semantics is clear. We use MP to denote the mobility patterns language with this extension.

Given a MOD d over $\mathbb{R}^2$, we map d into the MP data model in the following sense: we construct a partition of $\mathbb{R}^2$ using the regions in d by considering all possible combinations among them. Given a trajectory z in d, we then construct an MP object o by listing the regions in the partition z goes through. The time lengths can also be computed easily. We denote the mapping MP.

A TQ query Q_1 and an MP query Q_2 are *equivalent* if for each MOD d, $\text{MP}(Q_1(d)) = Q_2(\text{MP}(d))$.

A language QL_2 is as *expressive* as another language QL_1, denoted as $\text{QL}_1 \sqsubseteq \text{QL}_2$, if for every query in QL_1 there is an equivalent query in QL_2. QL_1 and QL_2 are *equivalent*, $\text{QL}_1 \equiv \text{QL}_2$, if $\text{QL}_1 \sqsubseteq \text{QL}_2$ and $\text{QL}_2 \sqsubseteq \text{QL}_1$.

Roughly speaking, a mobility pattern is a sequence of regions that a moving object goes through. Based on a single "inside-a-region" predicate, a mobility pattern is a regular expression over the predicates. MP allows regions to be identified with either names or variables. When a variable is used, the variable is instantiated *prior* to matching the patterns. In other words, region variables are global to the mobility patterns. Clearly TQ has no such global region variables and having such global region variable would increase expressive power. Thus,

Theorem 5.1 Variable-free MP $\sqsubseteq\not\equiv$ int-var-free TQ. ∎

The reason for the converse is that more spatial properties can be expressed in TQ using SQ.

The main reason that MP is not contained in int-var-free TQ is that it can use (existentially quantified) region variables while describing pattern expressions. It appears that TQ can be extended to allow free region and even trajectory variables while expressing patterns. Although it is interesting to work out the detailed semantics (e.g., allowing arbitrary quantifications), we think the data and combined complexity will remain unchanged with respect to the PTIME and EXPSPACE complexity classes.

We now consider the "language" proposed by Erwig and Schneider [13], which we call MS. Technically, MS allows expressing temporal properties in the same spirit as TQ, but it does not have a coincide language corresponding to SQ to express spatial properties. For our comparison purposes, we will use (a sublanguage of) SQ as the underlying sublanguage for spatial properties. For example, the language MS+SQ uses SQ for spatial properties and MS for temporal trajectory properties. Note that by using MS+SQ, comparisons will use MOD databases.

There are two important differences. Firstly, time is implicit in the predicates through lifting them from pure topological predicates to spatio-temporal predicates. In contrast, TQ allows an SQ query to be evaluated to define the time. For example, one can express in TQ when two moving objects enter region r_1, the third object must be outside of region r_2. Such a flexibility of referencing time is not provided in MS. Secondly, MS does not allow the use of closure in expressing patterns. Clearly this puts a severe restriction on its expressiveness.

Thus our comparison is based on the setting that when (int-var-free) SQ is used as the underlying spatial language in MS. We have:

Theorem 5.2 (1) MS+SQ $\sqsubseteq\not\equiv$ star-free TQ.

(2) MS+int-var-free SQ $\sqsubseteq\not\equiv$ star-free int-var-free TQ. ∎

6 Conclusions

In this paper we consider the problem of querying about spatial properties of moving objects. We presented a langauge TQ for expressing trajectory properties. We studied the complexity and expressive power of TQ. Our preliminary results show that TQ queries can be effectively evaluated, star-free TQ queries have PTIME data complexity, and intermediate variable free TQ queries have PTIME combined complexity.

Our work reported here makes only one step towards the design of query languages for trajectories. One immediate problem is to identify sublanguages of TQ that allow lower theoretical complexity bounds, efficient algorithms, and suitable data structures. It is unclear at this point how the known trajectory index techniques could be utilized in the query evaluation algorithms for trajectory query languages.

References

[1] P. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *19th ACM Symposium on Principles of Database Systems*, pages 175–186, 2000.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[3] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis. Nearest neighbor and reverse nearest neighbor queries for moving objects. In *Sixth International Database Engineering and Applications Symposium*, Edmonton, Alberta, Canada, July 2002.

[4] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *ACM International Conference on Management of Data SIGMOD*, 2003.

[5] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indxeing methods for probabilistic threshold queries over uncertain data. In *vldb*, Toronto Canada, 2004.

[6] J. Chomicki and P. Revesz. Constraint-based interoperability of spatiotemporal databases. *GeoInformatica*, 3(3):211–243, 1999.

[7] J. Chomicki and P. Z. Revesz. A geometric framework for specifying spatiotemporal objects. In *Proc. 6th Int. Workshop on Temporal Representation and Reasoning, (TIME '99)*, pages 41–46, Orlando, FL, May 1999.

[8] G. E. Collins. Quantifier elimination for real closed fields by cylindric decompositions. In *Proc. 2nd GI Conf. Automata Theory and Formal Languages*, volume 35 of *Lecture Notes in Computer Science*, pages 134–83. Springer-Verlag, 1975.

[9] C. du Mouza and P. Rigaux. Mobility patterns. In *Proceedings of the Second Workshop on Spatio-Temporal Database Management (STDBM04)*, Toronto, Canada, August 2004.

[10] M. J. Egenhofer. Reasonning about binary topological relations. In *Proc. Symp. on Large Spatial Databases*, 1991.

[11] M. J. Egenhofer and R. Franzosa. Point-set topological spatial relations. *Int. Journal of Geo. Info. Systems*, 5(2):161–174, 1991.

[12] M. Erwig, R. H. Guting, M. Schneider, and M. Vazirgiannis. Spatio-temporal data types: an approach to modeling and querying moving objects in databases. *GeoInformatica*, 3(3):269–296, 1999.

[13] M. Erwig and M. Schneider. Spatio-temporal predicates. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):881–901, 2002.

[14] L. Forlizzi, R. H. Guting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2000.

[15] S. Grumbach, P. Rigaux, and L. Segoufin. The DEDALE system for complex spatial queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, June 1998.

[16] S. Grumbach, P. Rigaux, and L. Segoufin. Spatio-temporal data handling with constraints. In *Proc. ACM Symp. on Advances in Geographic Information Systems*, 1998.

[17] S. Grumbach and J. Su. Finitely representable databases. *J. Comput. Syst. Sci.*, 55(2):273–298, Oct. 1997.

[18] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Varirgiannis. A foundation for representing and querying moving objects. *ACM Trans. Database Syst.*, 25(1), 2000.

[19] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *J. Comput. Syst. Sci.*, 51(1):26–52, 1995.

[20] G. Kollios, D. Gunopulos, and V. J. Tsotras. Nearest neighbor queries in a mobile environment. In *Spatio-Temporal Database Management*, pages 119–134, 1999.

[21] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *Proc. ACM Symp. on Principles of Database Systems*, pages 261–272, 1999.

[22] B. Kuijpers and J. V. den Bussche. On capturing first-order topological properties of planar spatial databases. In *Proc. Int. Conf. on Database Theory*, 1999.

[23] G. Kuper, L. Libkin, and J. Paredarns, editors. *Constraint Databases*. Springer Verlag, 2000.

[24] B. Lin, H. Mokhtar, R. Pelaez-Aguilera, and J. Su. Querying mmoving objects with unvertainty. In *IEEE semiannual Vehicular Technology Conference*, 2003.

[25] H. Mokhtar and J. Su. Universal trajectory queries for moving object databases. In *mdm*, pages 133–144, Berkeley California USA, Jan. 2004.

[26] H. Mokhtar, J. Su, and O. Ibarra. On moving object queries. In *Proc. ACM Symp. on Principles of Database Systems*, 2002.

[27] C. H. Papadimitriou, D. Suciu, and V. Vianu. Topological queries in spatial databases. In *Proc. ACM Symp. on Principles of Database Systems*, 1996.

[28] D. Pfoser and C. Jensen. Capturing the uncertainty of moving-object representations. In R. H. Güting, D. Papadias, and F. H. Lochovsky, editors, *Advances in Spatial Databases, 6th International Symposium, SSD'99, Hong*

Kong, China, July 20-23, 1999, Proceedings, volume 1651 of *LNCS*, pages 111–132. Springer, 1999.

[29] D. Pfoser, C. Jensen, and Y. Theodoridis. Novel approaches to the indexing of moving object trajectories. In *Proc. Int. Conf. on Very Large Data Bases*, 2000.

[30] D. Pfoser and N. Tryfona. Capturing fuzziness and uncertainty of spatiotemporal objects. In *ADBIS*, pages 112–126, 2001.

[31] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. *Journal of Symbolic Computation*, 13:255–352, 1992.

[32] S. Saltenis, C. Jensen, S. Leutenegger, and M. Lopez. Indexing the positions of continuously moving objects. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2000.

[33] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proc. Int. Conf. on Data Engineering*, 1997.

[34] Z. Song and N. Roussopoulos. k-nearest neighbor search for moving query point. In *Proc. Int. Sym. on Spatial and Tempporal Databases*, pages 79–96, 2001.

[35] J. Su, H. Xu, and O. Ibarra. Moving objects: Logical relationships and queries. In *Proc. Int. Sym. on Spatial and Tempporal Databases*, pages 3–19, 2001.

[36] Y. Tao and D. Papadias. MV3R-tree: A spatio-temporal access method for timestamp and interval queries. In *Proc. Int. Conf. on Very Large Data Bases*, 2001.

[37] G. Trajcevski, O. Wolfson, F. Zhang, and S. Chamberlain. The geometry of uncertainty in moving objects databases. In *Advances in Database Technology - EDBT 2002, 8th Int. Conf. on Extending Database Technology*, pages 233–250. Springer, March 2002.

[38] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving objects databases: issues and solutions. In *Proc. Int. Conf. on Statistical and Scientific Database Management*, pages 111–122, 1998.

Session 7: Multidimensional Problems

Mining The Most General Multidimensional Summarization of "Probable Groups" in Data Warehouses*

Hui Yu[1] Jian Pei[2] Shiwei Tang[1] Dongqing Yang[3]

[1] National Laboratory on Machine Perception, Peking University, Beijing, China, yuhui@db.pku.edu.cn, tsw@pku.edu.cn

[2] Simon Fraser University, Burnaby, BC, Canada, jpei@cs.sfu.ca

[3] Peking University, Beijing, China, dqyang@pku.edu.cn

Abstract

Data summarization is an important data analysis task in data warehousing and online analytic processing. In this paper, we consider a novel type of summarization queries, probable group queries, *such as "What are the groups of patients that have a* 50% *or more opportunity to get lung cancer than the average?" An aggregate cell satisfying the requirement is called a* probable group. *To make the answer succinct and effective, we propose that only the most general probable groups should be mined. For example, if both groups (smoking, drinking) and (smoking, *) are probable, then the former groups should not be returned. The problem of mining the most general probable groups is challenging since the probable groups can be widely scattered in the cube lattice, and do not present any monotonicity in group containment order. We extend the state-of-the-art BUC algorithm to tackle the problem, and develop techniques and heuristics to speed up the search. An extensive performance study is reported to illustrate the effect of our approach.*

1 Introduction

Data summarization is an important data analysis task in data warehousing and online analytic processing. For example, an insurance company may want to summarize the common features of low-risk customers based on a data warehouse of customers' claims. High accuracy and good understandability are the major requirements for high quality summarization.

Summarization from large databases, including multidimensional databases and data warehouses, has been studied extensively in previous work. For example, as a major research field in machine learning and data mining, accurate classification has been investigated intensively. A classifier for a target class can be viewed as the summarization. Accurate classifiers with good understandability, such as decision trees [16] and Bayesian networks [6], are often used as summarization of concepts in data analysis. As another example, the attribute-oriented induction [4] method is one of the pioneering database-oriented methods for concept summarization and generalization.

Although previous studies developed effective and efficient methods for summarization, most of them only work for *large* classes. That is, the previous methods implicitly or explicitly assume that the data (e.g., the training data set or the base table in a data warehouse) contains sufficient attributes and enough instances to support the summarization of the target class. However, this assumption may not be honored in some applications.

Example 1 (Motivation) In practice, more often than not a minor class may not be accurately characterized using the available attributes and cases. For example, although it is well known that smoking may lead to lung cancer, fortunately, more than 90% of smokers will not end up getting lung cancer. Generally, lung cancer happens in less than 0.1% of average population. In other words, lung cancer patients form a minor class. The available attributes in consensus data may not be sufficient to characterize the class of lung cancer patients. Therefore, it is often impossible to build an accurate classification model for lung cancer patients on consensus data sets. Instead, it would be more practical to identify the combinations of attributes, such as "smoking" and "family history of lung cancer", that have a much higher probability to lead to lung cancer than the average cases. For example, the statistics identifying the high-risk groups that have 50% or more opportunities to get lung cancer than the average might be very interesting and helpful in health-informatics research.

As another example, in security-informatics, it is gener-

*This research is supported in part by National Natural Science Foundations of China Grant NSFC 60473072, NSERC Grant RGPIN312194-05, NSF (US) Grant IIS-0308001, a President's Research Grant and an Endowed Research Fellowship Award in Simon Fraser University. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

ally very hard, if not impossible at all, to construct an accurate model for terrorists, since the class terrorists is a very minor class in population. Instead, it is more practical to identify the suspicious groups and then follow-up investigations can be conducted. ■

From the above example, we obtain the motivating observations as follows. There are real applications where the task is to summarize some minor classes that might not be accurately characterized using the available attributes and instances. In such cases, it is often useful to query the groups of instances that have a much higher probability to belong to the target minor classes. Such groups are called *probable groups*.

In this paper, we tackle the problem of *multidimensional summarization of probable groups in data warehouses*, and make the following contributions.

- *We identify a novel type of data summarization queries – probable group queries*. We illustrate that the probable group summarization queries are useful for summarization of minor classes. We also show that the problem of multidimensional summarization of probable groups in data warehouses is challenging since the probable groups may be widely scattered in the data cube lattice as the search space, and they do not present any monotonicity in group containment order.
- *We propose mining the most general multidimensional summarization*. We show that finding all probable groups can be ineffective and computational costly. Instead, we propose mining the most general probable groups as the succinct summarization.
- *We develop efficient algorithms*. We extend the sate-of-the-art cubing algorithm BUC [3] to compute all the most general probable groups. To make the mining more efficient, we further develop a heuristic dynamic-ordering method with smart techniques to prune unpromising recursive search. The new method is up to 3 times faster than the simple extension of BUC.
- *We report an extensive performance study*. The experimental results strongly suggest that our approach is efficient and scalable.

The rest of the paper is organized as follows. The problem is defined in Section 2. We develop algorithms for the problem in Section 3. An extensive performance study is presented in Section 4. We review related work in Section 5. The paper is concluded in Section 6.

2 Problem Description

In this section, we first introduce the preliminaries. Then, we present the probable group queries. Last, we examine how such queries should be answered effectively.

2.1 Preliminaries

Consider a *base table* $B = (D_1, \ldots, D_n, C)$, where D_1, $\ldots$, D_n are the n *dimensions* and C is the attribute of *class labels*. We assume that all dimensions are in categorical domains. For any tuple t, the value of t on attribute A is denoted by $t.A$.

An (*aggregate*) *cell* is a tuple $w = (w_1, \ldots, w_n)$, where $w_i \in D_i \cup \{*\}$ $(1 \leq i \leq n)$. When $w_i = *$, the dimension D_i is generalized in w. That is, $*$ matches any value in a dimension. The *cover* of an aggregate cell w, denoted by $cov(w)$, is the set of tuples in B that have the same values as w in all dimensions that $w_i \neq *$. That is,

$$cov(w) = \{t | (t \in B) \wedge (t.D_i = w.D_i \text{ for any } w.D_i \neq *)\}.$$

A cell w is called a *base cell* if for any dimension D_i, $w.D_i \neq *$. A base cell is a group-by of all dimensions in the base table.

For aggregate cells w_1 and w_2, w_1 is an *ancestor* of w_2 and w_2 is a *descendant* of w_1, denoted by $w_1 \succ w_2$, provided (1) for every dimension D_i such that $w_1.D_i \neq *$, $w_2.D_i = w_1.D_i$; and (2) there exists some dimension D_{i_0} such that $w_1.D_{i_0} = *$ and $w_2.D_{i_0} \neq *$. Particularly, if w_2 is a descendent of w_1 and agrees with w_1 on $(n-1)$ dimensions, then w_1 is called a *parent cell* of w_2, and w_2 is a *child cell* of w_1. It is easy to show the following.

Lemma 1 (Cover containment [12]) *For any cells w_1 and w_2 such that $w_1 \succ w_2$, $cov(w_1) \supseteq cov(w_2)$.* ■

However, the reverse direction of Lemma 1 is not true. That is, generally, we cannot derive $w_1 \succ w_2$ based on the fact $cov(w_1) \supseteq cov(w_2)$ [12].

2.2 Probable Group Queries

For a given *target class* $c \in C$ and an aggregate cell w, the *probability* of c in w, denoted by $prob(w, c)$, is the ratio of tuples in $cov(w)$ that belong to class c. That is,

$$prob(w, c) = \frac{|\{t | (t \in cov(w)) \wedge (t.C = c)\}|}{|cov(w)|}.$$

When the target class c is fixed and clear from context, we omit c and write $prob(w, c)$ as $prob(w)$.

An aggregate cell w is called a *probable group* or a *probable cell* provided that $prob(w, c) \geq min\_prob$, where c is the target class and min_prob is the *minimum probability threshold* specified by a user.

Problem definition 1 (Probable group queries) *Given a base table, a target class and a minimum probability threshold, a* probable group query *is to retrieve the complete set of probable groups.* ■

A	B	C	# tuples	# tuples in P	$prob$
a_1	b_1	c_1	7	1	14.29%
a_1	b_1	c_2	9	2	22.22%
a_1	b_2	c_1	4	1	25.00%
a_1	b_2	c_2	7	3	42.86%
a_1	b_3	c_1	10	2	20.00%
a_1	b_3	c_2	8	3	37.50%
a_2	b_1	c_1	5	0	0.00%
a_2	b_1	c_2	11	1	9.09%
a_2	b_2	c_1	7	2	28.57%
a_2	b_2	c_2	15	3	20.00%
a_2	b_3	c_1	4	0	0.00%
a_2	b_3	c_2	12	3	25.00%

Table 1. The base table as our running example.

As shown in Example 1, probable group queries are useful in summarization of minor classes, such as "*What are the groups of patients that have a* 50% *or more opportunity to get lung cancer than the average?*"

Now, the problem becomes searching all probable groups in a data warehouse. A nice property of data warehouse is that all aggregate cells in data warehouse can be organized in a lattice (called *cube lattice*) by the cell cover containment order [9, 12].

Example 2 (Cube lattice) Consider the base table B in Table 1 as our running example. The table has 3 dimensions, namely $A = \{a_1, a_2\}$, $B = \{b_1, b_2, b_3\}$, and $C = \{c_1, c_2, c_3\}$. The number of tuples in every group-by on dimensions A, B and C is also shown in the table (column "# tuples"). Let class P be the target minor class. The number of tuples of class P in every group-by is also shown in the column "# tuples in P". $prob(w, P)$ is also shown for every base cell w.

The set of all possible aggregate cells has $|A \cup \{*\}| \cdot |B \cup \{*\}| \cdot |C \cup \{*\}| = 3 \times 4 \times 3 = 36$ cells. The aggregate cells form a lattice as shown in Figure 1.

Suppose we are interested in the aggregate cells that have a ratio of 25% or up. Those aggregate cells are highlighted in Figure 1. There are in total 13 probable groups (cells). ■

Probable cells are scattered in the cube lattice, as demonstrated in Figure 1. If the probable cells have some monotonic properties in the cube lattice, the search can be facilitated substantially. Unfortunately, probable cells do not carry such a nice property.

Example 3 (Probable cells have no monotonic property) For aggregate cells $w_1 = (a_1, b_2, c_1)$, $w_2 = (a_1, *, c_1)$, and $w_3 = (a_1, *, *)$ in Figure 1, $w_1 \prec w_2 \prec w_3$. As shown in the figure, w_1 and w_3 are probable cells, but w_2 is not. Therefore, probable cells are not monotonic. That is, a probable cell w does not imply that the ancestors or the descendants of w must be probable cells. ■

2.3 Most General Probable Cells: Succinct Summarization

Although probable cells are not monotonic, as shown in Example 3, fortunately, they have a weak monotonic property as follows.

Lemma 2 (Weak monotonicity) *If w is a probable cell, then at least one child of w must also be a probable cell.*
Proof sketch. Let w' be a child cell of w such that $prob(w')$ is the maximum among all children cells of w. It can be shown that $prob(w) \leq prob(w')$. Since w is a probable cell, w' is also a probable cell. ■

Example 4 (Weak monotonicity) It is easy to verify that, in Figure 1, every probable cell has at least a child that is also a probable cells. In fact, for any probable cell c that is not a base cell, there is a path from some base probable cell to c such that each cell on the path is a probable cell. ■

The weak monotonicity gives us two important hints.

- *All probable cells stem from base probable cells*. In other words, although there can be many probable cells in a data warehouse, the base cells that have much higher ratio of the target class enable the more general aggregate probable cells. They are the "roots" of those probable cells.
- *The most general probable cells summarize the probable cells*. For any probable cell a, if it has some ancestor cell that is also a probable cell, it still can be generalized. A cell is *most general* if every ancestor cell of it is not a probable cell. The set of most general probable cells describe the most general extent of probable cells. Each probable cell is either most general, or is summarized by some most general probable cell.

Based on the above discussion, we can use the set of base probable cells and the set of most general probable cells to succinctly summarize a minor class.

Example 5 (Most general probable cells) In our running example, there are in total 13 probable cells. 5 of them are base probable cells. There are 3 most general probable cells, namely $(a_1, *, *)$, $(*, b_2, *)$ and $(*, b_3, c_2)$. In other words, only $\frac{3}{13} = 23.08\%$ of probable cells are most general, and another $\frac{5}{13} = 38.46\%$ of probable cells are base cells. If only the base probable cells and the most general probable

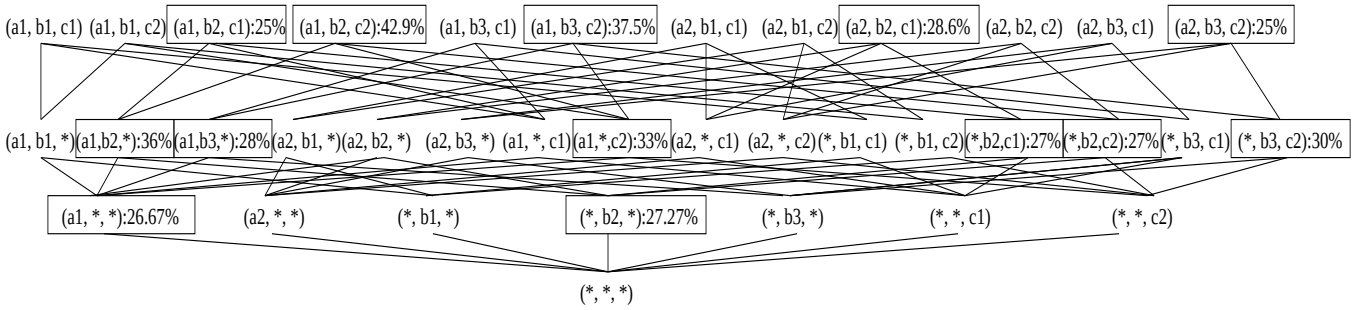


Figure 1. The cube lattice.

cells are used for the succinct summarization, we only need to record 8 probable cells, or $\frac{8}{13} = 61.54\%$ of all probable cells. There is a considerable saving.

As shown in our experimental results, using the base probable cells and the most general probable cells can achieve good saving in summarizing probable cells. ■

Clearly, the set of base probable cells can be computed as the group-by on all dimensions. Since the dimensions are categorical, we can use counting sort[1] to compute them efficient. As will be shown later, computing the set of base probable cells can be a byproduct of computing the set of most general probable cells.

Now, the problem becomes whether we can compute the set of most general probable cells efficiently. In the rest of the paper, we will focus on this issue.

Problem definition 2 (Succinct Summarization) *Given a base table, a target class, and a minimum probability threshold, the problem of* succinct summarization of the target class *is to compute the complete set of base probable cells and the complete set of most general probable cells.* ■

3 Algorithms

In this section, we first review BUC [3], a state-of-the-art algorithm for computing complete data cubes. Then, we discuss how BUC can be extended to mine the set of most general probable cells. We further develop a heuristic algorithm that can be much faster.

3.1 BUC: Bottom-up Cubing

In [3], Beyer and Ramakrishnan developed algorithm BUC, which computes the complete cube for a given base table, i.e., the complete set of aggregate cells. Extensive performance studies [3, 15] showed that BUC is efficient, scalable and moderate in main memory usage.

[1]According to Knuth, counting sort was invented by H.H. Seward in 1954. It is explained in many text books on algorithms, such as [5].

BUC conducts bottom-up computation and can use the monotonic iceberg conditions to prune. To compute a data cube on a base table $T(A, B, C, D)$, BUC first partitions the table according to dimension A, i.e., computing group-bys $(A, *, *, *)$. Then, BUC recursively searches the partition of $cov(a, *, *, *)$, where $a \in A$, and computes the descendant aggregate cells in depth-first search manner, such as $(a, b_1, *, *)$, $(a, b_2, *, *)$, and so on. The computation order is summarized in Figure 2. It also employs counting sort to make partitioning and group-by operations efficient.

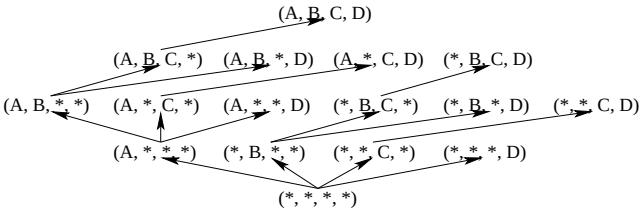


Figure 2. Bottom-up computation in BUC.

BUC can also efficiently incorporate monotonic conditions to compute iceberg cubes. A monotonic condition says that if an aggregate cell fails an iceberg condition, any descendants of it must also fail. If an aggregate cell $(a, *, *, *)$ fails the monotonic iceberg condition, any descendant of it, such as $(a, b, *, *)$, $(a, *, c, *)$ must also fail the condition and thus does not need to be computed in the depth-first search of BUC.

3.2 eBUC: Extending BUC to Mine Most General Probable Cells

Although BUC is efficient to compute the complete data cube, it cannot be directly used to compute the most general probable cells – it cannot use the weak monotonicity of probable cells to prune in a depth-first search. Here, we propose eBUC (for extended BUC), an extension of BUC to use the weak monotonicity in the mining.

The central idea of eBUC is the following observation.

Theorem 1 *An aggregate cell w is a probable cell only if w is a base probable cell or it is an ancestor of some base probable cell.*

Proof sketch. The theorem can be proved by induction on the number of $*$-dimensions in w. Lemma 2 can be applied repeatedly in the induction. ∎

eBuc conducts depth-first search just like BUC. At the beginning of eBUC, by sorting all tuples in the base table using counting sort, eBUC computes the complete set of base cells as a byproduct. It stores those base cells that are probable.

During the rest of the depth-first search, when a new aggregate cell w is encountered, eBUC "looks ahead". That is, it checks whether w is an ancestor of some base probable cells. If not, then following Theorem 1, w cannot be a probable cell. Moreover, any descendant of w cannot be an ancestor of a base probable cell, either. Thus, the recursive search starting at w cannot find any probable cells and thus can be pruned.

When the search encounters a probable cell w, it does not need to search any descendants of w, since they cannot be most general. w is stored and checked after the search. If w is not a descendant of any other probable cells encountered by the search, then w is one of the most general probable cells.

Example 6 (Extended BUC) Let us run eBUC on the running example (Table 1). The search is shown in Figure 3. Only the cells connect by a directed edge are searched. The isolated cells are not searched.

eBUC starts from the most general cell $(*, *, *)$. It is not a probable cell, but it is an ancestor of some base probable cells. Thus, eBUC searches its children recursively in depth-first manner. The children are sorted in the dimensions order A-B-C, and within each dimension, the alphabetical order is used.

The first child, $(a_1, *, *)$, is probable. Thus, no descendants of $(a_1, *, *)$ are searched.

The second child, $(a_2, *, *)$, is not a probable cell, but it is an ancestor of base probable cells (a_2, b_2, c_1) and (a_2, b_3, c_2). Thus, eBUC recursively searches its children. The first child, $(a_2, b_1, *)$, is not a probable cell, and it is not an ancestor of any base probable cells. Thus, as suggested by Theorem 1, the search of $(a_2, b_1, *)$ as well as its descendants can be pruned. eBUC moves to the sibling of $(a_2, b_1, *)$ and search recursively.

The rest of the search is conducted similarly. Limited by space, we omit the details here.

After the search, eBUC checks all the probable cells encountered. For example, although probable cells (a_2, b_2, c_1) and (a_2, b_3, c_2) are encountered by eBUC, they are not the most general since they are descendants of probable cells $(*, b_2, *)$ and $(*, b_3, c_2)$, respectively, and thus will not be output.

As shown in Figure 3, eBUC can find the complete set of most general probable cells. ∎

From Example 6, we can see that the most general probable cells and the weak monotonicity of probable cells can prune the search substantially. In this running example, only 17 of the 36 aggregate cells are searched. In other words, summarization takes only $\frac{17}{36} = 47.22\%$ of the cost of computing the complete cube.

3.3 DYNO: Heuristic Search by Dynamic Ordering

Algorithm eBUC shows good progress on mining the most general probable cells. It can be further improved based on the following two observations.

- In depth-first search, when an aggregate cell has multiple children to be searched, the search from the left-most child covers the largest number of descendant cells. The search from a child cell always covers more descendant cells than that from its right sibling. If a child cell is probable, then all its descendants do not need to be searched. Thus, if we can order the cells dynamically such that the more promising a cell or its descendants are probable, the more left the cell is put, then sharper pruning is likely accomplished.

- As indicated by Theorem 1, only aggregate cells that are ancestors of some base probable cells should be considered. Thus, when expanding the search to children cells, only the dimension values that appear in some base probable cells that are descendants of the current cell should be used to expand the children of the current cell. All other children of the current cell are not promising.

Based on the above two observations, we develop algorithm DYNO (for DYNamic Ordering). DYNO follows the framework of eBUC and has the major improvements as follows.

In the depth-first search, if the current cell w is not a probable cell but is an ancestor of some base probable cell, then DYNO dynamically generates and orders the children cells.

DYNO does not expand all children cells of w. Instead, DYNO collects all base probable cells that are descendants of w. Only dimension values of those base probable cells are used to assemble children cells of w. The correctness of this improvement follows the second observation above. Moreover, to guarantee the completeness of the search and avoid searching a cell more than once in the depth-first

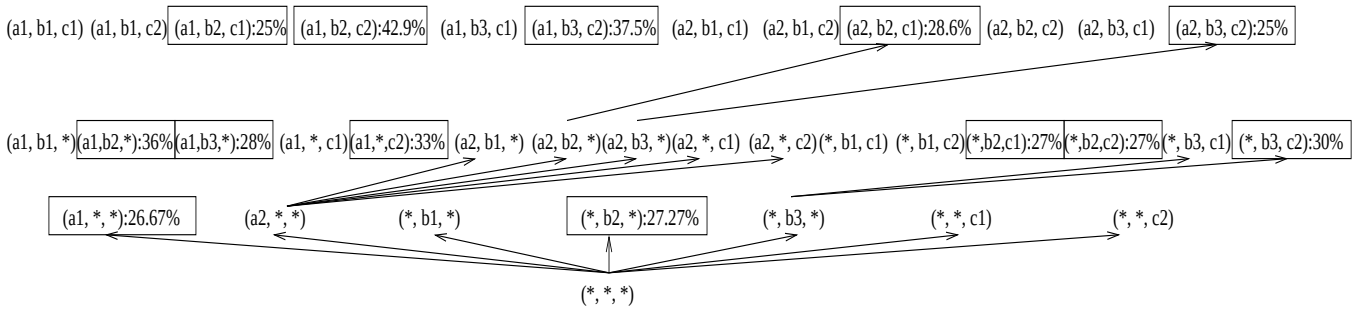


Figure 3. Search using eBUC.

search, DYNO joins w with the right siblings of w to generate its children cells, where the join is defined as follows.

For a pair of sibling cells w_1 and w_2, two cases may arise.

- w_1 and w_2 agree on all dimensions except for one dimension D. That is, $w_1.D \neq w_2.D$ and both do not take $*$ on dimension D. In this case, the join is not defined. In other words, w_1 and w_2 cannot be joined.
- w_1 and w_2 agree on all dimensions except for two dimensions D and D'. That is, $w_1.D = *$, $w_2.D \neq *$, $w_1.D' \neq *$ and $w_2.D' = *$. In this case, the join is defined as w_3 such that w_3 take values as its parent except for dimensions D and D', $w_3.D = w_2.D$ and $w_3.D' = w_1.D'$.

The current cell w may have multiple children. Then, according to the first observation discussed above, we should search them in the order of likelihood that they are probable cells. Heuristically, we can search them in their probability descending order – the higher the probability, the better chance that it or some of its descendants are a probable cell.

We need to show that the above dynamic generation and ordering of children retains the completeness and non-redundancy of depth-first search.

Theorem 2 (Dynamic generation and ordering) *A depth-first search with the dynamic generation and ordering of children cells visits each aggregate cell once and only once if no pruning is taken.*

Proof sketch. The theorem can be proved by induction on the number of non-$*$ dimensions in aggregate cells. For each cell w, it can be shown that w will be generated once and only once. Limited by space, we only show the essential idea here. ∎

Example 7 (DYNO) Let us apply algorithm DYNO on our running example. The search is illustrated in Figure 4.

DYNO starts from the most general cell $(*, *, *)$. Since it is not a probable cell, but it is an ancestor of some base probable cells, we need to search its children.

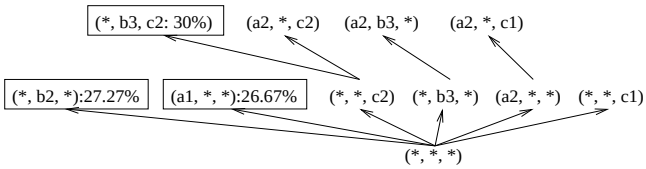


Figure 4. Search using DYNO.

When generating the children cells of $(*, *, *)$, DYNO notices that b_1 never appears in any base probable cell. Thus, any aggregate cells having b_1 cannot be a probable cell. Although $(*, b_1, *)$ is a child of $(*, *, *)$, it is unpromising and thus should not be generated. The children cells of $(*, *, *)$ generated by DYNO are $(*, b_2, *)$, $(a_1, *, *)$, $(*, *, c_2)$, $(*, b_3, *)$, $(a_2, *, *)$, and $(*, *, c_1)$, in the probability descending order.

$(a_1, *, *)$ and $(*, b_2, *)$ are probable cells. They are stored for postprocessing. The descendants of the two cells will not be searched.

DYNO recursively searches cell $(*, *, c_2)$. By joining the right siblings, two children cells are generated, namely $(*, b_3, c_2)$ and $(a_2, *, c_2)$. The mining can be conducted recursively. Limited by space, we omit the details here.

After the search, for each probable cell w encountered in the search, DYNO checks whether w is a descendant of some other encountered probable cells. If not, then cell w is output as a most general probable cell.

It can be verified that DYNO can find the three most general probable cells. ∎

From the above example, we can see that DYNO can find the complete set of most general probable cells. Moreover, DYNO searches much fewer cells than eBUC. In this example, DYNO searches 11 cells, while eBUC searches 17 cells. DYNO searches $\frac{6}{17} = 35.29\%$ less cells than eBUC. Our experimental results show that DYNO can search over 50% less cells than eBUC. This is a major saving in the mining. To summarize, algorithm DYNO is shown in Figure 5.

Algorithm DYNO
Input: a base table B, a target class c, and a minimum probability threshold δ;
Output: the set of base probable cells and the set of most general probable cells;
Method:

1. sort tuples in B, compute and output the base probable cells, also compute $prob(*, \ldots, *)$;
2. let $W = \emptyset$;
3. conduct depth-first search from cell $(*, \ldots, *)$, for each current cell w, do
4. if w is not an ancestor of any base probable cell then return;
5. if $prob(w) \geq \delta$ then $W = W \cup \{w\}$, return;
6. generate children of w by joining w with the right siblings of w, using only the dimension values that appear in descendant base probable cells of w
7. compute probability for children cells;
8. sort the children of w in the probability descending order;
9. search the children recursively in depth-first manner;

// Postprocessing
10. remove cells w from W such that w has an ancestor w' in W;
11. output W;

Figure 5. Algorithm DYNO.

4 Experimental Results

We conducted extensive experiments using synthetic data sets. The results are consistent. Limited by space, we only reported some results in this section.

All the algorithms are implemented using Microsoft Visual C++ V6.0. The experiments are conducted on a PC with a P4 1.5G Hz CPU and 512 MB main memory. The operating system is Microsoft Windows XP.

By default, a base table has 10 dimensions. The cardinality of each dimension is 100. There are 100 thousand tuples in the base table. Each tuple is a base cell with a population and a probability of the target class. The probability of the target class in base cells follows the Half-Normal Distribution in $[0, 1]$, i.e., a normal distribution with mean 0 and standard deviation $\frac{1}{\theta}$ limited to the domain $[0, 1]$. In the results reported in this section, we set $\theta = 1$.

First of all, it is interesting to examine the change of the number of probable cells and the number of most general probable cells with respect to the probable threshold, which is shown in Figure 6. As the minimum probability threshold goes down, the number of probable cells keeps growing. However, the number of most general probable cells does not monotonically change. When the minimum probability threshold is high, there are only a small number of probable cells, and the number of most general probable cells is also small. As the minimum probability threshold goes down, both the number of probable cells and the number of most general probable cells increase. When the minimum probability threshold is lower than 50% in our experiments, there are many probable cells. They can be summarized by some quite general probable cells. The strong capability of high level aggregate cells to summarize the low level cells brings down the number of most general aggregate cells. In the extreme case, when the most general cell in the cube, $(*, \ldots, *)$, is probable, there is only one most general probable cell.

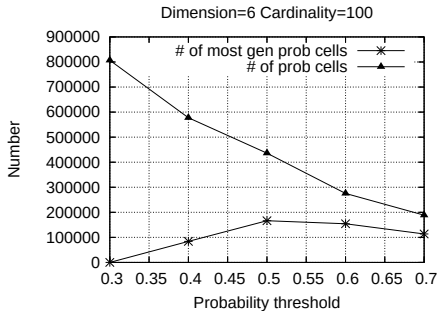


Figure 6. Number of probable cells with respect to minimum probability threshold.

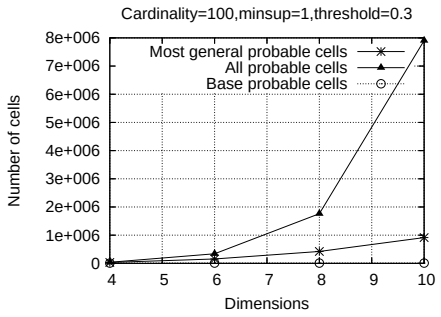


Figure 7. The number of probable cells with respect to dimensionality.

The number of probable cells and the number of most general probable cells also increase as the dimensionality increases, as shown in Figure 7. However, the number of most general probable cells has a much more moderate increase rate.

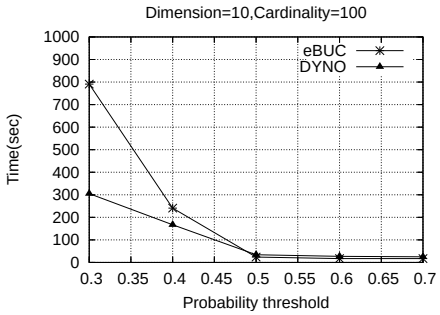


Figure 8. The scalability with respect to minimum probability threshold.

In Figure 8, we tested the scalability of eBUC and DYNO with respect to the probability threshold. When the probability threshold is set high, the number of probable cells and the number of most general probable cells are small. Thus, both algorithms are fast and the difference between the two algorithms is minor. However, when the probability threshold is low, there can be many probable cells. DYNO has a much better scalability than eBUC.

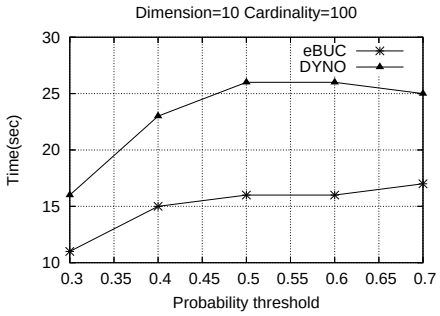


Figure 9. The depth-first search runtime with respect to minimum probability threshold.

The runtime of both DYNO and eBUC can be divided into two parts: the time for depth-first search and the time for postprocessing. An interesting observation is that the depth-first searches in DYNO and eBUC only take a small part in the total runtime. The runtime for depth-first search is shown in Figure 9. As can be seen, when the minimum probability threshold is low and the number of most general probable cells decreases, DYNO and eBUC become more efficient in the depth-first search. In other words, the curves of depth-first search runtime of DYNO and eBUC in Figures 9 are consistent with the curve of number of most general probable cells in Figure 6. In terms of search time per cell, eBUC is shorter than DYNO since DYNO needs to collect more information than eBUC. However, the major advantage of DYNO is that it generates much less candidate cells than eBUC, which makes the postprocessing of DYNO clearly faster.

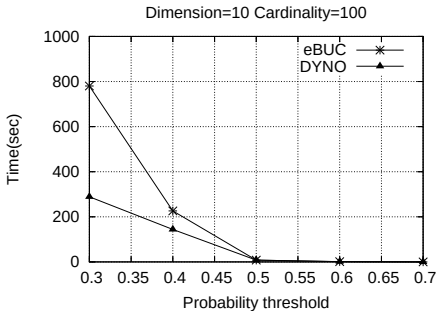


Figure 10. The postprocessing runtime with respect to minimum probability threshold.

Figure 10 shows the postprocessing runtime. Both DYNO and eBUC use the same method in postprocessing to remove the non-most general probable cells. Since the heuristic search in DYNO (dynamic generation and ordering of children cells) can effectively reduce the number of probable cells searched, the postprocessing cost in DYNO is substantially smaller than that in eBUC.

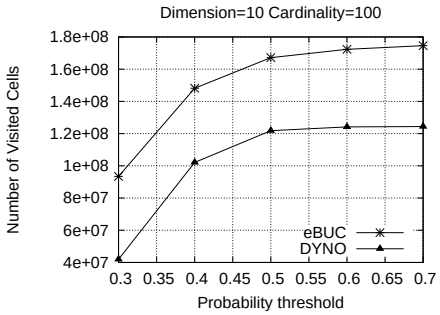


Figure 11. The number of aggregate cells searched with respect to minimum probability threshold.

Figure 11 supports the claim that DYNO visits substantially less aggregate cells in finding the most general probable cells. DYNO also encounters much less probable cells in the depth-first search than eBUC. The numbers of probable cells encountered by DYNO and eBUC, respectively, follow the trends similar to the results in Figure 10. Limited by space, we omit the details here. It shows that the pruning techniques in DYNO are effective.

To test the scalability of our methods, we ranged the di-

mensionality from 4 to 10. The results are shown in Figure 12. DYNO has a better scalability. Moreover, the results are consistent with the number of most general scalable cells shown in Figure 7.

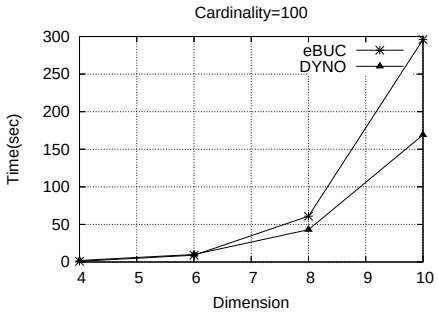


Figure 12. The runtime with respect to dimensionality.

We also tested the runtime of DYNO and eBUC on the number of tuples in the base table. Both are linearly scalable, and DYNO has a better scalability. Limited by space, we omit the details here.

In summary, the extensive experimental results strongly suggest that using the most general aggregate cells can effectively summarize the probable cells. DYNO is an efficient method to compute the most general probable cells.

5 Related Work

The data cube operator [9] is one of the most influential operators in OLAP. Many approaches have been proposed to compute data cubes efficiently from scratch (e.g., [24, 17, 18, 3]). In general, they speed up the cube computation by sharing partitions, sorts, or partial sorts for group-bys with common dimensions.

It is well recognized that the space requirements of data cubes in practice are often huge. Some studies investigate partial materialization of data cubes, e.g., [11, 3]. Example methods to compress data cubes are [19, 20, 12, 13]. Moreover, [1, 2, 21] investigate various approximation methods for data cubes.

There are several major methods on computing (iceberg) cubes. MultiWay [24] is an array-based top-down approach to computing complete data cube. The basic idea is that a high level aggregate cell can be computed from its descendants instead of the base table. To compute a data cube on a base table $T(A, B, C, D)$, MultiWay first scans the base table once and computes group-bys (A, B, C, D), $(*, B, C, D)$, $(*, *, C, D)$, $(*, *, *, D)$ and $(*, *, *, *)$. These group-bys can be computed simultaneously without resorting the tuples in the base table. Once these group-bys are computed, we do not need to scan the base table any more. MultiWay may not be efficient in computing iceberg cubes with monotonic iceberg conditions, since the top-down search cannot use the monotonic iceberg condition to prune.

Fang et al. [7] proposed the concept of iceberg queries and developed some sampling algorithms to answer such queries. An iceber cube is the set of aggregate cells in a cube that satisfy some user-specified condition. Beyer and Ramakrishnan [3] introduced the problem of iceberg cube computation in the spirit of [7] and developed algorithm BUC, which is revisited in Section 3.1. Often, monotonic iceberg conditions are used to prune in the computation of iceberg cubes.

H-cubing [10] uses a hyper-tree data structure called H-tree to compress the base table. Then, the H-tree can be traversed bottom-up to compute iceberg cubes. It also can prune unpromising branches of search using monotonic iceberg conditions. Moreover, a strategy was developed in [10] to use weakened but monotonic conditions to approximate non-monotonic conditions to compute iceberg cubes. The strategies of pushing non-monotonic conditions into bottom-up iceberg cube computation were further improved by Wang et al. [22]. A new strategy, divide-and-approximate, was developed. The general idea is that the weakened but monotonic condition can be made up for each search sub-branch and thus the approximation and pruning power ca be stronger.

In [23], Xin et al. developed Star-Cubing by extending H-tree to Star-Tree and integrating the top-down and bottom-up search strategies. Feng et al. [8] proposed another interesting cubing algorithm, Range Cube, which uses a data structure called range trie to compress data and identify correlation in attribute values. On the other hand, since iceberg cube computation is often expensive in both time and space, parallel and distributed iceberg cube computation has been investigated. For example, Ng et al. [15] studied how to compute iceberg cubes efficiently using PC clusters.

In all the previous studies, either the complete cube or the complete iceberg cube is computed. None of them consider the problem of computing a summarization of the cells that satisfy some user-specified condition. None of them either deal with mining the most general aggregate cells. To the best of our knowledge, this paper is the first one that addresses the issue.

On the other hand, this paper is also related to previous work on concept summarization [4], generalization and learning [14]. However, different from those approaches, we use the most general aggregate cells to summarize probable groups, which have not been discussed in those previous studies.

6 Conclusions

Data summarization is an important data analysis task in data warehousing and online analytic processing. In this paper, we identified a new type of summarization queries, *probable group queries*, and proposed a succinct summarization answer to the queries using the base probable cells and the most general probable cells. The problem of mining the most general probable cells is challenging since the probable cells can be widely scattered in the cube lattice, and do not present any monotonicity in cover containment order. We extended the state-of-the-art BUC algorithm to tackle the problem, and developed techniques and heuristics to speed up the search. An extensive performance study verified that our approach is effective and efficient.

This study raises several interesting problems for future studies. For example, it is interesting to improve the performance of DYNO further, especially reducing the cost of ancestor-descendant checking in the postprocessing. Moreover, summarization and understanding of minor classes are important for data analysis and applications. Theoretical framework as well as practical mining methods should be explored further.

References

[1] D. Barbara and M. Sullivan. Quasi-cubes: Exploiting approximation in multidimensional databases. *SIGMOD Record*, 26:12–17, 1997.

[2] D. Barbara and X. Wu. Using loglinear models to compress datacube. In *WAIM'2000*, pages 311–322, 2000.

[3] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*, pages 359–370, Philadelphia, PA, June 1999.

[4] Y. Cai, N. Cercone, and J. Han. An attribute-oriented approach for learning classification rules from relational databases. In *Proc. 1990 IEEE Int. Conf. Data Engineering (ICDE'90)*, pages 281–288, Los Angeles, CA, Feb. 1990.

[5] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[6] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[7] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*, pages 299–310, New York, NY, Aug. 1998.

[8] Y. Feng, D. Agrawal, A. E. Abbadi, and A. Metwally. Range Cube: Efficient cube computation by exploiting data correlation. In *Proc. 2004 Int. Conf. Data Engineering (ICDE'04)*, pages 658–669, Boston, MA, April 2004.

[9] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational operator generalizing group-by, cross-tab and sub-totals. In *Proc. 1996 Int. Conf. Data Engineering (ICDE'96)*, pages 152–159, New Orleans, Louisiana, Feb. 1996.

[10] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg cubes with complex measures. In *Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'01)*, pages 1–12, Santa Barbara, CA, May 2001.

[11] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'96)*, pages 205–216, Montreal, Canada, June 1996.

[12] L. Lakshmanan, J. Pei, and J. Han. Quotient cube: How to summarize the semantics of a data cube. In *Proc. 2002 Int. Conf. Very Large Data Bases (VLDB'02)*, Hong Kong, China, Aug. 2002.

[13] L.V.S. Lashmanan, J. Pei, and Y. Zhao. QC-Trees: An efficient summary structure for semantic OLAP. In *Proc. 2003 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'03)*, San Diego, California, June 2003.

[14] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[15] Raymond T. Ng, Alan S. Wagner, and Yu Yin. Iceberg-cube computation with PC clusters. In *Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'01)*, Santa Barbara, CA, May 2001.

[16] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[17] K. Ross and D. Srivastava. Fast computation of sparse datacubes. In *Proc. 1997 Int. Conf. Very Large Data Bases (VLDB'97)*, pages 116–125, Athens, Greece, Aug. 1997.

[18] Kenneth A. Ross and Kazi A. Zaman. Optimizing selections over datacubes. In *Statistical and Scientific Database Management*, pages 139–152, 2000.

[19] Jayavel Shanmugasundaram, Usama Fayyad, and P. S. Bradley. Compressed data cubes for olap aggregate query approximation on continuous dimensions. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 223–232, San Diego, California, United States, 1999. ACM Press.

[20] Yannis Sismanis, Nick Roussopoulos, Antonios Deligiannakis, and Yannis Kotidis. Dwarf: Shrinking the petacube. In *Proc. 2002 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'02)*, Madison, Wisconsin, June 2002.

[21] J. S. Vitter, M. Wang, and B. R. Iyer. Data cube approximation and historgrams via wavelets. In *Proc. 1998 Int. Conf. Information and Knowledge Management (CIKM'98)*, pages 96–104, Washington DC, Nov. 1998.

[22] K. Wang, Y. Jiang, J. X. Yu, G. Dong, and J. Han. Pushing aggregate constraints by divide-and-approximate. In *Proc. 2003 Int. Conf. Data Engineering (ICDE'03)*, pages 291–302, Bangalore, India, March 2003.

[23] D. Xin, J. Han, X. Li, and B. W. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In *Proc. 2003 Int. Conf. on Very Large Data Bases (VLDB'02)*, pages 476–487, Berlin, Germany, Sept. 2003.

[24] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97)*, pages 159–170, Tucson, Arizona, May 1997.

Optimizing Multiple Top-K Queries over Joins

Dirk Habich and Wolfgang Lehner
Dresden University of Technology
Database Technology Group
01062 Dresden, Germany
{dirk.habich, lehner}@inf.tu-dresden.de

Alexander Hinneburg
Martin-Luther-University of Halle/Wittenberg
Database and Data Mining Group
06099 Halle, Germany
hinneburg@informatik.uni-halle.de

Abstract

Advanced Data Mining applications require more and more support from relational database engines. Especially clustering applications in high dimensional features space demand a proper support of multiple Top-k queries in order to perform projected clustering. Although some research tackles to problem of optimizing restricted ranking (top-k) queries, there is no solution considering more than one single ranking criterion. This deficit - optimizing multiple Top-k queries over joins - is targeted by this paper from two perspectives. On the one hand, we propose a minimal but quite handy extension of SQL to express multiple top-k queries. On the other hand, we propose an optimized hash join strategy to efficiently execute this type of queries. Extensive experiments conducted in this context show the feasibility of our proposal.

1. Motivation

With the advent of data warehousing concepts, knowledge discovery in general became one of the most prominent database application areas. Many extensions were proposed to better support KDD and warehouse queries and optimize their execution. Before the SQL:1999/SQL:2003 standardization, a top-k query with a single rank could be written as SELECT statement containing an ORDER BY plus a limiting clause like STOP AFTER k ROWS [3] or FETCH FIRST k ROWS ONLY (DB2 dialect) to retrieve the top-k results. The standard introduced an alternative formulation with nested SELECT statements making use of OLAP functions like RANK in combination with the OVER clause.

The extension of the OVER()-clause allows the specification of a column-wise ordering, partitioning and windowing scheme. Positions are computed by three additional aggregation functions RANK(), DENSERANK(), and ROWNUMBER() differing in the semantics of breaking ties. Due to simplicity, we do not further consider ties and refer to the RANK()-operator. The limitation with respect to the first k rows must be indirectly specified in a surrounding query. The following example shows how to state one or more top-k queries within a single SQL query.

```
SELECT x.id, x.pos1, x.pos2
FROM (
    SELECT id,
           RANK() OVER (ORDER BY f1()) AS pos1,
           RANK() OVER (ORDER BY f2()) AS pos2
    FROM R INNER JOIN S ON ...
    WHERE ...) x
WHERE x.pos1 <= :k OR x.pos2 <= :k
```

Computing top-k queries using this SQL extension is based on the principle of ordering the underlying data set with regard to the (usually numeric) ranking criterion and returning only the first k values per column. In the above example, after joining tables R and S, two different ranks are determined for each tuple according to sort criteria functions $f_1()$ and $f_2()$. The restriction to the top-k tuples of both rankings is applied in an surrounding $select$ statement.

Example 1: The concept of multiple top-k queries naturally appears in several relevant data mining and information retrieval applications. Many information retrieval systems employ relevance feedback. The idea is that the system learns iteratively from the users rating of the presented results to improve the retrieval quality. For example the concept of Kim and Chung [12] extends the basic idea of query point movement. Instead of moving the query point based on user feedback towards an assumed ideal query point the extended concept of complex similarity queries allows a set of multiple query points $Q = \{q_1, \ldots, q_n\}$. The top-k result tuples can be defined by a new distance function, which requires that the result tuple is close to at least one of the query points in Q:

$$dist(x, Q) = \min_{q_i \in Q}{}_k\{dist(x, q_i)\},\ 1 \leq i \leq n$$

The distance function can be expressed as a SQL query using n top-k rankings, one for each query point. The combined results of the top-k queries are ordered according to

SID	PID	QUAN.	SALES
1	1	500	1200,00
1	2	300	100,50
...			
10	5	100	50,00

(a) fact table

SID	G	F	EU
1	0.2	0.3	0.2
2	0.1	0	0.05
3	0	0.2	0.1
...			

(b) weighting table

Figure 1. Example for multiple top-k queries over joins

the distance to their nearest query points and the k tuples with the smallest distance are returned.

Example 2: Another application scenario appears in data warehouse environments, where multiple top-k queries over joins are useful. Consider following example, where a fact table holds objects like products or shops and corresponding facts. These informations are either stored in the data warehouse or computed with SQL-statements. The table in figure 1(a) holds some facts for sold products (PID) in shops (SID), like quantity and sales. The objects can now be ranked according to the different facts with regard to weighting factors. Such factors represent the importance of the objects in different contexts and they are used to align raw data and to statistically correct samples. For example, typical weighting factors for shops are the market power with regard to geographic location, e.g. Germany, France and Europe. This weighting factors are typically stored in a dimension table (figure 1(b)). To rank the objects into multiple directions with regard to the multiple facts and multiple weighting factors a join between the fact table and the weighting table is necessary and the result have to be ordered according to multiple ranking functions. In this case, the parameters of the ranking functions come from both relations.

Our Contribution

In this paper we consider a class of ranking functions described by $f(g_1(R.A_1, R.A_2, \ldots), g_2(S.B_1, S.B_2, \ldots))$ where $f(\cdot, \cdot)$ is monotonic in its two input attributes. The functions $g_1()$ and $g_2()$ might be any functions taking inputs from tables R and S respectively. We also consider multiple ranking functions taking only inputs from one relation.

Moreover, it is worth mentioning that without applying very specific optimization strategies the top-k-computation is done by computing the ranks for *all* tuples requiring one sort for each individual ranking criterion. Finally, for applying specific optimization algorithms, the currently available top-k ranking formulations show the problem that the information about the top-k predicate is structurally very far from the ranking declaration implying very sophisticated query graph pattern recognition mechanisms to detect situations in which the query could be optimized.

To soften the two major problems - no direct support of top-k queries in the SQL formulations and no internal optimization algorithms for computing multiple top-k queries simultaneously, we propose the following concepts in this paper:

- First of all, we introduce a small SQL extension of ORDERING SETS to simplify the declaration of multiple rankings. Additionally, we inject LIMIT BY clauses in the ORDERINGS SETS as well as within the already existing OVER-clause.
- We discuss the limitations of existing rank optimizations in the presence of multiple ranks and give a potential extension of an early stop algorithm based on sort-merge joins.
- We finally propose a variation of the well-known hash-join algorithm which considers the presence of multiple top-k columns. This variation outperforms all other join strategies and can be easily integrated into existing database engines.

The rest of the paper is organized as follows: After gleaning related work in the following section, we present our SQL extension for computing multiple top-k queries in section 3. In section 4 we consider simple queries with multiple ranks, but without joins. Thereafter, in section 5, we describe an extension of an early stop algorithm and introduce our proposed extension of a hash-join method considering the existence of rankings. In section 6, we demonstrate the improved efficiency of our algorithms by describing the results of extensive experiments run on a prototypical implementation. The paper closes with a summary and conclusion.

2. Related Work

The goal behind top-k queries is to apply a scoring function on multiple attributes coming from one or multiple tables to select the best k tuples ranked by the function. So far, top-k queries with single ranking function have been intensively studied in the last years of database research. In particular it is worth mentioning that top-k queries have been considered in various contexts.

Carey and Kossmann [3] extended SQL's SELECT statement by a STOP AFTER clause, which limits the cardinality of a query result. The authors showed that this clause especially in combination with ORDER BY leads to significant better query plans and execution times. In the follow up paper [4] they presented extended implementation techniques for the STOP AFTER clause based on range partitioning. Donjerkovic and Ramakrishnan [7] proposed to map a top-k query to a range query with the range $[max, \kappa]$

where κ is chosen in a probabilistic way so that the range contains approximately k tuples. While this and the previous articles focused on orderings based on the column values of a single attribute themself, later papers take also ranking conditions based on multiple attributes, e.g. multi-dimensional metrics, into account. Chaudhuri et. al [1, 5] studied the use of multi-dimensional histograms to evaluate top-k queries with multi-attribute ranking conditions, namely metrics like Maximum, Euclidean and Manhattan. Here a top-k query is mapped to a multi-dimensional range query centered around a given query point. In their work they included experiments with ranking conditions based on up to four attributes. Cheng and Ling [6] proposed an approximative variant of the method of Chaudhuri et al. based on sampling, which scales better to high-dimensional data (up to 100 attribute) and has only a small loss of accuracy. Another approach was taken by Hristides et al. [9], who used multiple materialized views to efficiently answer top-k queries, with ranking conditions based on linear functions of the attributes of a relation. For a given ranking condition the best matching materialized view is selected to approximate the query answer.

None of the above described approaches considered top-k queries in conjunction with joins. Ilyas et. al [10, 11] proposed a new rank join operator producing *single* top-k results progressively during the join results. They consider a set of tables R_1 to R_n, where each tuple in R_i is associated with a local score. The global score is computed according to a function f combining the local scores of the individual tables. In section 5 we give a more detail description, because one of our algorithm extends the rank join approach to evaluate multiple top-k join results. In [11] Ilyas et. al present a rank-aware query optimization framework integrating the rank-join operators into relational query engines. The generation of a rank-aware query plan is done with a probabilistic model for estimating the input cardinality, and cost of the rank-join operators.

In a recent article Slivinskas, Jensen and Snodgrass [13] identified the optimization problem of database queries containing ORDER BY as a very important problem, which has been underestimated in the database community. They propose an extended algebra taking a single ORDER BY and top-k queries into account and give several formal transformation rules for such queries.

However, research so far on top-k queries considers only queries with a single ranking, i.e. sort and limitation condition.

3. SQL Extension for Top-k Queries

This section outlines minimal SQL extensions providing a new concept of computing multiple top-k queries within a single select statement. In a first step, we revise the current state of the art, demonstrate the problems in retrieving multiple top-k database entries and finally introduce the ORDERING SET() and LIMIT BY() concepts from a syntactical as well as a semantic point of view.

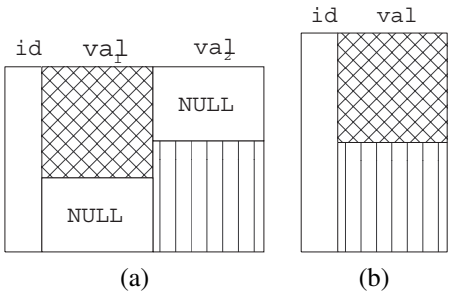


Figure 2. Schematic sketch of the possibilities for returning result of queries with multiple ranking conditions.

3.1. Multiple Top-k Queries

Conceptually there are two possible methods to return multiple top-k results without unnecessary information. Figure 2 illustrates the shape of the resulting tables.

The following query pattern basically extends the basic SQL pattern given in the motivation with the assumption that only the first k values of each sorting criterion have to be considered. This can be achieved by performing an $n-1$-ary outer join to compute the required result (figure 2a).

```
SELECT v1.id ,v1.pos1 ,COALESCE(v1.ind1 ,0) AS ind1 ,
                v2.pos2 ,COALESCE(v2.ind2 ,0) as ind2
FROM (
   (SELECT v1.id, v1.pos1, 1 AS ind1
    FROM (
        SELECT id ,
            RANK() OVER(ORDER BY f1()) AS pos1 ,
        FROM ...
        WHERE ...) u1
    WHERE u1.pos1 <=: k) v1(id, pos1)
    FULL OUTER JOIN
   (SELECT v2.id, v2.pos2, 1 AS ind2
    FROM (
        SELECT id ,
            RANK() OVER(ORDER BY f2()) AS pos2 ,
        FROM ...
        WHERE ...) u2
     WHERE u2.pos2 <=: k) v2(id, pos2)
     ON v1.id = v2.id )
```

Within this query template, each subquery locally computes the individual ranking results which are then 'concatenated' using a full outer join so that the individual ranks

larger than the given k are set to NULL. Additionally, an indicator function *COALESCE* is used to differentiate between natural NULL values and NULL values generated by full outer joining the individual top-k queries. In the worst case, this may yield an extremely sparse table where each tuple holds only one valid rank. So for computing n ranks of size k, we may yield a cardinality between k and $n \cdot k$.

Alternatively (figure 2b), the local results of the individual ranking values can be concatenated vertically by performing an union after computing the local ranks. For the running example, the corresponding query pattern might look like the following:

```
SELECT u1.id, '1' AS indicator, u1.pos1 AS pos
FROM (
    SELECT id,
           RANK() OVER (ORDER BY f1()) AS pos1,
    FROM ...
    WHERE ...) u1
WHERE u1.pos1 <= :k
UNION
SELECT u1.id, '2' AS indicator, u2.pos2 AS pos
FROM (
    SELECT id,
           RANK() OVER (ORDER BY f2()) AS pos2,
    FROM ...
    WHERE ...) u2
WHERE u2.pos2 <= :k
```

An additional indicator column denotes the local result set. This solution is perfect if there is almost no overlap in the result set implying that a single row appears only once within the top-k values with regard to a single ranking.

Comparing both alternatives from a query formulation and query optimization perspective leads to the following observation. The individual subqueries are computed locally and combined in a subsequent step, which is either a union or a full outer join so that applying sophisticated rank-operators eventually computing multiple top-k results becomes extremely difficult. Additionally, the query structures of both variants are inadequate to serve as language expressions because of the huge statements necessary to express the same pattern and repetitive computation of the (potentially complex) table expressions in the FROM clauses.

To put it into a nutshell, it is clear that SQL does not adequately support multiple orderings in combination with a limitation of the output stream either for vertically or horizontally constructed result sets. The query expressions are extremely voluminous. Additionally, it is extremely difficult for the rewrite system inside of the database engine to detect these query patterns and to apply specific optimization techniques.

3.2. The ORDERING SET-Operator

To weaken the problems of multiple orderings and limiting the output stream, we propose a much simpler language construct, namely ORDER BY ORDERING SET, which operates quite similar to the GROUPING SET-operator and therefore fits nicely into the set of SQL extensions.

The ORDERING SET()-operator (as an extension of the ORDER BY-clause) denotes that the same data is sorted according to multiple ordering criteria and may be seen quite similar to the construct of a GROUPING SET()-operator, which is an extension of the simple GROUP BY-clause. Additionally, the individual ordering criteria may be extended with a LIMIT BY-parameter to restrict the number of rows for the particular ordering criterion. The ORDERING SET()-operator delivers the tuples of a table in the order according to the actual ranking criterion. When all tuples of the table are returned or the limit is reached the next ranking is processed.

```
SELECT ..., f1(), f2()
FROM ...
WHERE ...
ORDER BY ORDERING SET(
      (f1() DESC LIMIT BY :k),
      (f2() DESC LIMIT BY :k) )
```

To illustrate the ORDERING SET()-operator, we refer to the example in figure 3, which shows how to compute the first 3 rows with the highest values in $f_1()$ and $f_2()$.

The result (right table) of the ORDERING SET()-operator, which ranks first according to column f_1 and then f_2 with the limit $k = 3$. The horizontal line in the right table indicates when the second ordering starts.

Like the normal ORDER BY expression, one single ORDERING SET expression can exhibit multiple sorting criteria including ASC and DESC annotations to distinguish between ascending and descending ordering. A single ORDERING SET expression is equivalent to a normal ORDER BY expression, e.g.

```
ORDER BY ORDERING SET( f1() ASC, f2() DESC)
≡
ORDER BY f1() ASC, f2() DESC
```

Although in the general case, the value for k may be different for each individual ranking, in many applications k

ID	f_1	f_2
1	4	11
2	3	22
3	2	33
4	1	44

⇒

ID	f_1	f_2
1	4	11
2	3	22
3	2	33
4	1	44
3	2	33
2	3	22

Figure 3. Example of a 2-ary ranking of size 3

ID	$f_1()$	$f_2()$
1	44	11
2	33	22
3	22	33
4	11	44

⇒

ID	ORDERING($f_1()$)	ORDERING($f_2()$ DESC)	ORDERING($f_1()$ DESC, $f_2()$)
4	1	0	0
3	1	0	0
4	0	1	0
3	0	1	0
1	0	0	1
2	0	0	1

Figure 4. Example for the ORDERING function

will be the same for all sorting criteria. In this case, we allow an alternative global limitation for the ORDERING SETS as a shortcut, e.g.

```
ORDER BY ORDERING SET( (f1() LIMIT BY :k),
                       (f2() LIMIT BY :k) )
≡
ORDER BY ORDERING SET( ( f1() ) ,( f2() ) )
                    LIMIT BY :k
```

Like the GROUPING()-function for the GROUPING SET-extension we introduce the ORDERING()-function, which indicates to which ordering set a tuple in the result set belongs to. The function returns 1 if the current row was sorted according to the given sorting criterion. Thus, ORDERING(x) returns 1 if the current row was sorted according to expression x. In case the ordering set is defined over multiple sorting criteria (e.g. $f_1()$ DESC, $f_2()$ ASC) the ORDERING()-function takes also a list of expressions.

To illustrate the semantics in more detail, we consider the following ORDERING SET()-clause returning the first three rows of each ranking:

```
SELECT ...,
       ORDERING(f1()),
       ORDERING(f2() DESC),
       ORDERING(f1() DESC, f2())
FROM ...
WHERE ...
ORDER BY ORDERING SET( (f1()), (f2() DESC),
    (f1() DESC, f2()) ) LIMIT BY 2;
```

Figure 4 illustrates the identification of the individual ordering set membership for each row.

In case the ranking criteria are compatible with each other, e.g. only one expression is used for each ordering set, the result table can be explicitly transformed into the schema shown in figure 2b with the help of the ORDERING()-functions by adding a CASE statement like the following:

```
SELECT ...,
       1 * ORDERING(f1()) +
       2 * ORDERING(f2()) AS indicator,
       CASE WHEN ORDERING(f1()) = 1 THEN f1()
            WHEN ORDERING(f2()) = 1 THEN f2()
       END AS value, ...
FROM ...
WHERE ...
ORDER BY ORDERING SET((f1()),(f2())) LIMIT BY :k
```

3.3. The LIMIT BY Over-Clause Extension

Similar to the relationship of GROUP BY (with GROUPING SETS()) and PARTITION BY in the context of the OVER()-clause, we extend the functionality of reporting functions by a local LIMIT BY clause resulting in multiple benefits. This implies that the restriction of the output data is now close to the RANK() function avoiding nested queries. The scenario above may now be specified without any nesting by the following expression:

```
SELECT ...,
       RANK() OVER(ORDER BY f1()
              LIMIT BY 10) AS pos1,
       RANK() OVER(ORDER BY f2()
              LIMIT BY 10) AS pos2,
       LIMIT() OVER(ORDER BY f1()
              LIMIT BY 10) AS indicator1,
       LIMIT() OVER(ORDER BY f2()
              LIMIT BY 10) AS indicator2
FROM ...
WHERE ...
```

From a local perspective of a single column, the values are sorted according to the given ORDER BY criterion. In a second step, the LIMIT BY-clause propagates the first k rows from the preceding sort operator to the following RANK() function. All subsequent values are replaced by NULL values indicating that they are not contributing to the overall result. As an indicator, the new LIMIT()-function returns a numeric 0 if the corresponding original value with regard to the given OVER()-clause is omitted and 1 if the original value is part of the aggregation process (in most cases applied to the RANK()-operator). The same semantics applies in the presence of an additional PARTITION BY-clause with an optional window specification. The limitation applies to each partition locally without affecting the succeeding window definition.

3.4. Summary

Supporting ordering in relational database systems has a long tradition to pre-process returning data for presentational use. With the advent of data warehouse and information retrieval applications limited orderings (i.e. ranks) and multiple orderings (according to different combinations of

the data space) are becoming tremendously important. We introduced a small and seamless SQL-extension dedicated to support these requirements. The ORDERING SETS and the LIMIT BY extension fits seamlessly into the SQL language and greatly enhances query capability by reducing the complexity of the query statement. The following two sections outline the implementation of a special operator for simple queries and join queries with multiple ranks.

4. Simple Queries with Multiple Ranks

This section introduces the MRANK()-operator supporting our new language concepts and details the underlying algorithm. In the presence of joins the MRANK()-operator will be combined with join algorithms as shown in the next section. Since the mechanism of locally computing ranks using the OVER-clause with the LIMIT BY-extension is similar to the global construct of ORDERING SETS(), we restrict the following discussion to the latter one.

Assume the underlying data is stored in a relation $R(tid, col_1, \ldots, col_m)$ and the ranking functions $F = \{f_1, \ldots, f_n\}$ order the objects in descending manner. When mapping the ORDERING SET()-operator to queries of forms like presented in section 3.1 the current implementations compute the query body individually, apply the specific ordering functions, return the first k rows, and concatenate the single tuple streams using an union operator thus forming the overall result stream. Figure 5a) illustrates this approach. Unfortunately, such implementations require the complete sort of the underlying data stream according to each ordering function f_i. Instead we propose a novel (logical) operator MRANK() with multiple corresponding (physical) operators, which can be directly exploited when parsing the query. Figure 5b) shows how the query plan changes.

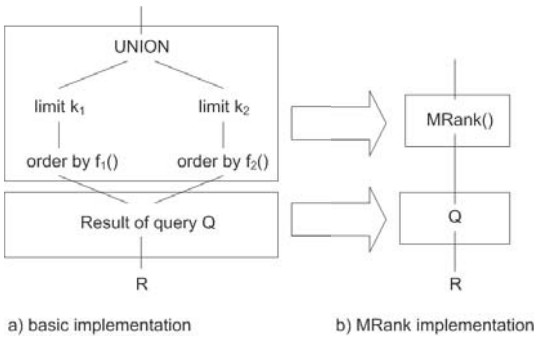


Figure 5. Applying MRank()-operator to compute limited ordering sets

The simplest implementation of the MRANK()-operator is based on data structures holding a minimal set of data in main memory. The algorithm holds a heap structure for every ranking criterion. It computes a single top-k query holding the $rowid$ and the values of the ordering function (Algorithm 1).

Algorithm 1 Main memory based algorithm

Require: Relation $R(id, col_1, \ldots, col_n)$
Query Q with ordering functions $F = \{f_1, \ldots, f_n\}$ and local limits $k_1, \ldots, k_n$
1: H:= set of heap data structures $h_i[id, f_i]$ of size k_i for all top-k operators $1 \leq i \leq n$
2: {Phase 1: Compute Multiple Orderings}
3: **for all** tuple $t \in R$ **do**
4: **for all** $i \in H$ **do**
5: **if** $h_i.count() < k_i$ **then**
6: $h_i.add(t[id], f_i(t))$
7: **else if** $f_i(t) > min(h_i)$ **then**
8: $h_i.remove(min(h_i))$
9: $h_i.add(t[id], f_i(t))$
10: **end if**
11: **end for**
12: **end for**
13: {Phase 2: Generate Output Stream}

In a first phase, each row of the incoming data stream is split according to the individual sorting criteria and added to the heap structure if the number of elements in the heap is still below the limit. If the heap size has already reached the limit, the current value c replaces the minimum value m stored at the top of the heap, if the $c > m$. When all heaps have been checked in that way the next tuple is processed.

In a second phase, the entries of the heaps are sorted, the final positions are assigned and the final rows are constructed and given to the next operator.

In case not enough main memory is available to hold all heaps, vertical or horizontal splitting and temp table techniques can be applied. However, due to the lack of space we do not present those technique in this paper .

Please note that the query body may include join-expressions. Still the basic MRANK()-operator can be applied, but the whole join has to be performed. In the next section we investigate how to push down the top-k criteria which may result in an early stop of the join.

5. Join Queries with Multiple Ranks

For single top-k queries with joins [10] proposes a rank join, which avoids to perform a full join of the underlying tables. First, we give an extension of the idea to deal with multiple top-k queries, which however requires the ranking functions to be linear combinations of the attributes.

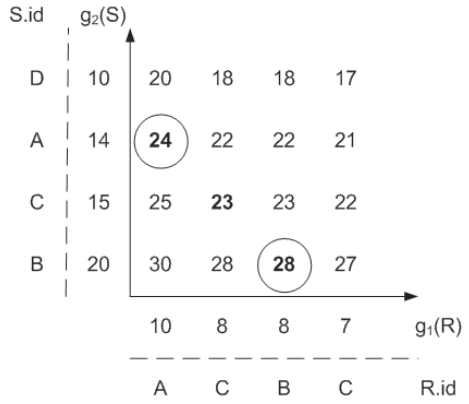


Figure 6. Evaluation of a single top-k join query

Second, we propose a new join method for multiple top-k queries, which is faster and is not limited to linear ranking functions.

5.1. Early Stop in Sort Merge Joins

The section of related work already mentioned the core principle of the approach given by [10]. From a join operator perspective, the main idea consists in stopping the flow of incoming tuples from the join partners, if future combinations of join tuples can never be within the set of the top-k tuples.

Given the join tables R and S and the ranking function $f(g_1(R), g_2(S))$ the key idea of the rank join is to process the tuples from R and S in decreasing order of $g_1(R)$ respectively $g_2(S)$. Let be x_{max} the first tuple (which yields the maximum of $g_1(R)$) and x the current tuple from R and y_{max} and y the analogous tuples from S then $T = max\{f(g_1(x_{max}), g_2(y)), f(g_1(x), g_2(y_{max}))\}$ is an upper bound for the ranks of unprocessed join combinations. If T is smaller than the smallest rank of the top-k tuples seen so far, than the join can be stopped early as no future tuple combination will be included in the top-k result.

Consider the following example with the ranking function $f(R,S) = g_1(R) + g_2(S)$, $g_1(R) = R.A_1 + R.A_2$ and $g_2(S) = S.B_1 + S.B_2$. The join condition is R.id=S.id. To achieve an early stop, the input data streams are sorted according to local scores $g_1(R)$ respectively $g_2(S)$. In the example of figure 6, after reading the third tuple of R and S the join can be early finished. The two circled results of the three matches are the top-2 tuples based on the ranking function. All other potential join combinations can never contribute to the top-2. An early stop after reading three tuples from R and only two tuples from S is not possible although two join combinations are found. This is because the upper bound T is still 25 and thus exceeds the smallest rank of top-k tuples seen so far, which is 23.

Although the idea works well for single top-k queries, however it is not directly applicable in the context of multiple independent ranking criteria. The problem is that in general we have no ordering of R and S for which we can determine an upper bound for all local scores of the different ranking functions.

In the special case that the ranking functions differ only in their local scores $f_i(R,S) = f(g_{1,i}(R), g_{2,i}(S)),\ 1 \leq i \leq n$ and the local scores have the form $g_{1,i}(R) = \alpha_{1,i} \cdot R.A_1 + \alpha_{2,i} \cdot R.A_2 + \ldots$ and $g_{2,i}(S) = \beta_{1,i} \cdot S.B_1 + \beta_{2,i} \cdot S.B_2 + \ldots$ we can extend the idea from [10] to multiple top-k queries.

Therefore we define a global score function for R $\bar{g}_1(R) = \max\limits_{1 \leq i \leq n} \{\alpha_{1,i}\}|R.A_1| + \max\limits_{1 \leq i \leq n} \{\alpha_{2,i}\}|R.A_2| + \ldots$. The global score function $\bar{g}_2(S)$ for S is analogous. It is easy to see that global scores $\bar{g}_1(R), \bar{g}_2(S)$ are always larger or equal than the respective local scores $g_{1,i}(R),\ g_{2,i}(S)$.

The tuples from tables R and S are processed in decreasing order according to their global scores $\bar{g}_1(R), \bar{g}_2(S)$. With the global sorting criteria we cannot assume that the largest local scores appear in the first tuples, so we must keep the maximum local scores $g_{1,i}^{max},\ g_{2,i}^{max}$ seen so far for each of the n ranking functions as we scan through R and S. Also the top-k tuples for each ranking seen so far are stored in heap structures H_i. Each heap has the smallest rank min_{H_i} of the particular ranking on top.

Let be x the current tuple from R and y the current tuple from S. Then $T_i = max\{f(g_{1,i}^{max}, \bar{g}_2(y)), f(\bar{g}_1(x), g_{1,i}^{max})\}$ is an upper bound for the ranks of future join combinations of ranking i. We can stop early if the condition $min_{H_i} \geq T_i$ holds for all rankings. In the experiment section we show that the proposed global sorting criteria has a negative effect on the performance of the algorithm because the linear combination of many attributes slowly pushes the upper boundaries to the lower values and thus delaying the early stops.

5.2. Early Stop in Hash Joins

The global orderings of R and S are suboptimal for most of the ranking functions. To avoid that disadvantage we can extend the hash-join principle to compute multiple top-k rankings and make use of early stops as much as possible. In this subsection we do not assume any special structure of the local scores $g_{1,i}(R)$ and $g_{2,i}(S)$ as well as the $f_i(\cdot,\cdot)$ might be different but monotonic.

The basic idea of the hash join extension is to partition the incoming (and not necessarily sorted) data stream of one join partner (in general the smaller table, say R) according

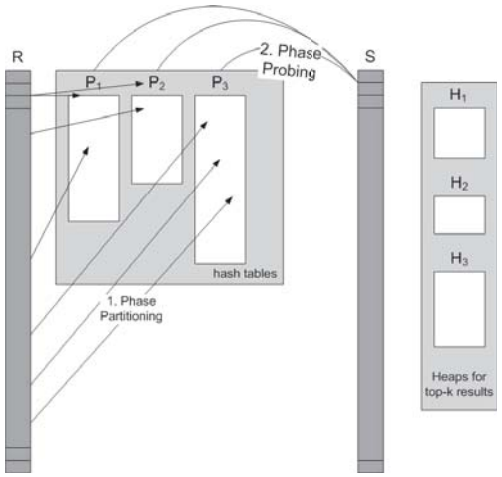


Figure 7. Evaluation multiple top-k with hash join

to the local scores $g_{1,i}(R)$ (and not according to the join attribute). Therefore we need n hash tables of size l_i, one for each local score.

Our hash-join for multiple top-k queries process R iteratively, with two phases per iteration. In a first phase we select the top-l_i tuples of R according to the local scores $g_{1,i}(R)$ into the hash tables P_i. For this we use the MRANK()-operator for simple multiple top-k queries (see section 4). Figure 7 illustrates an example with the three top-k ordering functions. Note that a single tuple may be placed into more than one hash table. As all hash tables fit in main memory this phase requires one scan over R.

In a second probing phase, all tuples of the join partner S are probed against the hash entries. If a join partner is found in hash table P_i, the joined tuple is inserted into the corresponding top-k heap structure H_i (used already within the MRANK()-operator) if the ith rank function of the combined tuple yields a value larger than the smallest top-k value for this ranking seen so far. The probing phase requires one scan over S. Then the hash tables are emptied and in the next iteration the next l_i tuples are filled into the hash tables.

The two phases are repeated until all entries of table R are handled once in each hash table P_i or the computation of all top-k values stops early. For early stops we maintain for each ranking an individual upper bound $T_i = f_i(min\{g_{1,i}(P_i)\}, max\{g_{2,i}(S)\})$. Note that the tuples of R are put into the hash table P_i in decreasing order according to $g_{1,i}()$. The maximal local score $g_{2,i}(S)$ can be determined during the probing phase of the first iteration. If the probing phase of the first iteration is not finished we use the maximum seen so far.

It is worth mentioning that each local partition P_i holds the complete tuple such that the final result can be computed without any further effort. An important effect on the performance of the algorithm have the cardinalities of the hash tables P_i. Given an amount C of main memory (in number of tuples) we determine the cardinalities $l_i = |P_i|$ as follows:

$$|P_i| = \frac{|C|}{\sum_{i=1}^{n} k_i} \cdot k_i$$

In this case the partitioning of the main memory depends only on the local limits of the multiple rankings. The runtime of the hash join for multiple top-k rankings is $O(iter \cdot (|R| + |S|))$, where $iter$ is the number of iterations. In the experimental section we show that in most cases the number of iteration is quite low, because the individual upper bounds T_i are quite tight.

5.3. Summary

In this section, we outlined two alternatives to push-down the limitation of multiple ranks into join operators. The first idea is based on the proposal of [10]. This only suitable if the sorting expressions have special structure and are highly correlated. For the general case of arbitrary sorting expressions, we propose a solution based on the hash join technique, which is expected to run faster than the sort-merge join approach.

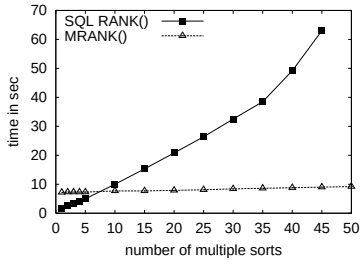
6. Experiments

We conducted multiple experiments of our MRANK operator to demonstrate the benefit in multiple situations, i.e. with different implementations and different parametric environments. All experiments were carried out on a Linux machine with an AMD Athlon XP 3000+ CPU and 1.5 GB main memory. The following subsections describe different scenarios based on single table expressions and – most important – in combination with joins.

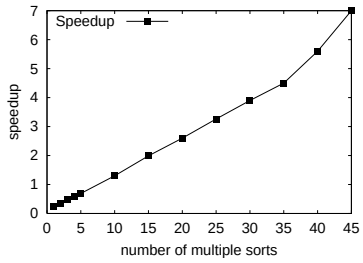
6.1. Simple Queries with Multiple Ranks

In this section we present our experiment results, which are based on simple queries. The real data we used come from the UCI KDD Archive and contain 32-dimensional color histograms of 66.615 images. For this experiments we used the commercial DB2 V8.1 database system. We also implemented a prototype of our MRANK-operator as C++ client on top of DB2, which has to read the data over an ODBC connection. So our MRANK-operator had a much slower access to the data as the system itself.

We simulated an image search application based on complex similarity search queries. For the experiment we varied the number of query points in the query point set Q from 1



(a)



(b)

Figure 8. Experimental comparison: images

to 45. Each query point translates to a separate ranking criteria. The limit parameter was set to $k = 20$, which means that each query point the 20 best matching objects were returned. The results are shown in figure 8. The MRANK implementation performs better than the database systems original implementation when more than 5 query points (different ranking functions) are used. We argue that the minor performance for small query point sets (1-5 points) is caused by the top of database implementation of our prototype. In case of 45 query points the MRANK-operator is 7 times faster than the database system. As the query point set and the limit k were reasonably small the main memory approach was used only.

6.2. Join Queries with Multiple Ranks

We proposed two algorithms to optimize multiple ranks over joins. The first algorithm is an extension of [10] considering multiple top-k ranking, which have a special form (linear). Ilays et. al. [10] showed that the rank join operator outperforms existing methods in database today. Our approaches performs also better than the today's database operators, because we avoid multiple sorts after joining R and S. Therefore we concentrate on analyzing and comparing our two proposed algorithms for joins in this section.

In the first experiment we compared the run time of the two approaches. The result is shown in figure 9. For this experiments we generated relations where the local scores are independently from the join condition and varied the number of multiple top-k ordering functions ($k = 10$). The relation R contains $50,000$ and S $150,000$ tuples. With a join selectivity of 0.01, $75,000,00$ join combinations exists. All temporary data structures are kept in main memory. For the hash join we limited the main memory space for all hash tables to $1,000$ tuples and respectively to $5,000$ tuples. We distributed the available space proportional to individuals hash tables. With increasing number of ranking functions the sizes of the individual hash tables decrease.

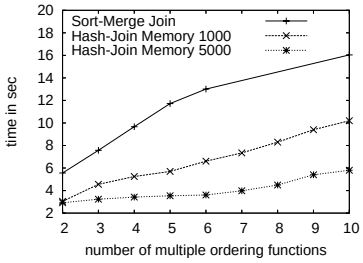


Figure 9. Comparing Sort-Merge and Hash Join approach

The hash join outperforms the sort merge approach, because the hash join does not sort the relations according global score functions. Furthermore, the upper bounds for early stop condition of the sort merge join become less tight as the number of ranking functions increases. Beyond a certain number of rankings all tuples of R and S have to be processed (see figure 10). A second observation from figure 9 is that the hash join gets faster when more main memory is available because of fewer scans of S.

That fact is further investigated in the second experiment. It shows the effect of the main memory on the performance of the algorithm. We varied the number of tuples in R and generated S, such that each tuple in R had 5 join partners in S. In the experiments we computed ten top-k ranking function, with $k = 20$. Figure 11(a) shows the result for main memory space of $1,000$ and $5,000$ tuples for the hash tables. The figure shows clearly, that the hash join can utilize the larger main memory very effectively.

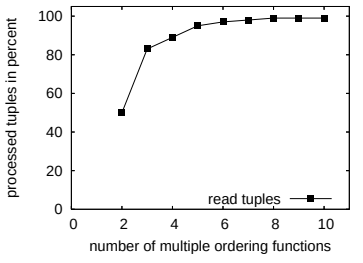


Figure 10. Processed tuples, Sort-Merge Join

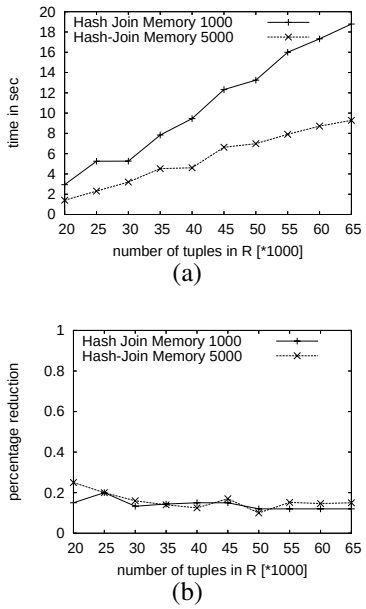


Figure 11. Hash Join

Our hash join approach extents the classic hash technique to join two relations considering multiple ranking functions. The classic hash join technique puts a fraction of the tuples from the smaller relation in the main memory. With a scan over S all tuples of S are probed against the tuples in the main memory hash tables. The amortized complexity is $O(R + \frac{C}{|R|} \cdot S)$, because the relation R have to be read only once. In our approach we invest more time to built up the hash tables but can reduce on the other hand the number of scans over S because of early stops. Figure 11(b) presents the percentage reduction of the number of scan over S compared to the classic hash join with the same amount of main memory.

7. Conclusion

In this paper, we analyzed the problem of supporting multiple top-k queries from a relational database engine perspective. We proposed a minimal SQL extension to ease the specification of multiple rankings within one SQL query and gave some ideas of applications which can benefit from it. Additionally, we proposed a variant of the well-known hash-join strategy which enables an early pruning of potential join candidates. Finally, we demonstrated the feasibility of our approach with a variety of different experiments. With our proposed SQL extension of ORDERING SET and column wise limitation in combination with an optimized implementation, we are convinced that this technology pushes the envelope and makes relational database technology more applicable for a huge range of data-intensive applications.

References

[1] Nicolas Bruno, Suraji Chaudhuri, and Luis Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Trans. Database Syst.*, 27(2):153–187, 2002.

[2] Nicolas Bruno, Luis Gravano, and Amélie Marian. Evaluating top-k queries over web-accessible databases. In *Proceedings of the 18th International Conference on Data Engineering*, pages 369–, 2002.

[3] Michael J. Carey and Donald Kossmann. On saying "enough already!" in sql. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 219–230. ACM Press, 1997.

[4] Michael J. Carey and Donald Kossmann. Reducing the braking distance of an sql query engine. In *Proceedings of 24rd International Conference on Very Large Data Bases*, pages 158–169. Morgan Kaufmann, 1998.

[5] Surajit Chaudhuri and Luis Gravano. Evaluating top-*k* selection queries. In *Proceedings of 25th International Conference on Very Large Data Bases*, pages 397–410, 1999.

[6] Chung-Min Chen and Yibei Ling. A sampling-based estimator for top-*k* query. In *Proceedings of the 18th International Conference on Data Engineering*, pages 617–629, 2002.

[7] Donko Donjerkovic and Raghu Ramakrishnan. Probabilistic optimization of top n queries. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 411–422. Morgan Kaufmann, 1999.

[8] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2001.

[9] Vagelis Hristidis, Nick Koudas, and Yannis Papakonstantinou. Prefer: a system for the efficient execution of multi-parametric ranked queries. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 259–270. ACM Press, 2001.

[10] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. In *Proceedings of 29th International Conference on Very Large Data Bases (VLDB'03,September 9-12, 2003, Berlin, Germany)*, pages 754–765, 2003.

[11] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. *VLDB Journal*, 13(3):207–221, 2004.

[12] Deok-Hwan Kim and Chin-Wan Chung. Qcluster: relevance feedback using adaptive clustering for content-based image retrieval. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data*, pages 599–610, 2003.

[13] Giedrius Slivinskas, Christian S. Jensen, and Richard T. Snodgrass. Bringing order to query optimization. *SIGMOD Rec.*, 31(2):5–14, 2002.

Integrating Heterogeneous Multidimensional Databases

Luca Cabibbo and Riccardo Torlone
Dipartimento di Informatica e Automazione
Università Roma Tre
{cabibbo, torlone}@dia.uniroma3.it

Abstract

In this paper we present a number of techniques that can be at the basis of a practical integration tool for multidimensional databases. We start by addressing the basic issue of matching heterogeneous dimensions and provide a number of general properties that a dimension matching should fulfill. We then propose two different approaches to the problem of integration that try to enforce matchings satisfying these properties. The first approach refers to a scenario of loosely coupled integration, in which we just need to identify the common information between sources and perform drill-across queries over the original sources. The goal of the second approach is the derivation of a materialized view built by merging the sources, and refers to a scenario of tightly coupled integration in which queries are performed against the view. We finally show how these techniques can be actually used to perform drill-across operations over heterogeneous multidimensional information sources.

1 Introduction

The problem of integrating heterogeneous multidimensional databases arises in common scenarios in which information from autonomous (i.e., independently developed and operated) data marts need to be combined. A common practice for building a data warehouse is indeed to implement a series of integrated data marts, each of which provide a dimensional view of a single business process. These data marts should be based on conformed (i.e., common) dimensions and facts, but very often different departments of the same company develop their data marts independently, and it turns out that their integration is a difficult task. The need for combining autonomous data marts arises in other common cases. For instance, when different companies merge or get involved in a federated project or when there is the need to combine a proprietary data warehouse with data available elsewhere, for instance, in external (and likely heterogeneous) data warehouses.

In an earlier paper [5], we have introduced and investigated a fundamental notion underlying data mart integration: dimension *compatibility*. Intuitively, two dimensions (belonging to different data marts) are compatible if their common information is consistent. We have shown that dimension compatibility gives the ability to correlate, in a correct way, multiple data marts by means of *drill-across queries* [9], based on joining data over common dimensions. Building on this preliminary study, in this paper we introduce a number of notions and algorithms that can be used in a practical integration tool for multidimensional databases, similarly to how Clio [12] supports heterogeneous data transformation and integration.

We start from the problem of integrating a pair of autonomous dimensions and identify a number of desirable properties that a *matching* between dimensions (that is, a one-to-one correspondence between their levels) should satisfy: the *coherence* of the hierarchies on levels, the *soundness* of the paired levels, according to the members associated with them, and the *consistency* of the functions that relate members of different levels within the matched dimensions. We propose two different approaches to the problem of integration that try to enforce matchings satisfying the above properties. The first approach refers to a scenario of loosely coupled integration, in which we need to identify the common information between sources (intuitively, the intersection), while preserving their autonomy. This approach supports drill-across queries, to be performed over the original sources. The goal of the second approach is rather merging the sources (intuitively, making the union) and refers to a scenario of tightly coupled integration, in which we need to build a materialized view that includes the sources. With this approach, queries are then performed against the view built from the sources. As a preliminary tool, we introduce a powerful technique, the *chase of dimensions*, that can be used in both approaches to test for consistency and combine the content of the dimensions to integrate.

We believe that the proposed techniques can be applied

in more general contexts in which there is the need to integrate generic heterogenous data sources and we have at our disposal taxonomies of concepts describing the sources (e.g, ontologies).

The concept of compatibility among dimensions in a data warehouse has been discussed, under the name of "conformity", by Kimball [9] in the context of data warehouse design. Our notion of compatibility is actually more suitable to autonomous multidimensional data integration than the notion of conformity since we consider an "integration" perspective rather than a "design" one. The integration of heterogenous databases has been studied in the literature extensively (see, e.g., [6, 7, 10, 13, 17]). In this paper, we take apart the general aspects of the problem and concentrate our attention on the specific problem of integrating *multidimensional* data. Differently from the general case, this problem can be tackled in a more systematic way for two main reasons. First, multidimensional databases are structured in a rather uniform way, along the widely accepted notions of dimension and fact. Second, data quality in data warehouses is usually higher than in generic databases, since they are obtained by reconciling several data sources. To our knowledge, the present study is the first systematic approach to this problem. A somehow related issue is the *derivability* of summary data from heterogeneous data sets in the context of statistical databases [11, 18]. Some work has been done on the problem of integrating data marts with external data, stored in various formats: XML [8, 14] and object-oriented [15]. This is related to our tightly coupled approach to integration, where dimensions are "enriched" with external data. On the other hand, our loosely coupled approach to integration is related to the problem of *drill-acrossing* [1]. Finally, the chase of dimensions can be viewed as exact method of *missing value imputation*, which has been studied in statistical data analysis and classification, for instance, by use of estimation with the EM algorithm [16]. However, the goal of these studies is different from ours.

The paper is organized as follows. In Section 2 we recall a multidimensional model that will be used throughout the paper. In Section 3 we present the notion of dimension matching and provide a basic tool, called d-chase, for the management of matchings. In Section 4 we illustrate two techniques for dimension integration and, in Section 5, we describe how they can be used to integrate data marts. Finally, in Section 6, we sketch some conclusions.

2 Preliminaries

2.1 A dimensional data model

In this section, we will briefly recall the $\mathcal{MD}$ data model [4], a multidimensional conceptual data model. It generalizes the notions commonly used in multidimensional analysis or available in commercial OLAP systems and, for this reason, is adopted as a basic framework for our study. $\mathcal{MD}$ is based on two main constructs: the dimension and the data mart.

Definition 2.1 (Dimension) A *dimension* d is composed of:

- a *scheme* $S(d)$, made of: (i) a finite set $L = \{l_1, \ldots, l_n\}$ of *levels*, and (ii) a partial order $\preceq$ on L (if $l_1 \preceq l_2$ we say that l_1 *rolls up to* l_2), and
- an *instance* $I(d)$, made of: (i) a function m associating *members* with levels; and (ii) a family of functions ρ including a *roll up function* $\rho^{l_1 \to l_2} : m(l_1) \to m(l_2)$ for each pair of levels $l_1 \preceq l_2$.

We assume that L contains a bottom element $\perp$ (wrt $\preceq$) whose members represent real world entities that we call *basic*.[1] Members of other levels represent groups of basic members.

Let $\{\tau_1, \ldots, \tau_k\}$ be a predefined set of *base types*, (including integers, real numbers, etc.).

Definition 2.2 (Data mart) A *data mart* f over a set D of dimensions is composed of:

- a *scheme* $f[A_1 : l_1, \ldots, A_n : l_n] \to \langle M_1 : \tau_1, \ldots, M_m : \tau_m \rangle$, where each A_i is a distinct *attribute* name, each l_i is a level of some dimension in D, each M_j is a distinct *measure* name, and each τ_j is some base type; and
- an *instance*, which is a partial function mapping coordinates for f to facts for f, where:
  - a *coordinate* is a tuple over the attributes of f mapping each attribute name A_i to a member of l_i;
  - a *fact* is a tuple over the measures of f mapping each measure name M_j to a value in the domain of type τ_j.

Example 2.1 *Figure 1 shows a* Sales *data mart that represents daily sales of products in a chain of stores.*

It is worth noting that, according to the traditional database terminology, the $\mathcal{MD}$ is a *conceptual* data model and therefore its schemes can be implemented using several *logical* data models [3].

[1] In [5], we called them *ground* members.

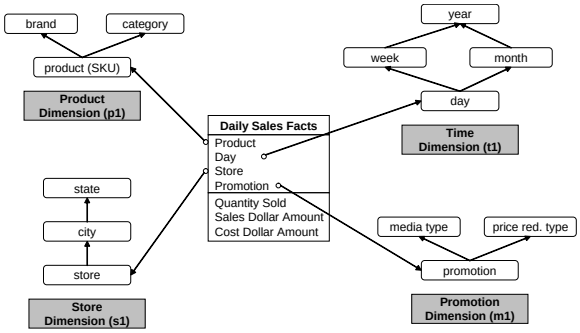


Figure 1. Sales data mart

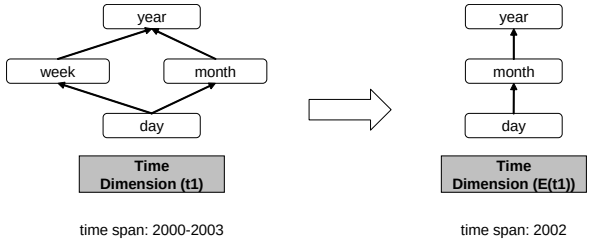


Figure 2. Application of a DA expression

2.2 An algebra for dimensions

Let d denote a dimension having scheme $(L, \preceq)$ and instance (m, ρ). The *dimension algebra* (*DA*) can be used to manipulate dimensions and is based on three operators, as follows.

Definition 2.3 (Selection) Let G be a subset of the basic members of d. The *selection* $\sigma_G(d)$ of G over d is the dimension d' such that: (i) the scheme of d' is the same of d and (ii) the instance of d' contains: the basic members in G, the members of d that can be reached from them by applying roll-up functions in ρ, the restriction of the roll-up functions of d to the members of d'.

Definition 2.4 (Projection) Let X be a subset of the levels of d including $\perp_d$. The *projection* $\pi_X(d)$ of d over X is the dimension d' such that: (i) the scheme of d' contains X and the restriction of $\preceq$ to the levels in X, (ii) the instance of d' contains: the members of d that belong to levels in X and the roll-up functions $\rho^{l_1 \to l_2}$ of d involving levels in X.

Definition 2.5 (Aggregation) Let l be a level in L. The *aggregation* $\psi_l(d)$ of d over l is the dimension d' such that: (i) the scheme of d' contains l, the levels of d to which l rolls up, and the restriction of $\preceq$ to these levels, and (ii) the instance of d' contains: the members of d that belong to levels in d' and the roll-up functions $\rho^{l_1 \to l_2}$ of d involving levels in d'.

For a DA expression E and a dimension d, we denote by $E(d)$ the dimension obtained by applying E to d.

Example 2.2 *Let us consider the time dimension $t1$ of the data mart in Figure 1 and let D_{2002} denote the days that belong to year 2002. The DA expression $E = \pi_{day,month,year}(\sigma_{O_{2002}}(t1))$ generates a new dimension with level day, month and year having as basic members all the days of 2002 (see Figure 2).*

The following is a desirable property of DA expressions.

Definition 2.6 (Lossless expression) A DA expression E over a dimension d is *lossless* if for each member o in $E(d)$, all the members that roll up to o in d belong to $E(d)$.

In [5] we have shown that the satisfaction of this property prevents inconsistencies between aggregations over d and aggregations over $E(d)$.

DA expressions involving only projections and aggregations are always lossless [5]. On the other hand, if a DA expression involves selections, the lossless property can fail to hold: it depends on the particular sets of elements chosen to perform the selections.

3 Matching autonomous dimensions

In what follows, d_1 and d_2 denote two dimensions, belonging to different data marts, having scheme $S(d_i) = (L_i, \preceq_i)$ and instance $I(d_i) = (m_i, \rho_i)$, respectively.

3.1 Dimension matching and its properties

Let us start with the basic notion of dimension matching.

Definition 3.1 (Dimension Matching) A *matching* between two dimensions d_1 and d_2 is a (one-to-one) injective partial mapping μ between L_1 and L_2.

With a little abuse of notation, given a matching μ, we will denote by μ also its inverse. We also extend μ to sets of levels in the natural way (that is, $\mu(L)$ is the set containing $\mu(l)$ for each level l in L). Also, we will assume that μ is the identity on the levels on which it is not defined.

A number of desirable properties can be defined over a matching between dimensions.

Definition 3.2 (Matching Properties) Let μ be a matching between two dimensions d_1 and d_2. Then:

- **Coherence:** μ is *coherent* if, for each pair of levels l, l' of d_1 on which μ is defined, $l \preceq_1 l'$ if and only if $\mu(l) \preceq_2 \mu(l')$;

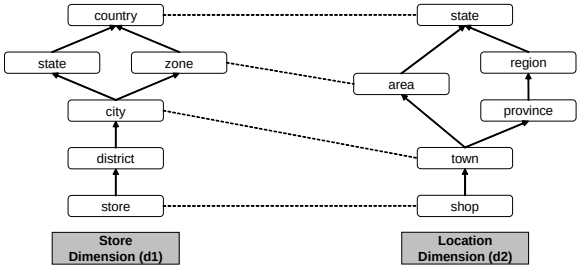


Figure 3. A matching between two dimensions

- **Soundness:** μ is *sound* if, for each level $l \in L_1$ on which μ is defined, $m_1(l) = m_2(\mu(l))$;[2]
- **Consistency:** μ is *consistent* if, for each pair of levels $l \preceq_1 l'$ of d_1 on which μ is defined, $\rho_1^{l \to l'} = \rho_2^{\mu(l) \to \mu(l')}$.

A total matching that is coherent, sound and consistent is called a *perfect* matching.

Note that coherence means order preservation, soundness means member set preservation, and consistency means roll-up functions preservation.

Example 3.1 *Figure 3 shows an example of matching between two geographical dimensions that associates store with shop, city with town, zone with area, and country with state. The matching is coherent. If the mapped levels have the same members it is also sound. Consistency follows from the equivalence of the roll-up functions between levels.*

Clearly, a perfect matching is very difficult to achieve in practice. In many cases however, autonomous dimensions actually share some information. To identify this common information, we need the ability to select a portion of a dimension. This comment leads to the following definition.

Definition 3.3 (Dimension Compatibility) Two dimensions d_1 and d_2 are *compatible* if there exist two lossless DA expressions E_1 and E_2 over d_1 and d_2, respectively, such that there is a perfect matching between $E_1(d_1)$ and $E_2(d_2)$. In this case we say that d_1 and d_2 are compatible using E_1 and E_2.

The rationale underlying the definition of compatibility is that: (i) two dimensions may have common information; (ii) the intersection can be identified by DA expressions; and (iii) lossless expressions guarantee the correctness of OLAP operations over the intersection [5].

Example 3.2 *The dimensional matching reported in Figure 3 can be made perfect by applying the following expressions to d_1 and d_2:*

$$\pi_{store,city,zone,country}(\sigma_{m_1(\text{store}) \cap m_2(\text{shop})}(d_1)),$$

$$\pi_{shop,town,area,state}(\sigma_{m_1(\text{store}) \cap m_2(\text{shop})}(d_2)).$$

provided that the roll-up functions of the two dimensions are consistent. If the original dimensions had basic members in common and the selection over them made the two expressions lossless, then they would be compatible.

3.2 Chase of dimensions

We now describe a procedure called d-chase (for chase of dimensions) that applies to members of autonomous dimensions and show that it can be used for integration purposes.

Let V be a set of *variables* and $L = l_1, \ldots, l_k$ be a set of levels. A *tableau* T over L is a set of tuples t mapping each level l_i to a member of l_i or a variable in V.

Now, let μ be a matching between two dimensions d_1 and d_2.

Definition 3.4 (Matching Tableau (MT)) The *matching tableau* over d_1, d_2 and μ, denoted by $T_\mu[d_1, d_2]$, is a tableau over $L = L_1 \bigcup \mu(L_2)$ having a tuple t_m for each member m of a level $l \in L$ such that:

- $t_m[l] = m$,
- $t_m[l'] = \rho^{l \to l'}(m)$, for each level l' to which l rolls up,
- $t_m[l''] = v$, where v is a variable not occurring elsewhere, for all other levels in L.

Example 3.3 *A possible matching tableau for the matching between dimensions in Figure 3 is the following.*

store	city	zone	country	district	state	prov.	region
1st	NewYork	v_1	USA	v_2	NY	v_3	v_4
2nd	LosAng.	U2	USA	Melrose	CA	v_5	v_6
1er	Paris	E1	France	Marais	v_7	v_8	v_9
1mo	Rome	E1	Italy	v_{10}	v_{11}	RM	Lazio
1st	NewYork	U1	USA	v_{12}	v_{13}	v_{14}	v_{15}
1er	Paris	E1	France	v_{16}	v_{17}	75	IledeFr

In this example, the first three tuples represent members of d_1 and the others members of d_2. The first four columns represent the matched levels and the other columns represent levels of the two dimensions that have not been matched. Note that a variable occurring in a tableau may

[2]Note that, for simplicity, we follow a *conceptual* approach, under which two levels coincides if they are populated by the same real world entities. In a *logical* approach this notion would be based on a bijection between the identifiers representing the entities.

represents an unknown value (for instance, in the first row, the zone in which the store 1st *is located, an information not available in the instance of d_1) or an inapplicable value (for instance, in the last row, the district in which the store* 1er *is located, a level not present in the scheme of d_2).*

The *d-chase* (chase of dimensions) is a procedure inspired by an analogous procedure used for reasoning about dependencies in the relational model [2]. This procedure takes as input a tableau T over a set of levels L and generates another tableau that, if possible, satisfies a set of roll-up functions ρ defined over the levels in L. This procedure modifies values in the tableau, by applying *chase steps*. A chase step applies when there are two tuples t_1 and t_2 in T such that $t_1[l] = t_2[l]$ and $t_1[l'] \neq t_2[l']$ for some roll up function $\rho^{l \to l'} \in \rho$ and modifies the l'-values of t_1 and t_2 as follows: if one of them is a constant and the other is a variable then the variable is changed (is *promoted*) to the constant, otherwise the values are equated. If a chase step tries to identify two constants, then we say that the d-chase encounters a *contradiction*, and the process stops generating a special tableau that we denote by T_∞ and call the inconsistent tableau.

Definition 3.5 (D-chase) The d-chase of a tableau T, denoted by $DCHASE_\rho(T)$, is a tableau obtained from T and a set of roll-up functions ρ by applying all valid chase steps exhaustively to T.

Example 3.4 *By applying the d-chase procedure to the matching tableau of Example 3.3 we do not encounter contradictions and obtain the following tableau.*

store	city	zone	country	district	state	prov.	region
1st	NewYork	U1	USA	v_2	NY	v_3	v_4
2nd	LosAng.	U2	USA	Melrose	CA	v_5	v_6
1er	Paris	E1	France	Marais	v_7	75	IledeFr
1mo	Rome	E1	Italy	v_{10}	v_{11}	RM	Lazio

The d-chase promotes, for instance, v_1 to U1, and v_8 to 75.

Note that in the d-chase procedure, a promotion of a variable always corresponds to the detection of an information present in the other dimension and consistent with the available information but not previously known.

An important result about the d-chase, which follows from general properties of d-chase, is the following.

Lemma 3.1 *The d-chase process terminates on any input with a unique end result.*

The following result states that the d-chase provides an effective way to test for consistency.

Theorem 3.5 *A matching μ between two dimensions d_1 and d_2 is consistent if and only if $DCHASE_{\rho_1 \cup \mu(\rho_2)}(T_\mu[d_1, d_1]) \neq T_\infty$.*

We finally define a special operation over a tableau that will be used in the following. Let T be a tableau over a set of levels L and $S = (L', \preceq)$ be the scheme of a dimension such that $L' \subseteq L$.

Definition 3.6 (Total projection) The *total projection* of T over S, denoted by $\pi^{\downarrow}_S(T)$, is an instance (m, ρ) of S defined as follows.

- for each level $l \in L$, $m(l)$ includes all the members occurring in the l-column of T.
- for each pair of levels l_1, l_2 in L such that $l_1 \preceq l_2$ and for each tuple t of T such that: (i) both the l_1-value and the l_2-value are defined, and (ii) there is no other tuple t' in T such that $t[l_1] = t'[l_1]$ and $t[l_2] \neq t'[l_2]$, then $\rho^{l_1 \to l_2}(t[l_1]) = t[l_2]$ and is undefined otherwise.

Let d be a dimension and μ a matching between d and any other dimension d'. We can easily show the following.

Lemma 3.2 $I(d) \subseteq \pi^{\downarrow}_{S(d)}(DCHASE_{\rho \cup \mu(\rho')}(T_\mu[d, d']))$.

This result states an interesting property of the chase that goes beyond the test of consistency. If we apply the d-chase procedure over a matching tableau that involves a dimension d and then project the result over the scheme of d, we obtain the original instance and, possibly, some additional (and consistent) information that has been identified in the other dimension. As noted above, this situation occurs when, in a tuple for a member in d, the d-chase promotes a variable to a member of the other dimension.

4 Two approaches to dimension integration

In this section we propose two different approaches to the problem of the integration of autonomous data marts.

4.1 A loosely coupled approach

In a *loosely coupled integration* scenario, we need to identify the common information between various data sources and perform drill-across queries over the original sources. Therefore, our goal is just to select data that is shared between the sources. Thus, given a pair of dimensions d_1 and d_2 and a matching μ between them, the approach aims at deriving two expressions that makes μ perfect. The approach is based on Algorithm 1, which is reported in Figure 4.

First of all, the algorithm selects the levels L of d_1 involved in the matching μ (Step 1). Then, for each minimal

Algorithm 1
Input: two dimensions d_1 and d_2 and a matching μ;
Output: two expressions E_1 and E_2 that make μ perfect;
begin
1) $L :=$ the levels of d_1 involved in μ;
2) **for each** minimal level l_m of L **do**
3) $L := L - \{l \in L$ such that $l_m \not\preceq_1 l\}$;
4) **if** there exist $l_1, l_2 \in L$ **such that** $l_1 \preceq_1 l_2$ **and** $\mu(l_1) \not\preceq_2 \mu(l_2)$ **then output** 'not coherent' **and exit**;
5) $E_1 := \pi_L(\psi_{l_m}(d_1))$; $E_2 := \pi_{\mu(L)}(\psi_{\mu(l_m)}(d_2))$;
6) $M := m_1(l_m) \bigcap m_2(\mu(l_m))$;
7) $T := T_\mu[\sigma_M(E_1(d_1)), \sigma_M(E_2(d_2))]$;
8) $T := {}_{DCHASE_{\rho_1 \cup \mu(\rho_2)}}(T)$;
9) **if** $T = T_\infty$ **then output** 'not consistent' **and exit**;
10) $d_1 := \pi^{\downarrow}_{S(d_1)}(T)$; $d_2 := \pi^{\downarrow}_{S(d_2)}(T)$;
11) **for each** non basic member $m \in m_{1,2}(l)$ in T **do**
12) **if** $\exists\, m' \in m_{1,2}(l')$ **such that** $l' \preceq_{1,2} l$ **and** $\rho^{l' \to l}_{1,2}(m') = m$ **and** m' does not occur in T **then** $T := T - \{t \mid t[l] = m\}$
13) $M := \{m \mid t[l_m] = m$ for some $t \in T\}$;
14) $E_1 := \sigma_M(E_1(d_1))$; $E_2 := \sigma_M(E_2(d_2))$;
15) **output** E_1 and E_2;
endfor
end

Figure 4. An algorithm for deriving the common information between two dimensions.

level l_m in L (that is, for which there is no other level $l \in L$ such that $l \preceq_1 l_m$), it selects only the levels to which l_m rolls up (Step 3). The rationale is to find the expressions that detect the intersection of d_1 and d_2 in the levels above l_m. If there are several minimal levels, the algorithm iterates over all of them (Step 2) thus possibly generating several pairs of expressions.

Step 4 consists of testing for coherence of the matching according to Definition 3.2. Actually, this test can be done efficiently by taking advantage of the transitivity of $\preceq$.

In Step 5 two preliminary expressions E_1 and E_2 are identified: they aggregate over l_m ($\mu(l_m)$) and project over L ($\mu(L)$). Since no selection is involved, by a result in [5], these expressions are lossless.

The rest of the algorithm aims at finding the selection of members that, applied to E_1 and E_2, identifies common data in the two dimensions. This is done by building a matching tableau over the members that occur both in l_m and $\mu(l_m)$ (Steps 6 and 7) and then chasing it (Step 8). According to Theorem 3.5, this corresponds to a test of consistency for the restriction of μ to the levels in L.

As we have noticed at the end of Section 3, when the d-chase promotes a variable to a member, this means that a previously unknown value in one dimension has been identified in the other dimension. To preserve soundness, this event asks for the addition of this member in the original dimension: this is implemented by Step 10.

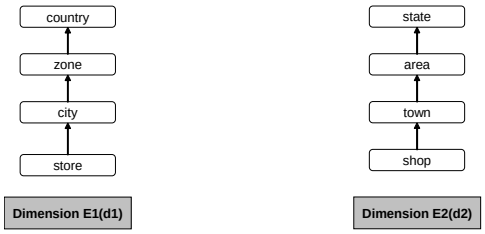


Figure 5. The dimensions generated by Algorithm 1 on the matching in Figure 3

Steps 11 and 12 serves to identify, from the members occurring in the working tableau T, all the members that invalidate the property of lossless expression (Definition 2.6). Finally, all the members that still occur in T at level l_m are used to perform the final selection (Steps 13 and 14).

Example 4.1 *Let us consider the application of Algorithm 1 to the dimensions and the matching in Figure 3, assuming that that the dimensions are populated by the members of Example 3.3. Since the matching involves the bottom levels of the two dimensions, no aggregation is required and the first part of the algorithm generates the following expressions:* $\pi_{store,city,region,country}(d_1)$ *and* $\pi_{shop,town,area,state}(d_2)$. *The intersection of the basic members contains only the stores* 1st *and* 1er *and so the d-chase produces the following tableau:*

store	city	zone	country	district	state	prov.	region
1st	NewYork	U1	USA	v_2	NY	v_3	v_4
1er	Paris	E1	France	Marais	v_7	75	IledeFr

This tableau contains the member E1 at the zone level to which a member of d_2 *rolls up (the store* 1mo*), but is not present in the tableau. It follows that in Step 12 the second row is deleted and we obtain as output of the algorithm the following final expressions:*

$$\sigma_{\{1st\}}(\pi_{store,city,region,country}(d_1)),$$

$$\sigma_{\{1st\}}(\pi_{shop,town,area,state}(d_2)).$$

The schemes of the dimensions we obtain by applying these expressions to the original dimensions are reported in Figure 5.

By construction, and according to the results of the previous section, we can state the following.

Theorem 4.2 *The execution of Algorithm 1 over two dimensions* d_1 *and* d_2 *and a matching* μ *between them returns*

Algorithm 2
Input: two dimensions d_1 and d_2 and a matching μ;
Output: a new dimension d that embeds d_1 and d_2;
begin
1) $L :=$ the levels of d_1 involved in μ;
2) **if** there exist $l_1, l_2 \in L$ **such that**
$l_1 \preceq_1 l_2$ **and** $\mu(l_1) \not\preceq_2 \mu(l_2)$
then output 'not coherent' **and exit**;
3) $L := L_1 \bigcup \mu(L_2)$;
4) $\preceq := (\preceq_1 \bigcup \mu(\preceq_2))^+$;
5) **if** $\preceq$ has several minimal levels
then
6) $d_1' :=$ d_1 augmented with a new bottom level $\perp_1'$;
7) $d_2' :=$ d_2 augmented with a new bottom level $\perp_2'$;
8) $\mu' := \mu \bigcup \{(\perp_1', \perp_2')\}$;
9) $L := L \bigcup \perp_1'$;
10) $\preceq := (\preceq_1' \bigcup \mu(\preceq_2'))^+$;
else $d_1' := d_1$; $d_2' := d_2$; $\mu' := \mu$;
11) $T := {}_{DCHASE_{\rho_1' \cup \mu(\rho_2')}}(T_{\mu'}[d_1', d_2'])$;
12) **if** $T = T_\infty$ **then output** 'not consistent' **and exit**;
13) $d := \pi^{\downarrow}_{(L,\preceq)}(T)$;
14) **output** the dimension d;
end

Figure 6. An algorithm for merging two dimensions.

two expressions E_1 and E_2 if and only if d_1 and d_2 are compatible using E_1 and E_2.

The most expensive step of the algorithm is the d-chase that requires time polynomial with respect to the size of the tableau, which in turn depends on the cardinality of the dimensions involved. It should be said however that the size of dimensions in a data warehouse is much smaller than the size of the facts. Moreover, the content of a dimension is usually stable in time. It follows that the algorithm can be executed off-line and occasionally, when it arises the need for integration or when changes on dimensions occur.

4.2 A tightly coupled approach

In *tightly coupled integration*, we want to build a materialized view combining different data sources and perform queries over this view. Our goal is the derivation of new dimensions obtained by merging the dimensions of the original data sources. In this case, given a pair of dimensions d_1 and d_2 and a matching μ between them, the integration technique aims at deriving a new dimension obtained by merging the levels involved in μ and including, but taking apart, all the other levels. The approach is based on Algorithm 2, which is reported in Figure 6.

First of all, similarly to Algorithm 1, Algorithm 2 performs a check for coherence of the input matching. If the test is successful, it then builds a new (preliminary) dimension scheme $S = (L, \preceq)$ by merging the levels (Step 3) and the roll-up relations between levels (Step 4) of the input dimensions. For the latter, we need to guarantee that the relation we obtain is a partial order. Irreflexivity and asymmetry follow by the coherence of the matching. To enforce transitivity, the transitive closure is computed over the union of the two roll-up relations.

Next step takes into account the special case in which the relation $\preceq$ we obtain has more than one minimal level. In this case, in Steps 6 and 7, two new auxiliary bottom levels $\perp_1'$ and $\perp_2'$ are added below the original bottom levels $\perp_1$ and $\perp_2$ of d_1 and d_2, respectively (clearly, this can be done without actually modifying the original dimensions). These levels have as members copies of the basic members of $\perp_1$ and $\perp_2$, suitably renamed so that the intersection of the two sets of copies is empty. Then, two new roll-up functions $\rho^{\perp_1' \to \perp_1}$ and $\rho^{\perp_2' \to \perp_2}$ mapping each copy to the corresponding member are added to the instances of the dimensions. Finally, the map $(\perp_1', \perp_2')$ is added to μ (Step 8) and the scheme $S = (L, \preceq)$ is modified accordingly (Steps 9 and 10). All of this guarantees the uniqueness of the bottom level for L without generating undesirable inconsistencies.

A matching tableau is then built on the (possibly modified) dimensions and a d-chase procedure is applied to the tableau (Step 11). If no contradiction is encountered (which corresponds to a test for consistency), the total projection of the resulting tableau over the scheme S generates the output dimensions (Steps 12 and 13).

Example 4.3 *Let us consider again the matching between dimensions in Figure 3 but assume that the level store does not map to the level shop. This means that the corresponding concepts are not related. It follows that the union of the schemes of the two dimensions produces two minimal levels. Then, the application of Algorithm 2 to this matching introduces two bottom levels below store and shop. The scheme of the dimension generated by the algorithm is reported in Figure 7. If the dimensions are populated by the member of Example 3.3, the output instance contains all the members occurring in the chased matching tableau reported in Example 3.4.*

We say that a dimension d *embeds* another dimension d' if there exists a DA expression E such that $E(d) = d'$. By construction and on the basis of the discussion above, we can state the following result.

Theorem 4.4 *The execution of Algorithm 2 over two dimensions d_1 and d_2 and a matching μ between them returns a new dimension d embedding both d_1 and d_2.*

Again, the complexity of the algorithm is bounded by the d-chase procedure that requires polynomial time in the size

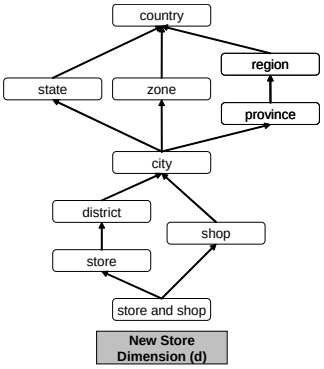


Figure 7. The dimension generated by Algorithm 2 on a variant of the matching in Figure 3

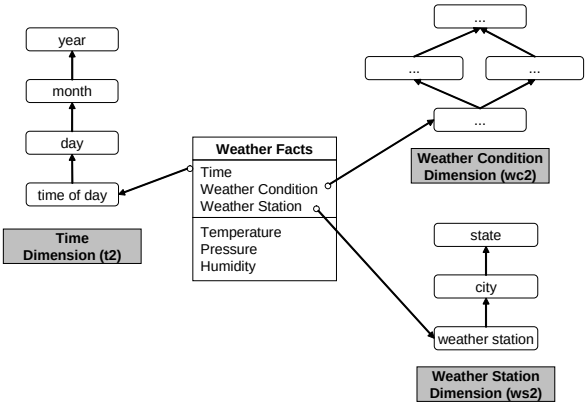


Figure 8. Weather data mart

of the dimensions involved. Hence, we can make for this algorithm the same considerations done for Algorithm 1.

5 Data mart integration

In this section we discuss, by means of some examples, how the techniques described in Section 4 can be used in data warehouse integration.

Drill-across queries have the goal of combining and correlating data from multiple data marts, and are especially useful to perform value chain analysis [9]. These queries are based on joining different data marts over common dimensions [9]. Since join operations combine relations on the basis of *common* data, the existence of shared information between data marts is needed in order to obtain meaningful results.

The loosely coupled approach supports drill-across queries between data marts, in that it aims at identifying the intersection between their dimensions. Actually, the proposed algorithm also checks for the *quality* of such intersection; in particular, dimension compatibility (e.g., the

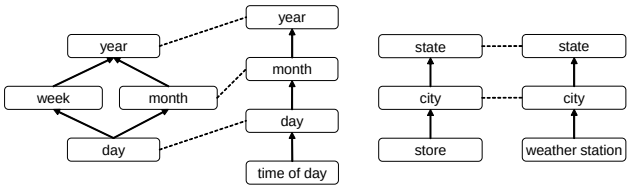


Figure 9. A matching between time and location dimensions

existence of a "perfect" intersection). As discussed in [5], this is a necessary condition for obtaining meaningful results when aggregations must be computed over data marts.

Assume, for instance, that we wish to integrate the Sales data mart reported in Figure 1 with the data mart storing weather information reported in Figure 8, in order to correlate sales of products with weather conditions. The integration between these data sources can be based on the matchings between the time ($t1$ and $t2$) and the location dimensions ($s1$ and $ws2$) as indicated in Figure 9.

The application of Algorithm 1 to this input checks for compatibility of dimensions and returns the following pairs of expressions. The first two expressions select the members in common in the time dimensions:

$$\pi_{day,month,year}(\sigma_{\mathrm{day}_{t1} \cap \mathrm{day}_{t2}}(t1)),$$
$$\psi_{day}(\sigma_{\mathrm{day}_{t1} \cap \mathrm{day}_{t2}}(t2)).$$

where $\mathrm{day}_{t1} \cap \mathrm{day}_{t2}$ denotes the days in common that make the matching between $t1$ and $t2$ perfect. The other pair of expressions select the members in common in the location dimensions:

$$\psi_{city}(\sigma_{\mathrm{city}_{s1} \cap \mathrm{city}_{ws2}}(s1)).$$
$$\psi_{city}(\sigma_{\mathrm{city}_{s1} \cap \mathrm{city}_{\mathrm{ws2}}}(ws2)).$$

where $\mathrm{city}_{s1} \cap \mathrm{city}_{ws2}$ denotes the cities in common that make the matching between $s1$ and $ws2$ perfect.

It turns out that we can join the two data marts to extract daily and city-based data, but hourly or store-based data can not be computed. Moreover, if we apply the above expressions to the underlying dimensions before executing the drill-across operation we prevent inconsistencies in subsequent aggregations over the result of the join. It follows that drill-across queries can be defined over the virtual view shown in Figure 10.

The tightly coupled approach aims at combining data from different dimensions, intuitively, by computing their union rather than their intersection. This can be useful when we need to reconcile and merge two data marts that have been developed independently.

Consider again the example above. If we apply Algorithm 2 over the time and location dimensions and the

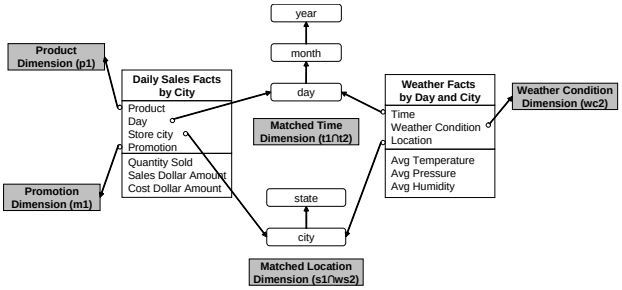


Figure 10. A virtual view over the common portion of the dimensions

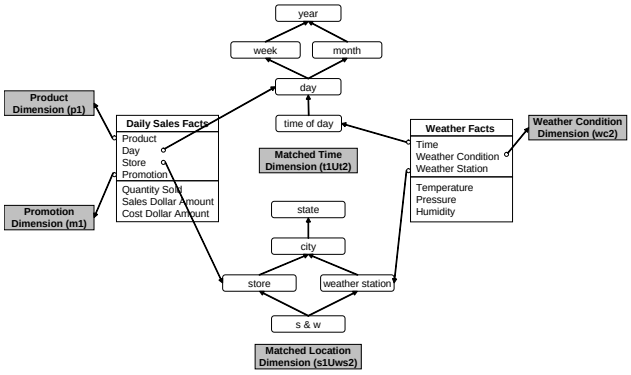


Figure 11. A materialized view over the merged dimensions

matchings in Figure 9, we generate two new dimensions that can be materialized and used for both data marts. We can then refer to the homogenous scheme reported in Figure 11 to perform drill-across queries.

Another application of the second approach is when we wish to extend local dimensions with data from external dimensions, but ignoring remote factual data, to extend local querying capabilities. For instance, to specify further selections and groupings (as suggested in [14]).

For example, consider again the Sales data mart. It could be integrated with an external and more sophisticated location dimension to select, for instance, sales in cities having more than 100.000 inhabitants.

6 Conclusion

We have proposed in this paper a number of concepts and techniques for the integration of heterogeneous multidimensional databases. We have first addressed the problem from a conceptual point of view, by introducing the desirable properties of coherence, soundness and consistency that "good" matchings between dimensions should enjoy.

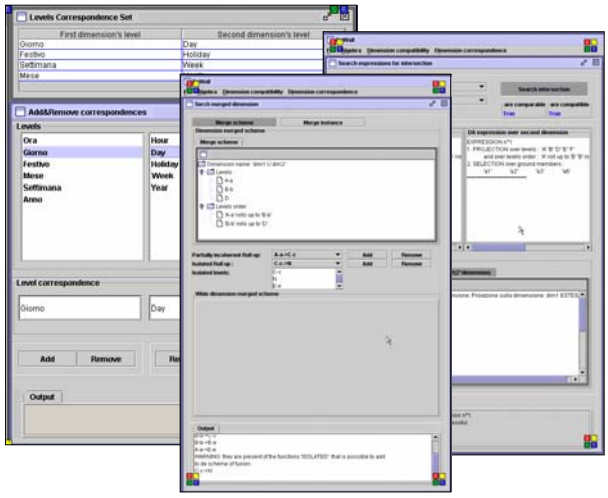


Figure 12. A prototype of the system

We have then presented two practical approaches to the problem that refer to the different scenarios of loosely and tightly coupled integration. We have shown that, if possible, both approaches guarantee the fulfillment of the above properties. To this end, we have introduced a practical tool, the chase of dimensions, that can be effectively used in both approaches to compare the content of the dimensions to integrate.

To test our approach, we have designed and developed the first release of an interactive tool for the integration of multidimensional databases, called DaWaII (for Data Warehouse IntegratIon), that implements the proposed techniques. Specifically, this tool is able to: (i) access data marts stored in a variety of systems (DB2, Oracle, SQL Server, among others); (ii) extract from these systems metadata describing cubes and dimensions and translate these descriptions in $\mathcal{MD}$ format; (iii) specify by means of a graphical interface matchings between autonomous dimensions; (iv) test for coherence, consistency, and soundness of matchings; (v) generate the intersection between two dimensions, according to the loosely integration approach; and (vi) merge two dimensions, according to the the tightly integration approach. An hint of the graphical interface provided by this tool is reported in Figure 12.

We believe that the techniques presented in this paper can be generalized to much more general contexts in which, similarly to the scenario of this study, we need to integrate heterogenous sources and we possess a taxonomy of concepts that describe their content. As a matter of fact, we note that dimensions have structural and functional similarities with *ontologies*, which provide descriptions of concepts in a domain and are used to share knowledge. It turns out that some of the notions and the techniques presented here can provide a contribution to the problem of integrat-

ing generic information sources using ontologies. This is subject of current investigation.

Acknowledgements

We would like to thank Ivan Panella, who is actively involved in the development of DaWaII. Moreover, we would like to thank the anonymous referees for their helpful comments and suggestions.

References

[1] A. Abelló, J. Samos, and F. Saltor. On relationships Offering New Drill-across Possibilities. In *ACM Fifth Int. Workshop on Data Warehousing and OLAP (DOLAP 2002)*, pages 7–13, 2002.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone. *Database Systems: concepts, languages and architectures*. McGraw-Hill, 1999.

[4] L. Cabibbo and R. Torlone. A logical approach to multidimensional databases. In *Sixth Int. Conference on Extending Database Technology (EDBT'98), Springer-Verlag*, pages 183–197, 1998.

[5] L. Cabibbo and R. Torlone. On the Integration of Autonomous Data Marts. In *16th Int. Conference on Scientific and Statistical Database Management (SSDBM'04)*, pages 223–234, 2004.

[6] A. Elmagarmid, M. Rusinkiewicz, and A. Sheth. *Management of Heterogeneous and Autonomous Database Systems*. Morgan Kaufmann, 1999.

[7] R. Hull. Managing Semantic Heterogeneity in Databases: A Theoretical Perspective. In *16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems*, pages 51–61, 1997.

[8] M.R. Jensen, T.H. Møller, and T.B. Pedersen. Specifying OLAP Cubes on XML Data. *J. Intell. Inf. Syst.*, 17(2-3): 255–280, 2001.

[9] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Second edition, 2002.

[10] M. Lenzerini. Data Integration: A Theoretical Perspective. In *21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems*, pages 233-246, 2002.

[11] F. M. Malvestuto. The Derivation Problem for Summary Data. *ACM SIGMOD International Conference on Management of Data*, pages 82–89, 1988.

[12] R.J. Miller, M.A. Hernández, L.M. Haas, L. Yan, C.T.H. Ho, R. Fagin, and L. Popa. The Clio Project: Managing Heterogeneity. *SIGMOD Record*, 30(1): 78–83, 2001.

[13] R.J. Miller (editor). Special Issue on Integration Management. *IEEE Bulletin of the Technical Committee on Data Engineering*, 25(3), 2002.

[14] D. Pedersen, K. Riis, and T.B. Pedersen. XML-Extended OLAP Querying. In *Int. Conference on Scientific and Statistical Database Management (SSDBM'02)*, pages 195–206, 2002.

[15] T.B. Pedersen, A. Shoshani, J. Gu, and C.S. Jensen. Extending OLAP Querying to External Object Databases. In *Int. Conference on Information and Knowledge Management*, pages 405–413, 2000.

[16] R. Redner and H. Walker. Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review*, 26(2):195–239, 1984.

[17] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334-350, 2001.

[18] H. Sato. Handling Summary Information in a Database: Derivability. *ACM SIGMOD International Conference on Management of Data*, pages 98–107, 1981.

Session 8: Time/Object Queries

Using Multi-Scale Histograms to Answer Pattern Existence and Shape Match Queries over Time Series Data

Lei Chen and M. Tamer Özsu
University of Waterloo
School of Computer Science
{l6chen,tozsu}@uwaterloo.ca

Vincent Oria
New Jersey Institute of Technology
Dept. of Computer Science
oria@njit.edu

Abstract

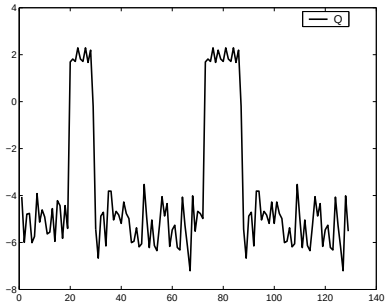
Similarity-based querying of time series data can be categorized as pattern existence queries and shape match queries. Pattern existence queries find the time series data with certain patterns while shape match queries look for the time series data that have similar movement shapes. Existing proposals address one of these or the other. In this paper, we propose multi-scale time series histograms that can be used to answer both types of queries, thus offering users more flexibility. Multiple histogram levels allow querying at various precision levels. Most importantly, the distances of time series histograms at lower scale are lower bounds of the distances at higher scale, which guarantees that no false dismissals will be introduced when a multi-step filtering process is used in answering shape match queries. We further propose to use averages of time series histograms to reduce the dimensionality and avoid computing the distances of full time series histograms. The experimental results show that multi-scale histograms can effectively find the patterns in time series data and answer shape match queries, even when the data contain noise, time shifting and scaling, or amplitude shifting and scaling.

1 Introduction

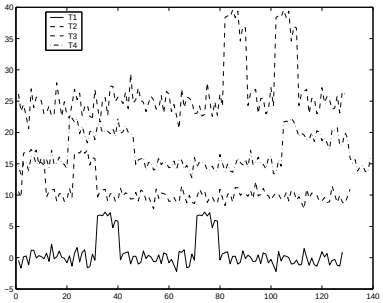
Similarity-based time series data retrieval has been studied in the database and knowledge discovery communities for several years, due to its wide use in various applications, such as financial data analysis [26], content-based video retrieval [17], and musical retrieval [28]. Typically, two types of queries on time series data are studied: pattern existence queries [2, 18, 19], and shape match queries [1, 6, 10].

In pattern existence queries, users are interested in the general pattern of time series data and ignore the specific details. For example,"Give me all the temperature data of patients in last 24 hours that have *two peaks*".

The example query is used to detect "goalpost fever", one of the symptoms of Hodgkin's disease, in the temperature time series data that contain peaks exactly twice within 24 hours [19]. Figure 1(a) shows an example time series with *two peaks*. Using this as query data, a *two peaks* existence query should retrieve various time series that contain *two peaks* as shown in Figure 1(b). Therefore, for pattern existence queries, the important thing is the existence of the specified pattern in time series data, regardless of where the pattern appears and how it appears.



(a) An example query time series with two peaks

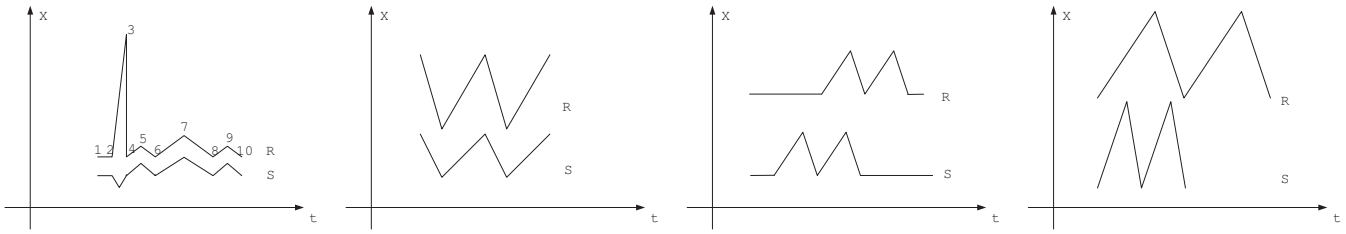


(b) Various time series data containing two peaks

Figure 1. A pattern existence query on two peaks

The retrieval techniques for pattern existence queries should be invariant to the following:

- **Noise.** In Figure 2(a), two similar time series R and S are shown. Point 3, which is likely a noise data point, will introduce a large difference when Euclidean distance is computed between two time series, possibly



(a) Two similar time series with noise (b) Two similar time series with amplitude scaling and shifting (c) Two similar time series with time shifting (d) Two similar time series with time scaling

Figure 2. The different factors that may affect the similarity between two time series

causing them to be treated as dissimilar.

- **Amplitude scaling and shifting.** In Figure 2(b), the two time series R and S are similar in terms of the pattern that they contain (they both have the "two bottom" pattern), but their amplitudes are different (in fact, $R = aS + b$ where a and b are scaling factor and shifting factor, respectively).
- **Time shifting.** In Figure 2(c), time series R and S record the same event (e.g. temperature data) but from different starting times; shifting R to the left on the time axis can align the shape with S. R and S are considered dissimilar if they are simply compared by the respective positions of the data points. However, it can be argued that R and S are similar because they contain the same pattern ("two peaks").
- **Time scaling.** In Figure 2(d), time series R and S have different sampling rates. However, both R and S contain "two peaks".

Several approximation approaches have been proposed to transform time series data to character strings over a discrete alphabet and apply string matching techniques to find the patterns [2, 18, 19]. However, the transformation process is sensitive to noise. Furthermore, because of the quantization of the value space for transforming time series data into strings, data points located near the boundaries of two quantized subspaces may be assigned to different alphabets. As a consequence, they are falsely considered to be different by string comparison techniques.

For shape match queries, users are interested in retrieving the time series that have similar movement shapes to the query data, e.g. "Give me all stock data of last month that is similar to IBM's stock data of last month". As shown in Figure 3, we are looking for the stock data within the boundaries that are described by two dashed curves. Most of the previous work focused on solving these types of queries, by applying distance functions such as Euclidean distance [1, 6, 10], DTW [3, 27] and LCSS [4, 22] to compute the distance between two data sequences.

There exist applications that require answers from both types of queries. An example is an interactive analysis of time series data. Users may be initially interested in retrieving all the time series data that have some specific pattern that can be quickly answered by pattern existence queries. They can then apply shape match queries to these results to retrieve those that are similar to a time series that is of interest. However, there are no techniques that have been developed to answer both pattern existence queries and shape match queries. Of course, two different techniques used to answer pattern existence queries and shape match queries can be applied to these applications together; however, different types of representation (e.g. symbolic representation and raw representation), distance functions (e.g. string matching and Euclidean distance), and indexing structures (e.g. suffix tree and R*-tree) will require storing redundant information and cause difficulties in improving the retrieval efficiency.

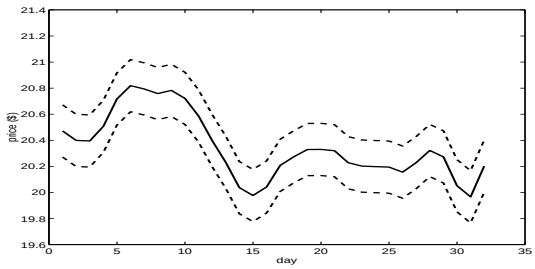


Figure 3. A query example on stock time series data

In this paper, based on the observation that data distributions can capture patterns of time series data, we propose a multi-scale histogram-based representation to approximate time series data that is invariant to noise, amplitude shifting and scaling, and time shifting and scaling. The histogram representation can answer both pattern existence queries and shape match queries, while multiple scales offer users flexibility to search time series data with different precision levels (scales). The cumulative histogram distance [20], which is closer to perceptual similarity than L_1-norm, L_2-norm, or weighted Euclidean distance, is used to measure the similarity between two time series histograms. We prove that distances of lower scale time series histograms are lower bounds of higher scale distances, which guarantees that using multi-step filtering process in answering shape match queries will not introduce false dismissals. In order to reduce the computation cost in computing the dis-

tances of full time series histograms, we use the distances between averages of time series histograms as the first step of the filtering process, since the distances between averages are lower bounds of distances of time series histograms at scale 1. We investigate two different approaches in constructing histograms: *equal size bin* and *equal area bin*. The experimental results show that our histogram-based representation, together with the cumulative histogram distance measure, can effectively capture the patterns of time series data as well as the movement shapes of time series data. Finally, We compare our representation with another multi-scale representation, namely wavelets, and conclude that wavelets are not suitable to answer pattern existence queries and are not robust to time shifting when used to answer shape match queries.

The rest of the paper is arranged as follows: Section 2 presents, as background, concepts of time series histograms and two variations of them. We present multi-scale time series histograms in Section 3. In Section 4, we present experimental results using multi-scale time series histograms on finding patterns and answering shape match queries, followed, in Section 5, by a comparison of our representation with wavelet. Related work are addressed in Section 6. We conclude in Section 7 and indicate some further work.

2 Time Series Histograms

A *time series* R is defined as a sequence of pairs, each of which shows the value (r_i) that is sampled at a specific time denoted by a timestamp (t_i): $R = [(r_1, t_1), \ldots, (r_N, t_N)]$ where N, the number of data points in R, is defined as the *length* of R. We refer to this sequence as the *raw representation* of the time series data.

Given a set of time series data $\mathcal{D} = \{R_1, R_2, \ldots, R_L\}$, each time series R_i is normalized into its *normal form* using its mean (μ) and variance (σ) [8]:

$$Norm(R) = [(t_1, \frac{r_1 - \mu}{\sigma}), \ldots, (t_N, \frac{r_N - \mu}{\sigma})] \quad (1)$$

The similarity measures computed from time series normal form are invariant to amplitude scaling and shifting.

Time series histograms are developed in the following way. Given the maximum ($max_{\mathcal{D}}$) and minimum ($min_{\mathcal{D}}$) values of normalized time series data, the range $[min_{\mathcal{D}}, max_{\mathcal{D}}]$ is divided into τ disjoint equal size subregions, called *histogram bins*. Given a time series R, its histogram H_R can be computed by counting the number of data points h_i ($1 \leq i \leq \tau$) that are located in each histogram bin i: $H_R = [h_1, \ldots, h_\tau]$.

We normalize the time series histogram by dividing the value of each histogram bin by the total number of data points in the time series. Since a time series histogram is computed from the normal form of a time series, the distance that is computed from two time series histograms are invariant to amplitude scaling and shifting. Furthermore, because time series histograms ignore the temporal information, they are also robust to time shifting and scaling. For example, in Figures 2(c) and 2(d), the histogram of normalized R are similar to that of S. Moreover, since time series histograms show the whole distribution of the data, and noise only makes up a very small portion, comparisons based on histograms can remove the disturbance caused by noise. Therefore, time series histograms are ideal representations for answering pattern existence queries.

L_1-norm or L_2-norm [21] can be used to measure the similarity between two histograms. However, these do not take the similarity between time series histogram bins into consideration, which may lead to poor comparison results. Consider three histograms H_R, H_S and H_T representing three time series of equal length. Assume that H_R and H_S have the same value on consecutive bins and H_T has the same value in a bin which is quite far away from the bins of H_R and H_S. L_1 and L_2-norm distances between any two of these three histograms are equal. However, for answering shape match queries, H_R is closer to H_S than it is to H_T. Even for pattern existence queries, data points which are located near the boundary of two histogram bins should be treated differently compared to those points that are far apart, which is not considered by L_1-norm and L_2-norm.

A weighted Euclidean distance can be used to compute the distance between two histograms [9]. Given two time series R and S, the *weighted Euclidean distance* (WED) between their time series histograms H_R and H_S is:$WED(H_R, H_S) = Z^T A Z$ where $Z = (H_R - H_S)$, Z^T is the transpose of Z, and $A = [a_{ij}]$ is a similarity matrix whose element $a_{ij} = 1 - |j - i|/\tau$ (where τ is the number of bins) denotes similarity between two time series histogram bins i and j. As a_{ij} gets larger, bins i and j become more similar.

Compared to L_1-norm and L_2-norm, WED underestimates distances because it tends to estimate the similarity of data distribution without a pronounced mode [20].

Cumulative histogram distances [20] overcome the shortcomings of L_1-norm, L_2-norm and WED, the similarity that is measured by cumulative histogram distance is closer to the perceptual similarity of histograms. Therefore, we use this distance function to measure the similarity between two time series histograms. Given a time series R and its time series histogram $H_R = [h_1, h_2, \ldots, h_\tau]$, where τ is the number of bins, the *cumulative histogram* of R is:$\hat{H}_R = [\hat{h}_1, \hat{h}_2, \ldots, \hat{h}_\tau]$ where $\hat{h}_i = \sum_{j \leq i} h_j$. Given two cumulative histograms $\hat{H}_R$ and $\hat{H}_S$ of two time series R and S, respectively, the cumulative histogram distance (CHD) is defined as:

$$CHD(\hat{H}_R, \hat{H}_S) = \sqrt{\hat{Z}^T \hat{Z}} \quad (2)$$

where $\hat{Z} = (\hat{H}_R - \hat{H}_S)$.

The algorithm for answering pattern existence queries using CHD is then simple. For each $\hat{H}_i$ of R_i, if $CHD(\hat{H}_i, \hat{H}) \leq \epsilon$, then R_i is in the result set (ϵ is a matching threshold). Later, we experimentally compare the effectiveness of WED and CHD in answering pattern existence queries. In this algorithm, a matching threshold (ϵ) has to be set to determine whether the examined time series contains a pattern similar to that of the query time series. In our experiments, we find a suitable threshold based on the histogram of the query time series.

So far, we defined a histogram with equal size bins. However, values of time series data normally are not uniformly distributed, leaving a lot of bins empty. In such cases, computing the distances between two time series histograms that contain many zeros is not helpful to differentiate the corresponding time series data.

It has been claimed [14] that distributions of most time series data follow normal distribution, which we have also verified on the data sets that we use in our experiments. Consequently, instead of segmenting the value space into τ equal size sub-regions, we segment the value space into sub-regions (called sub-spaces) that have the same area size under the normal distribution curve of that data. The boundary of each subspace can be computed as follows. Assuming that $h_{i,l}$ is the lower bound of subspace i and $h_{i,u}$ is the upper bound: $\int_{min_{\mathcal{D}}}^{max_{\mathcal{D}}} p(x)dx = \sum \int_{h_{i,l}}^{h_{i,u}} p(x)dx$, where $p(x)$ is the normal distribution function, $1 \leq i \leq \tau$, $h_{1,l} = min_{\mathcal{D}}$, and $h_{\tau,u} = max_{\mathcal{D}}$. Even though we use the normal distribution function to create equal area bin histogram bins, the idea can be easily extended to other data distributions.

With equal area bin segmentation, we assign equal probability to each histogram bin into which data points of time series fall. For example, for the "cameramouse" data that we used in our experiment, the average filling ratio of 16 equal area bin histograms is about 98% (the filling ratio is defined as the number of non-empty bins to the total number of bins). However, it is only 40% for the 16 bin equal size bin histograms. In our experiments, we compare the effectiveness of equal size bin histograms and equal area bin histograms in terms of classification accuracy.

3 Multi-scale Histograms

The time series histograms, as defined in the previous section, give a global view of the data distribution of time series data. However, they do not consider the order of values in the sequence.

A multi-scale representation of a time series histogram is designed for better discrimination of time series data based on their order details to facilitate shape match queries. Given a time series R of length N, it can be equally divided into two segments, and each segment can be recursively divided into two, and so on. For each segment, its time series histogram can be computed and all these histograms form the multi-scale time series histograms of R. The number of levels (scales) is controlled by a single parameter, δ, that is the *precision level* (i.e. *scale*) of the histogram. For $\delta = 1$, the histogram covers the entire time series, as defined in the previous section. If further precision is required, one can set $\delta > 1$, which would segment the time series data into $2^{(\delta-1)}$ equal length subsequences and histograms are computed for each segment.

With multi-scale time series histograms, the shape match queries can be answered at several precision levels. It has to be guaranteed that similar time series at a higher scale will be also identified as similar at a lower scale. In order to guarantee this property, the cumulative histogram distance must be formulated in such a way that the distance at the lower scale is the lower bound of the distance at the higher scale. The average of the cumulative histogram distances is used as the result of comparison at scale δ, which, as proven below, satisfies this property.

Definition 3.1 *Given two time series data R and S, let their $\delta \geq 1$ scale histograms be $H^i_{R,\delta}$ and $H^i_{S,\delta}$, respectively. where $1 \leq i \leq 2^{\delta-1}$. The CHD at scale δ is:*

$$CHD_\delta = \frac{\sum_{i=1}^{2^{\delta-1}} CHD(\hat{H}^i_{R,\delta}, \hat{H}^i_{S,\delta})}{2^{\delta-1}} \tag{3}$$

where $\hat{H}^i_{R,\delta}$ ($\hat{H}^i_{S,\delta}$) denotes the cumulative histograms of i^{th} segment of R (S) at scale δ.

Theorem 3.1 *In a δ-level multi-scale time series histogram, if CHD_l denotes the cumulative histogram distance between two time series histograms at scale l, then*

$$CHD_{l-1} \leq CHD_l \tag{4}$$

where $2 \leq l \leq \delta$.

We only prove the base case, which is $CHD_1 \leq CHD_2$; the general case can be proven by induction.

Proof. Given two time series data R and S, their cumulative histograms at scale i ($i \geq 1$) are denoted as: $\hat{H}_{R,i}$ and $\hat{H}_{S,i}$, respectively. Then: $CHD_1 = \sqrt{(\hat{H}_{R,1} - \hat{H}_{S,1})^T(\hat{H}_{R,1} - \hat{H}_{S,1})}$ and $CHD_2 = \frac{1}{2}\sum_{i=1}^{2} \sqrt{(\hat{H}^i_{R,2} - \hat{H}^i_{S,2})^T(\hat{H}^i_{R,2} - \hat{H}^i_{S,2})}$. Define $\|X\| = \sqrt{X^T X}$ and $\|X - Y\| = \sqrt{(X-Y)^T(X-Y)}$, where X and Y are τ dimensional vectors. Then, $CHD_1 = \|\hat{H}_{R,1} - \hat{H}_{S,1}\|$ and $CHD_2 = \frac{1}{2}\sum_{i=1}^{2} \|\hat{H}^i_{S,2} - \hat{H}^i_{R,2}\|$.

$$\begin{aligned} \|X + Y\| &= \sqrt{\sum_{i=1}^{n}(x_i + y_i)^2} \\ &= \sqrt{\|X\|^2 + \|Y\|^2 + 2\sum_{i=1}^{n}(x_i y_i)} \end{aligned}$$

Using Cauchy's inequality, which is

$$(\sum_{i=1}^{n}(x_i y_i))^2 \leq \sum_{i=1}^{n}(x_i)^2 \sum_{i=1}^{n}(y_i)^2 = \|X\|^2\|Y\|^2$$

we get

$$\|X+Y\| \leq \|X\| + \|Y\| \quad (5)$$

It is known that $\hat{H}_{R,1} = \frac{1}{2}(\hat{H}^1_{R,2} + \hat{H}^2_{R,2})$ and $\hat{H}_{S,1} = \frac{1}{2}(\hat{H}^1_{S,2} + \hat{H}^2_{S,2})$. Thus:

$$\begin{aligned}\|\hat{H}_{R,1} - \hat{H}_{S,1}\| &= \|\frac{1}{2}\sum_{i=1}^{2}(\hat{H}^i_{R,2}) - \frac{1}{2}\sum_{i=1}^{2}(\hat{H}^i_{S,2})\| \\ &\leq \frac{1}{2}\sum_{i=1}^{2}\|\hat{H}^i_{R,2} - \hat{H}^i_{S,2}\| \text{(from 5)}\end{aligned}$$

Therefore, $CHD_1 \leq CHD_2$. □

As we stated earlier, time series histograms at higher scales have better discrimination power; however, the computation of CHD at higher scales is more expensive than those at lower scales. Fortunately, with the lower bound property of CHD at each scale (Theorem 1), when we need to answer a shape match query at high scale l, instead of directly computing the CHD at scale l,we can start computing the CHD for each candidate time series at scale 1, and then scale 2 and so on [13]. This multi-step filtering strategy will not introduce false dismissals.

For both pattern existence queries and shape match queries, directly comparing τ-dimensional time series histograms is computationally expensive, even with the help of multidimensional access methods such as the R-tree. Since τ is higher than 12-16 dimensions, the performance of using an indexing structure will be worse than that of sequential scan [25]. Therefore, we use the averages of time series cumulative histograms to avoid comparisons on full cumulative histograms.

Definition 3.2 *Given a time series R and its cumulative histogram at scale level 1, $\hat{H}_{R,1} = [\hat{h}_{R,1}, \hat{h}_{R,2}, \ldots, \hat{h}_{R,\tau}]$, where τ is the number of histogram bins, the average of cumulative histogram is:*

$$\hat{H}^{avg}_{R,1} = \frac{\sum_{i=1}^{\tau}\hat{h}_{R,i}}{\tau} \quad (6)$$

Definition 3.3 *Given two time series R and S, let $\hat{H}_{R,1}$ and $\hat{H}_{S,1}$ be their cumulative histograms at scale level 1. The distance between averages of cumulative histograms is:*

$$ACHD = \sqrt{\tau(\hat{H}^{avg}_{R,1} - \hat{H}^{avg}_{S,1})^2} \quad (7)$$

The averages of time series cumulative histograms are one dimensional data, which means that a simple B+-tree can be used to improve the retrieval efficiency. Moreover, based on the definition of ACHD, the time series data retrieved by comparing ACHD are guaranteed to include all the time series data that should be retrieved by comparing cumulative histograms of time series data at scale 1. This is stated in the following theorem.

```
Procedure shape match queries{
  /* An example time series Q, its δ levels cumulative histograms
     a matching threshold ε, precision level δ*/
  (1) for each average of cumulative histograms Ĥ_i^avg of R_i{
  (2)   compute ACHD between Ĥ^avg and Ĥ_i^avg
  (3)   if (ACHD ≤ ε) { /* need to check */
  (4)     insert the time series id i into the result list resultlist_0
        } /* end-if, line 3 */
    } /* end-for, line 1 */
  (5) j = 1
  (6) do {
  (7)   for each i in resultlist_{j-1} {
  (8)     if (CHD_j(Ĥ_i, Ĥ) ≤ ε){
  (9)       insert the time series id i into the result list resultlist_j
          } /* end-if, line 8 */
        } /* end-for, line 7 */
  (10) j = j + 1
  (11) }while (j == δ + 1) or (resultlist_{j-1} is empty)
  (12) if (resultlist_{j-1} is empty)
  (13) return NULL
  (14) else return the result list resultlist_δ
}
```

Figure 4. The algorithm for answering shape match queries with multi-scale filtering

Theorem 3.2 *For any two τ-dimensional time series cumulative histograms $\hat{H}_{R,1}$ and $\hat{H}_{S,1}$ at scale 1, $ACHD \leq CHD_1$.*

Proof. Given two histograms of scale 1 $\hat{H}_{R,1}$ and $\hat{H}_{S,1}$,$CHD_1^2 = \sum_{i=1}^{\tau}(\hat{h}_{R,i} - \hat{h}_{S,i})^2$ and $ACHD^2 = \frac{(\sum_{i=1}^{\tau}\hat{h}_{R,i} - \sum_{i=1}^{\tau}\hat{h}_{S,i})^2}{\tau}$

Define $X = \hat{H}_{R,1} - \hat{H}_{S,1}$, where $x_i = \hat{h}_{R,i} - \hat{h}_{S,i}$ and $1 \leq i \leq \tau$. Therefore, the only thing that needs to be proven is: $\sum_{i=1}^{\tau} x_i^2 \geq \frac{(\sum_{i=1}^{\tau} x_i)^2}{\tau}$. According to Arithmetic-Geometric Mean inequality:

$$\begin{aligned}\frac{(\sum_{i=1}^{\tau} x_i)^2}{\tau} &\leq \frac{(\sum_{i=1}^{\tau}(x_i)^2 + \sum_{i=1}^{\tau-1}\sum_{i=1}^{\tau}(x_i)^2)}{\tau} \\ &= \sum_{i=1}^{\tau}(x_i)^2 \quad \square\end{aligned}$$

Therefore, instead of directly computing the cumulative histogram distances, we can first compute the distance between the averages of two time series cumulative histograms to prune false alarms from the database. Averages can be considered as the first filter when we use multi-step filtering to answer an shape match query at a higher scale l. The algorithm for multi-step filtering is given in Figure 4. We also show experimentally the pruning power of averages of cumulative histograms.

4 Experiments and Discussion

In this section, we present the results of experiments that we have conducted to evaluate the efficacy and robustness

of the histogram-based similarity measure and the matching algorithm. All programs are written in C and experiments are run on a Sun-Blade-1000 workstation under Solaris 2.8 with 1GB of memory.

4.1 Efficacy and Robustness of Similarity Measures

Experiment 1. This experiment is designed to test how well the cumulative histogram distances perform in finding patterns in time series data. We first compare our approach with Shatkay's algorithm [19] on labelled time series data sets: Cylinder-Bell-Funnel (CBF). In Shatkay's algorithm, first, a best fitting line algorithm is used to detect the movement slope of segments of time series data. Then, the slope of each line segment is mapped to a symbol according to the predefined mapping tables and consecutive symbols are connected together to form a string. Finally, a string matching algorithm is used to find the matches between query regular expression and the converted strings. Another related work [18] requires users to specify, in addition to the pattern, the "unit length" (the length of time series data) in which the specified pattern may appear. It is quite difficult for users to know the "unit length" beforehand if they only want to check for the existence of a pattern. Therefore, we do not consider this one. The reason for using CBF data set is that it contains very simple distinct patterns, allowing a direct comparison of the techniques. The CBF data set has three distinct classes of times series: cylinder, bell and funnel. We generate 1000 CBF data sets with 100 examples for each class. Since the general shape of bell and funnel are treated as similar (both of them contain a *peak*), only cylinder and bell data sets are used to test existence queries.

In order to test robustness of the similarity measures, we added random Gaussian noise and time warping to both data sets using a modified version of the program in [23]. The modification includes the addition of non-interpolated noise and large time warping ($20-30\%$ of the series length) since pattern existence queries only involve the existence of a given pattern.

We use the first scale time series histograms to search the patterns in the time series data. We generated another "pure" CBF data set of size 150 as query data, where each class contains 50 examples. The histograms are also computed from the query data set. We use the well-known precision and recall to measure our retrieval results. Precision measures the proportion of correctly found time series, while recall measures the proportion of correct time series that are detected. In this experiment, we need to determine the matching threshold. We tested several values and found that half the cumulative histogram distance between the query histogram and an empty histogram gives the best results. Besides using cumulative histogram distance, we also run the program with weighted Euclidean distance to compare their effectiveness.

We run the query 50 times and average the results, as shown in Table 1, where histogram experiments are parameterized by the number of bins. In these experiments, results are reported as WED/CHD values. Even though Shatkay's approach achieves relatively high precision, its recall is very low; this is the effect of noise. Our histogram-based approach (both WED and CHD) can achieve relatively high precision and recall, which confirms that our approach is suitable for answering pattern existence queries. From Table 1, we also find that dividing the value space into too many sub-regions (histogram bins) does not improve the results significantly; we can achieve reasonably good results around 16 bins. As we expect, the results also show that CHD performs better than WED, this is because WED overestimates neighborhood similarity.

	Cylinder		Bell	
	precision	recall	precision	recall
Shatkay	81	44	75	31
his(8)	72/80	88/91	63/ 70	71/ 74
his(16)	72/81	90/92	78/ 84	80/ 85
his(32)	75/81	88/90	79/ 85	85/ 87
his(64)	72/80	88/90	79/ 83	85/ 88

Table 1. Comparison on pattern existences queries

Experiment 2. This experiment is designed to check the effectiveness and robustness of multi-scale histograms in answering shape match queries. According to a recent survey on time series data [11], the efficacy of a similarity measure for shape match queries can be evaluated by classification results on labelled data sets. We compare the classification error rates in results produced by CHD to those of DTW and LCSS by evaluating them on the Control-Chart (CC) data set. The comparison is done against DTW and LCSS because these two can also handle time shifting or noise disturbances. The classification error rate is defined as a ratio of the number of misclassified time series to the total number of the time series. There are six different classes of control charts. Each class contains 100 examples. We later tested our techniques using more complicated data sets. We added non-interpolated noise and time warping ($10-20\%$ of the series length) to test the robustness of cumulative histogram distance in answering shape match queries. For simplicity, we carried out a simple classification using 1-Nearest Neighbor with three similarity measures and checked the classification results using the "leave one out" verification mechanism.[1]

We repeat the experiment on all the time series of CC

[1]The "leave one out" verification mechanism takes one sample data from a class and finds a nearest neighbor of the data in the entire data set by applying a predefined distance metric. If the found nearest neighbor belongs to the same class as the sample data, it is a hit, otherwise, it is a miss.

data set. The error rates using DTW and LCSS are 0.12 and 0.16, respectively. It has been claimed that LCSS is more accurate than DTW [22]. However, our experiments could not replicate this result. The possible reason for this discrepancy is the difference in the choice of the matching threshold value. We report comparable results using the cumulative histogram distances in Table 2. We run the experiment with different number of bins (τ) and scales (δ) using CHD on time series histograms with equal size bin.

Comparing the error rates of DTW (0.12), LCSS (0.16) and those of Table 2, we observe that CHD performs better than the other two similarity measures in answering shape match queries. From Table 2, we find an interesting fact that very high scales (i.e., high values of δ) may lead to worse classification results. This is because the higher the scale, the more temporal details will be involved in computing CHD, and this causes time series histograms at that scale to be more sensitive to time shifting and scaling. Another fact demonstrated in Table 2 is that higher number of bins may not lead to more accurate classification results. This is because as the number of bins increases, more detailed information will be captured in time series histogram, including noise. These characteristics of multi-scale time series histograms exactly fit our needs, since they suggest that we only need to check the first few scale histograms with small number of histogram bins. However, the improvements over DTW or LCSS as reported in Table 2 are modest, especially for the first few scale histograms.

	number of bins τ			
scale δ	8	16	32	64
1	0.62	0.59	0.56	0.53
2	0.22	0.16	0.12	0.13
3	**0.14**	**0.10**	**0.11**	**0.11**
4	0.20	0.13	0.14	0.15
5	0.24	0.19	0.20	0.20

Table 2. Error rates with equal size bin histograms

To better understand these results, we investigated the filling ratio of equal size bin histograms. We found that nearly 50% of the bins are empty! Consequently, CHD between two time series histograms is not able to distinguish them properly, since most of the bins are identical! Therefore, we run the same experiment with equal area bin histograms. Table 3 reports the results.

	number of bins τ			
scale δ	8	16	32	64
1	0.15	0.11	0.12	0.13
2	0.10	0.08	0.07	0.06
3	**0.07**	**0.05**	**0.06**	**0.06**
4	0.10	0.08	0.08	0.09
5	0.18	0.15	0.17	0.17

Table 3. Error rates with equal area bin histograms

The results of Tables 2 and 3 demonstrate that the improvement when equal area bin is used is nearly 2 times for CC data! The filling ratio of time series histogram is around 80%. Table 3 also shows that using only the first few scales (e.g. $\delta = 3$), provides reasonably high accuracy.

Experiment 3. In this experiment, we test the matching accuracy of multi-scale time histograms versus DTW and LCSS on classifying time series data with more complicated shapes. Classifying these types of time series requires matching on temporal details of the data. We evaluate CHD, DTW and LCSS by two labelled trajectory data sets that are generated from the Australian Sign Language (ASL)[2] data and the "cameramouse" [7] data. Only x positions are used for both data sets. We first select a "seed" time series from each of the two data sets and then create two additional data sets by adding interpolated Gaussian noise and small time warping (5-10% of the time series length) to these seeds [23]. The ASL data set from UCI data consists of samples of Australian Sign Language signs. Various parameters were recorded as a signer was signing one of 95 words in ASL. We extracted one recording from each of 10 words[3]. The "cameramouse" data set contains 15 trajectories of 5 words (3 for each word) obtained by tracking the finger tip as people write various words. We use all the trajectories of each word as the seeds. The data set that is generated from seeds of "cameramouse" contains 5 classes and 30 examples of each class, while the one from ASL seeds contains 10 classes and each class has 10 examples. We use the same classification and verification algorithms as in the second experiment. Based on the observation of the second experiment, we use time series histogram with equal area bin. The error rate of using DTW for ASL and "cameramouse" was 0.11 and 0.08, respectively. Comparable values for LCSS were 0.29 and 0.30. Table 4 reports the error rates of CHD.

Table 4 shows that CHD again achieves better classification result. For both data sets, CHD can achieve 0 error rate. A surprising observation is that even at lower scales (e.g. $\delta = 2$), CHD can achieve better results than DTW and LCSS. Through the investigation of the filling ratios of both histograms, we find that the number of empty bins only accounts for less than 5% of the total number of bins. The standard deviations of both data sets are also very high with respect to their mean values, which indicates that data points are widely spread in their value space. Compared with results of the second experiment, we conclude that CHD on time series data whose data points are widely distributed in their value space (higher histogram filling ratio) can achieve better classification accuracy at lower scales.

[2] http://kdd.ics.uci.edu

[3] "seed" time series of ASL data: "Norway", "cold", "crazy", "eat", "forget", "happy", "innocent", "later", "lose" and 'spend"[23].

	ASL DATA				Cameramouse DATA			
	number of bins τ				number of bins τ			
scale	8	16	32	64	8	16	32	64
1	0.12	0.11	0.12	0.12	0.13	0.13	0.07	0.07
2	**0.06**	**0.05**	**0.04**	**0.04**	**0.04**	**0.03**	**0.02**	**0.02**
3	0.06	0.06	0.04	0.04	0.03	0.02	0.02	0.02
4	0.03	0.04	0.04	0	0.04	0.01	0	0.01
5	0.06	0.06	0.04	0.02	0.04	0.04	0.02	0.04

Table 4. Error rates using CHD on time series histograms with equal area bin

4.2 Efficiency of Multi-Step Filtering

Experiment 5. Theorem 1 in Section 3 established that multi-step filtering using multi-scale time series histograms will not introduce false alarms. However, the question remains as to how many histogram comparisons are saved using multi-step filtering? We use a real stock data set that contains 193 company stocks' daily closing price from late 1993 to early 1996, each consisting of 513 values [24]. We use each time series as a "seed" and create a new data set by adding interpolated Gaussian noise and small time warping (2-5% of the time series length) to each seed. The new data set contains 1930 time series (each seed is used to create 10 new time series). We randomly select a time series and conduct a range query at precision level 4. We compute the number of comparisons that are needed for searching using only level 4, using level 1 and then jumping to 4, and using all 4 levels on different range thresholds. We run the experiments 100 times and the average results are reported in Table 5. Step-by-step filtering is clearly the best strategy

	level 1 only	level 1 and level 4	all 4 levels
$\epsilon = 0.5$	11580	13498	17190
$\epsilon = 0.2$	11580	12760	7590
$\epsilon = 0.1$	11580	6838	3668
$\epsilon = 0.05$	11580	2356	2072

Table 5. Comparisons on different filtering approaches

to reduce the total number of comparisons for small thresholds. For large thresholds, directly computing the distance at a higher level is a better choice in terms of the number of comparisons. However, large thresholds are not very useful for finding the desirable results, since a larger portion of the data in the database will be returned.

Experiment 6. The number of comparisons needed for computing the histogram distance only relates to the CPU computation cost. However, the I/O cost for sequentially reading in the data files becomes a bottleneck as the database size grows. If we can filter out the false alarms without reading in the data files, a significant speed-up will be achieved. For our time series histograms, we store their averages separately from histograms. The distances between the average cumulative histogram of query data and those of stored data are first computed to remove the possible false alarms. Therefore, the pruning power of average cumulative histograms is quite important. The pruning power (P) is defined as the fraction of the database that must be examined before we can guarantee that the nearest match to 1-Nearest Neighbor query is found [10]. Figure 5 shows the pruning power of CHD with 16 bins (based on the previous experimental results, CHD can already achieve reasonable good results when the histogram bin size is 16). Because "cameramouse" and ASL data sets are small, we did not include them in this experiment. Figure 5 demonstrates that using the distance of average time series histograms, we can remove nearly 40% of the false alarms, which is helpful when the database size becomes large.

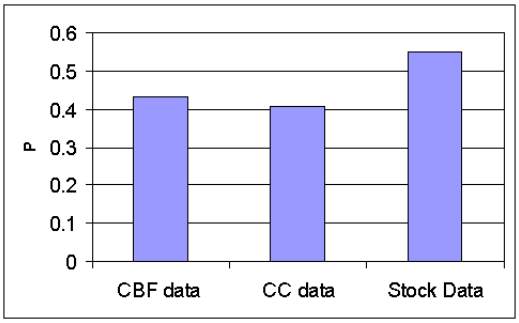


Figure 5. The punning power of averages of histograms

5 Comparison to Wavelets

In this section, we compare our representation with another multi-scale representation, wavelets. Wavelets have been widely used as a multi-resolution representation for image retrieval [16]; they have also been used as a dimensionality reduction technique in shape match queries in which Euclidean distance is used as the distance function [12]. Different from Fourier transform, wavelets can transform time series data into a multi-scale representation, where lower frequency bands are represented in lower scales and higher frequency bands are represented in higher scales. In this experiment, we used Euclidean distance to measure the similarity between two Haar wavelet coefficients (we used Haar wavelet, since it is the most popular wavelet transformation and has been used for similarity-based time series retrieval [12]).The same CBF data set used in the first experiment is used to measure the efficacy using different number of wavelet coefficients to answer pattern

existence queries. Table 6 reports the results.

	Cylinder		Bell	
	precision	recall	precision	recall
8	49	97	48	96
16	45	61	46	67
32	40	44	37	40
64	31	35	29	33

Table 6. Using wavelet for pattern existences queries

Compared to results that obtained by time series histograms (81/90 and 85/87), wavelets perform significantly worse in terms of accuracy. In fact, the wavelet transform transfers the time series data into time-frequency domain, therefore, it is difficult, using wavelets, to answer pattern existence queries, since frequency appearance order is encoded in the wavelet coefficients. Even using only the first few coefficients (8), which have very lower time resolution (scale), we could get high recall but low precision as shown in Table 6. The high recall was achieved by returning nearly all the data in the database as results and the lower precision is due to most of the important frequency bands are not used to answer queries. However, if we use more (32), both precision and recall drop. This is because the high frequency bands have higher time resolution, making them sensitive to time shifting.

For shape match queries, we compared the efficacy of wavelets and multi-scale histograms on CC data set. We did not use the ASL and Cameramouse data sets, because Euclidean distance requires the two compared wavelet coefficients to have the same length, thus the original time series must have the same length as well. Furthermore, since the wavelet transform requires the length of the sequences be a power of 2, we pad the sequences with 0 to make the length 64. We carried out the same classification test using different number of wavelet coefficients and the results are shown in Table 7.

	8	16	32	64
error rate	0.43	0.39	0.32	0.21

Table 7. Error rates using wavelet coefficients

Again the results are worse than the results achieved by using multi-scale histograms (0.06). The lower scale wavelets can not capture the information about higher frequency band, thus, the error rate with lower scale is higher than that of higher scale. The higher scale wavelets contain too much detail about the appearance time of frequency bands, which causes the wavelets to be sensitive to time shifting. In fact, with Euclidean distance, using all wavelet coefficients is the same as using raw representation of time series [12]. Thus, based on these two experiments and analysis, we conclude that wavelet representation is not robust to time shifting, making it unsuitable to answer pattern existence queries and introducing difficulties in answering shape match queries when data contain time shifting.

6 Related Work

The earliest proposal for similarity-based time series data retrieval was by Agrawal et. al. [1], which used Euclidean distance to measure the similarity between time series data and applied Discrete Fourier Transform (DFT) to reduce the dimensionality for fast retrieval. Later, this work was extended to design new similarity measures, other than Euclidean distance, for measuring the similarity between two time series data, such as Dynamic Time Warping (DTW) [3], Longest Common Subsequences (LCSS) [22], Edit distance with Real Penalty (ERP) [5] etc. All these techniques can be put into the category of shape match retrieval. A few approaches [2, 19, 18] have been proposed for finding movement patterns in time series data. The moving direction (the slope between two values) of a user specified interval [18], consecutive values [2], or a segment (obtained by a segmentation algorithm) [19] was represented as a distinct alphabet. Thus, these approaches converted the time series data into strings and could apply string-matching techniques to find the patterns. Lin et al. [14] have proposed a symbolic representation for time series data, where data points at neighborhood subspaces are treated as same and the distances between them are assigned value 0. However, this method overestimates the neighborhood similarity.

Compared to all the previous work on similarity-based time series retrieval, the multi-scale time histogram has the following advantages:

- Multi-scale time series histograms can be used to answer shape match and pattern existence queries.
- The cumulative histogram distance reduces the boundary effects introduced by value space quantization and overcomes the shortcomings of overestimating (such as L_1-norm and L_2-norm) or underestimating (such as weighted Euclidean distance) the distance.
- Multi-scale time histograms are invariant to time shifting and scaling, amplitude shifting and scaling. Moreover, they can reduce the noise disturbance.
- Multi-scale time histograms offer users flexibility to query the time series data on different accuracy level. Furthermore, lower scale histograms can be used to prune the false alarms from the database before we querying at higher scale histograms.

Recently, in spatial database domain, multi-scale histograms have been proposed to summarize rectangle objects for window queries [15]. In these approaches, the multi-scales refer to resolutions on value space; however, the multi-scales in our work refer to resolutions on time space.

7 Conclusions and Future Work

In this paper, we propose a novel representation of time series data using multi-scale time series histograms. This

representation is based on the intuition that the distribution of time series data can be an important cue for comparing similarity between time series. Multi-scale time series histograms are capable of answering both pattern existence queries and shape match queries. Moreover, they are invariant to time shifting and scaling, and amplitude shifting and scaling. A robust similarity measure, cumulative histogram distance, is used to measure the similarity between two time series histograms. Our experiments indicate that multi-scale time series histograms outperform string representations in finding patterns and outperform DTW and LCSS in answering shape match queries when the time series data contain noise or time shifting and scaling. The experiments also show that equal area bin histogram is more suitable for time series data comparison and distances of averages of histograms can effectively prune the false alarms from the database before computing the distance between two full histograms.

In future work, we will investigate the possibility of automatically setting up the scale value for users. We also plan to extend multi-scale histograms to subsequence matching.

References

[1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *Proc. 4th Int. Conf. of Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[2] R. Agrawal, G. Psaila, E. L. Wimmers, and M. Zaït. Querying shapes of histories. In *Proc. 21th Int. Conf. on Very Large Data Bases*, pages 502–514, 1995.

[3] D. J. Berndt and J. Clifford. Finding patterns in time series: A dynamic programming approach. In *Advances in Knowledge Discovery and Data Mining*, pages 229–248, 1996.

[4] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. pages 357–368, 1997.

[5] L. Chen and R. Ng. On the marriage of edit distance and Lp norms. In *Proc. 30th Int. Conf. on Very Large Data Bases*, pages 792–803, 2004.

[6] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 419–429, 1994.

[7] J. Gips, M. Betke, and P. Fleming. The camera mouse: Preliminary invertigation of automated visaul tracking for computer access. In *In Proc. Conf. on Rehabilitation Engineering and Assistive Technology Society of North America*, pages 98–100, 2000.

[8] D. Goldin and P. Kanellakis. On similarity queries for time series data: Constraint specification and implementation. In *Proc. of the Int. Conf. on Principles and Pratice of Constraint Programming*, pages 23–35, 1995.

[9] J. Hafner, H. Sawhney, W. Equitz, M. Flickner, and W. Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(7):729–739, 1995.

[10] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 151–162, 2001.

[11] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. In *Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 102–111, 2002.

[12] K.P.Chan and A.-C. Fu. Efficient time series matching by wavelets. In *Proc. 15th Int. Conf. on Data Engineering*, pages 126–133, 1999.

[13] K. Leung and R. T. Ng. Multistage similarity matching for sub-image queries of arbitrary size. In *Proc. of 4th Working Conf. on Visual Database Systems*, pages 243–264, 1998.

[14] J. Lin, E. Keogh, S. Lonardi, and B. Liu. A symbolic representation of time series, with implications for streaming algorithms. In *Proc. 8th Int. Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 2–11, 2003.

[15] X. Lin, Q. Liu, Y. Yuan, and X. Zhou. Multiscale histograms: Summarizing topological relations in large spatial datasets. In *Proc. 29th Int. Conf. on Very Large Data Bases*, pages 814–825, 2003.

[16] A. Natsev, R. Rastogi, and K. Shim. Walrus: a similarity retrieval algorithm for image databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 395–406, 1999.

[17] S. Park and W. Chu. Similarity-based subsequence search in image sequence databases. *Int'l. J. of Image and Graphics*, 3(1):31–53, 2003.

[18] Y. Qu, C. Wang, , L. Gao, and X. S. Wang. Supporting movement pattern queries in user-specified scales. *IEEE Trans. Knowledge and Data Eng.*, 15(1):26–42, 2003.

[19] H. Shatkay and S. Zdonik. Approximate queries and representations for large data sequences. In *Proc. 12th Int. Conf. on Data Engineering*, pages 536–545, 1996.

[20] M. Stricker and M. Orengo. Similarity of color images. In *Proc. 7th Int. Symp. on Storage and Retrieval for Image and Video Databases*, pages 381–392, 1995.

[21] M. J. Swain and D. H. Ballard. Color indexing. *Int. Journal of Computer Vision*, 7(1):11–32, 1991.

[22] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proc. 18th Int. Conf. on Data Engineering*, pages 673 – 684, 2002.

[23] M. Vlachos, J. Lin, E. Keogh, and D. Gunopulos. A wavelet-based anytime algorithm for k-means clustering of time series. In *Proc. Workshop on Clustering High Dimensionality Data and Its Applications*, pages 23–30, 2003.

[24] C. Wang and X. Wang. Supporting content-based searches on time series via approximation. In *Proc. 12th Int. Conf. on Scientific and Statistical Database Management*, pages 69–81, 2000.

[25] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Int. Conf. on Very Large Data Bases*, pages 194–205, 1998.

[26] H. Wu, B. Salzberg, and D. Zhang. Online event-driven subsequence matching over financial data streams. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 23–34, 2004.

[27] B.-K. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proc. 14th Int. Conf. on Data Engineering*, pages 23–27, 1998.

[28] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 181–192, 2003.

Searching for Related Objects in Relational Databases

Xiaoxin Yin
UIUC
xyin1@uiuc.edu

Jiawei Han
UIUC
hanj@cs.uiuc.edu

Jiong Yang
Case Western Reserve University
jiong@eecs.cwru.edu

Abstract

To discover knowledge or retrieve information from a relational database, a user often needs to find objects related to certain source objects. There are two main challenges in building an effective object search system: the huge amount of objects in the database and the large number of different relationships between objects. In this paper we introduce Ross*, an efficient and accurate relational object search system.* Ross *accepts complex queries that enable users to specify the relationships among objects. To measure the relationships of join paths,* Ross *considers the different semantics of different joins, and combines both selectivity and lengths of join paths to measure their strengths. A novel approach is used to find the best join paths between relations, which converts the database schema into a graph, so that the shortest paths in the graph correspond to best join paths in the database.* Ross *uses a stream-based system architecture to handle complex queries containing logical operators, which can find the most related objects upon users' requests. Comprehensive experiments are conducted to show the high scalability and effectiveness of* Ross.

1. Introduction

Most of the structured data in the world is stored in relational databases. To retrieve information and discover knowledge from a database, a user often needs to find objects related to certain source objects. For example, a new graduate student may want to know professors related to a certain research area; an e-business company may want to find customers related to each brand of products; and the DC police may want to find linkages between white chevy vehicles, rifles, and people to identify snipers. Although these tasks may be accomplished by submitting many SQL queries to find objects related to different source objects, the end users seldom have sufficient knowledge about database systems and familiarity with certain databases. It is also often infeasible to manually analyze all possible relationships among entities in a database. It is highly desired to automatically find objects that are strongly related to certain source objects, or find important linkages between a relation and certain source objects.

There are two main challenges in object search in relational databases: scalability and accuracy. A database usually contains many relations and a huge number of objects. It is time-consuming to analyze different relationships between different relations and objects. On the other hand, there are usually many different join paths between two relations. To search for related objects, the system must be able to measure the strengths of relationships represented by different join paths, which is a challenging task when no knowledge about the semantics of such relationships are available.

In recent years much attention has been paid to keyword query in relational databases [3, 1, 9, 7]. These systems model the objects in a database as a graph. Given a query containing a set of keywords, they return the minimum spanning trees of objects that contain all query words. Unlike a keyword query system, an object search system searches for objects in a target relation that are linked to a set of source objects through strong linkages in databases, and should have the following features. First, it should allow the user to specify the relationship among the source objects. For example, a query could be "finding students related to Prof. Smith and either database or data mining". In contrast, a keyword query system does not even allow the user to specify the relationship among keywords. For example, the minimum spanning tree for query "finding students related to Prof. Smith and data mining" could be "student - Prof.Smith - data mining", and the system will return students related to Prof. Smith, which are not what the user wants.

The second feature of an object search system is that, it should have a good measure for the relationships between objects. Most keyword query systems use lengths of join paths between objects as their distances [1, 9, 7]. In reality different joins have very different semantic meanings. Thus different join paths, even with identical lengths, may represent linkages of very different strengths. Please refer to the schema in Figure 1. The join path $Student \bowtie Register \bowtie TeachCourse \bowtie Register \bowtie Student$ (two students taking same course) represents a much weaker link than $Student \bowtie Advise \bowtie Professor \bowtie Advise \bowtie Student$ (two students supervised by same advisor). Another example is that,

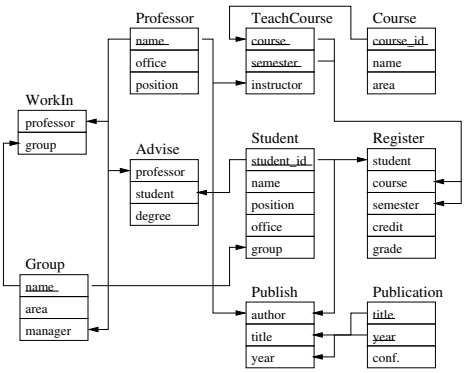


Figure 1: Schema of the CS Dept database

$Professor \bowtie TeachCourse \bowtie Register \bowtie Student$ (professor teaches student) represents a weaker link than $Professor \bowtie Publish \bowtie Publication \bowtie Publish \bowtie Student$ (professor coauthors paper with student). In our system we adopt a new measure of relationships that considers more information about join paths.

In this paper we propose Ross (Relational Object Search System), an object search system that handles complex queries and uses a new measure for strengths of relationships among objects. Ross accepts keyword queries that contain a target relation and a logical expression (containing "AND/OR") of source objects. An example query is "FIND Student RELATEDTO ('John Doe' OR 'Mike Smith') AND Course 'CS400"'. The query may also specify one or more relations in the middle of the join path between the target relation and a source object. For example, "FIND Student RELATEDTO 'John Doe' VIA Publication" will find students coauthoring papers with John Doe. Compared to queries containing sets of keywords, queries in Ross have more expressive power. Ross uses a stream-based system architecture for processing queries, which builds an execution tree when executing a query. Each node in the tree provides a stream of result objects and the root node provides the final results to the user.

To search for related objects, a system must be able to identify strong linkages among tuples. As shown above, the number of joins is not a good measure for the strengths of join paths, and the selectivity of a join path often indicates the strength of the relationship. A join path p having small selectivity means that an object in the first relation of p are joinable with a small number of objects in the last relation of p, which usually indicates strong relationship between those objects, and vice versa. The selectivity of strong links, such as professor advising students, is usually smaller than that of weak links, such as professor teaching students. Besides selectivity, the number of joins in a join path also affects its strength. Ross combines the selectivity and lengths to measure the strengths of relationships of join paths. It is shown by experiments that this measure is more effective than the lengths of join paths.

Because Ross uses selectivity as an important measure for strengths of relationships, it needs to make reasonably accurate estimation for selectivity. Selectivity estimation has been studied for many years [11, 10, 14, 13, 6]. However, Ross faces a very different challenge. No join path is provided in the query, and it needs to find the best path based on selectivity and length, and return the corresponding objects. Moreover, in many cases the best path does not match user's mind and more results are needed. Therefore, Ross must be able to provide the top-k best join paths between any two relations, which cannot be solved by traditional selectivity estimation approaches.

Ross uses a novel approach for finding best (or most selective) join paths between relations. It pre-computes the selectivity of many short join paths, which can be combined to make accurate estimations on selectivity of longer paths. The relational schema is converted into a graph, in which relations are modelled as nodes and joins as edges. By integrating the pre-computed information into the graph, the k shortest paths in the graph correspond to the k best join paths in the database, which can be found efficiently as in [12]. In experiments it is shown that Ross achieves high efficiency and scalability, and can accurately predict for strong relationships between different entities.

The rest of this paper is organized as follows. The problem definition is described in Section 2. We present the approach for selectivity estimation in Section 3. Section 4 describes the approach for finding best join paths. Section 5 describes the system architecture and query processing. Section 6 shows the experimental results. Related work is introduced in Section 7 and the study is concluded in Section 8.

2. Problem Definitions

2.1. Query Format

Given a target relation R_t and a set of source objects, Ross searches for objects in R_t that are most related to the source objects. A query contains the following parts: (1) the target relation R_t, (2) a set of source objects, and (3) one or more relations in the middle of join paths between the source object and R_t (optional). A source object is a tuple that represents an object. To specify a source object, the user should provide some keywords for that object, and the name of the relation (optional). An example query is Q = `FIND Student RELATEDTO (('John Doe') AND (Course 'Data mining' VIA Registration))` . In this query `Student` is the target relation, `Course` is the source relation, and `Registration` is the middle relation. In the database, each relation is manually labelled with a set of keywords, so that the user does not need to know the exact name for the relation. Different source objects form a logical expression containing {`AND, OR`}.

A query is a *simple query* if it does not have logical operators; otherwise it is a *complex query*.

The answer to a query is a list of tuples in R_t, which are ranked according to their relationship with the source objects. A complex query can be considered as the combination of two or more sub-queries. For example, the above query Q contains two sub-queries: Q_1 = `FIND Student RELATEDTO ('John Doe')` and Q_2 = `FIND Student RELATEDTO (Course 'Data mining' VIA Registration)`. For a simple query Q_1, each tuple t in the answer of Q_1 has a score $s(t, Q_1)$, which is the weight of the best path from which t is retrieved. A smaller score indicates a stronger relationship, and $s(t, Q_1) = +\infty$ if t does not appear in Q_1's answer.

For a complex query, the scores of answer tuples can be determined by scores of its sub-queries. Suppose $Q = Q_1$ AND Q_2. To determine $s(t, Q)$ based on $s(t, Q_1)$ and $s(t, Q_2)$, we consider the following scenario. Suppose query Q is associated with s_{max}, a threshold of score which is given by user. The answer to Q_i $(i = 1, 2)$ is all tuples t with $s(t, Q_i) \leq s_{max}$. Then only those tuples t with $s(t, Q_1) \leq s_{max}$ and $s(t, Q_2) \leq s_{max}$ are in the answer of Q. Therefore, $s(t, Q)$ should be defined as $\max(s(t, Q_1), s(t, Q_2))$. Similarly, if $Q = Q_1$ OR Q_2, then $s(t, Q) = \min(s(t, Q_1), s(t, Q_2))$. Other operators (such as average) can be defined similarly, which can be the future work.

2.2. Measure of Relationship

For join path p in a database, there are two features that are highly related to the semantic strength of relationship represented by p. They are the length and selectivity of p. Ross combines both of them to measure the strengths of relationships of join paths, in order to find objects most related to the source objects.

Consider a join path $p = R_1 \bowtie \cdots \bowtie R_l$. The *length* of p, denoted as $|p|$, is defined as the number of joins in p. In general, the *selectivity of a join path* p is the average number of tuples in R_l that is joinable with a tuple in R_1, defined as follows.

Definition 1 (Selectivity). *For a join path* $p = R_1 \bowtie \cdots \bowtie R_l$*, its* selectivity $S(p)$ *is the average number of tuples in* R_l *that are joinable to a tuple* t *in* R_1 *via* p*.*

$S(p)$ is the average selectivity for all tuples in R_1. The numbers of tuples in R_l that are joinable to different tuples in R_1 might be quite different. For example, a senior professor may have taught 500 students but a junior one may have taught only 50. The selectivity of an individual tuple does not indicate the semantic strength of the relationship between tuples. Therefore, the average selectivity is used for this purpose.

The selectivity and semantic strength of a join path are highly correlated. Consider two join paths p_1 and p_2 between relations R_1 and R_2. If $S(p_1)$ is much lower than $S(p_2)$, then on average an object o_1 in R_1 joins with much fewer objects in R_2 via p_1, which usually indicates that o_1 has stronger relationship with those objects. For example, $p_1 = Student \bowtie Advise \bowtie Professor \bowtie Advise \bowtie Student$ (a student shares advisor with another one), $p_2 = Student \bowtie Register \bowtie TeachCourse \bowtie Register \bowtie Student$ (a student takes same course with another one). The relationship represented by p_1 is usually much stronger, because a professor usually advises a small number of students (compared with course registration) and thus they know each other very well. Selectivity may be misleading in some cases, especially for long join paths. Ross uses the weighted average of logarithm of selectivity and lengths of join paths to estimate their semantic strengths. The detailed approach are described in Section 4.

3. Selectivity Estimation

A simple approach to estimating selectivity of a join path $p = R_1 \bowtie \cdots \bowtie R_l$ is to use the product of selectivity of every join in the path, *i.e.*, $S(p) = \prod_{i=1}^{l-1} S(R_i \bowtie R_{i+1})$. This approach is accurate only if the joins in p are independent. In real life joins are often related to each other. For example, in join path $Professor \bowtie TeachCourse \bowtie Course$, although each professor may teach ten courses in different semesters, the average number of different courses he teaches is probably only two or three, because the courses he teaches in different semesters are often overlapped.

Overlapping is the phenomenon that in a join path $R_1 \bowtie \cdots \bowtie R_l$, different intermediate tuples on a certain relation R_i join with same tuple in another relation R_j $(j > i)$. Overlapping happens in many join paths. To estimate selectivity of a join path, it is inevitable to estimate its degree of overlapping.

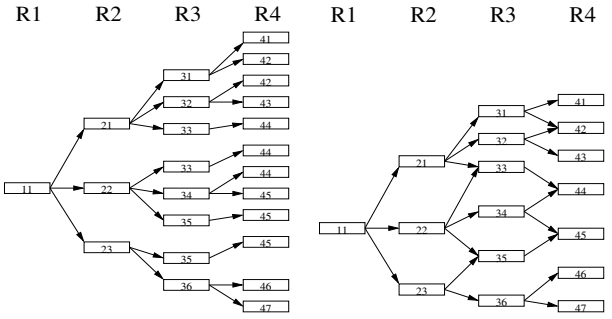
Selectivity of join paths can be estimated by many previous approaches [6, 11]. However, such approaches can only estimate selectivity of certain join paths, but cannot help Ross to find most selective join paths from a database. We use a different approach to estimate selectivity, mainly by pre-computing the degree of overlapping for many short join paths, and using them to estimate the overlapping of longer paths. It is shown by experiments that this approach achieves high accuracy and enables efficient search for selective join paths.

In the entity-relationship model of a database, different joins have different semantics. We consider the following three types of joins in relational databases.

1. **k-f join:** A k-f join is a key to foreign-key join. It does not cause overlapping because different tuples cannot join with the same tuple via a key to foreign-key join.
2. **f-k join:** An f-k join is a foreign-key to key join. Its selectivity is one and it may cause overlapping.
3. **indirect join:** There is an indirect join $R_1 \bowtie^{R_2} R_3$, if R_2 is a relation of relationship which connects R_1 and R_3. An indirect join is a many-to-

many join and may cause overlapping. It actually contains a k-f join and an f-k join.

We pre-compute the selectivity of every pair of consecutive joins. If there are joins $R_1 \bowtie R_2$ and $R_2 \bowtie R_3$, then the selectivity $S(R_1 \bowtie R_2 \bowtie R_3)$ is pre-computed. To estimate the selectivity of a join path with more than two joins, we need to combine the selectivity of each pair of joins in the path. Suppose the selectivity of $p = R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$ is to be estimated. For simplicity, we assume each join in this path is a many-to-many join, which is the most general case. To estimate p's selectivity $S(p)$, we need to combine $S(R_1 \bowtie R_2 \bowtie R_3)$ and $S(R_2 \bowtie R_3 \bowtie R_4)$. Figure 2 shows an example about the relationship between tuples, without and with overlapping considered.



a) Without overlapping b) With overlapping

Figure 2: The effect of overlapping on selectivity

We first define the *overlapping factor* of a join path of length three. The overlapping factor of path $R_1 \bowtie R_2 \bowtie R_3$ is defined as

$$O(R_1 \bowtie R_2 \bowtie R_3) = \frac{S(R_1 \bowtie R_2) \cdot S(R_2 \bowtie R_3)}{S(R_1 \bowtie R_2 \bowtie R_3)} \quad (1)$$

In Figure 2, $O(R_1 \bowtie R_2 \bowtie R_3)$ is the average number of times a tuple appears in R_3 in Figure 2(a).

To estimate $S(p)$, we need to know how many times each tuple appears in R_4 in Figure 2(a). First, for every tuple t in R_2, if all identical tuples in R_4 that are joinable to t are merged together, the number of tuples in R_4 will shrink to $\frac{1}{O(R_2 \bowtie R_3 \bowtie R_4)}$ of its original number. Then, if all identical tuples in R_3 are merged together, then the number of tuples in R_3 will shrink to $\frac{1}{O(R_1 \bowtie R_2 \bowtie R_3)}$ of its original number, and the number of tuples in R_4 will shrink accordingly. There might still be repeated tuples on R_4, such as tuple 45. However, only those tuples in R_4 that are joined with tuple 11 via different tuples in both R_2 and R_3 may be repeated. This kind of overlapping is usually insignificant, and we can estimate $S(p)$ as

$$\begin{aligned} \hat{S}(p) &= \frac{S(R_1 \bowtie R_2) \cdot S(R_2 \bowtie R_3) \cdot S(R_3 \bowtie R_4)}{O(R_1 \bowtie R_2 \bowtie R_3) \cdot O(R_2 \bowtie R_3 \bowtie R_4)} \\ &= S(R_1 \bowtie R_2 \bowtie R_3) \cdot \frac{S(R_2 \bowtie R_3 \bowtie R_4)}{S(R_2 \bowtie R_3)} \end{aligned} \quad (2)$$

In general, for a join path $p = R_1 \bowtie \cdots \bowtie R_l$, the overlapping factor on p is defined as the overlapping on p that is not due to the overlapping on any subpath of p. It is the average number of ways a tuple t_1 in R_1 joins to a tuple t_l in R_l through different tuples on every relation from R_2 to R_{l-1}.

Given $p = R_1 \bowtie \cdots \bowtie R_l$, suppose only the selectivity of every subpath with length no greater than k has been pre-computed. Then the overlapping factors on subpaths with length no greater than k are known to us. For a subpath p' with length greater than k, $O(p')$ cannot be directly based on given information. Fortunately, the overlapping of long paths are usually quite weak. The overlapping is usually caused by the semantic bonds between different intermediate tuples in a relation R_i $(1 < i < l)$.[1] If two tuples t_{i1} and t_{i2} in R_i $(i \geq 4)$ are joined from the tuple t_1 in R_1 via different tuples in every relation R_j $(1 < j < i)$, then the semantic bonds between t_{i1} and t_{i2} are usually pretty weak. Consequently, the overlapping factors for long paths are usually much smaller than those of short paths.

In general, if we have pre-computed the selectivity of every join path of length no greater than k, we estimate the selectivity of path p as

$$\hat{S}(p) = S(R_1 \bowtie \cdots \bowtie R_k) \prod_{i=2}^{l-k+1} \frac{S(R_i \bowtie \cdots \bowtie R_{i+k-1})}{S(R_i \bowtie \cdots \bowtie R_{i+k-2})}$$

It can be seen that there is a trade-off between the pre-computation cost and accuracy of selectivity estimation. Suppose we pre-compute the selectivity of every k consecutive joins in the database. As k gets larger, selectivity estimation becomes more accurate, but more cost are paid in pre-computation and the graph of schema becomes more complex (as shown in Section 4). In Section 6 we show experiments on the accuracy of selectivity estimation with different values of k. Finally, we choose $k = 2$ because it has reasonably good accuracy and is inexpensive in time and space.

4. Finding Best Join Paths

Given a user query containing target relation R_t, and a source object o in relation R_s. To answer this query, ROSS needs to find the best join path p_1 from R_s to R_t and return tuples retrieved from that path. If p_1 is not the path in user's mind, or more results are needed, ROSS needs to find the next best path p_2, and so on. Therefore, an approach is needed to find top-k best paths between two relations. Based on our method of selectivity estimation, we propose a novel approach that converts a database schema into a graph, so that the k shortest paths in the graph correspond to the

1 Overlapping may also be caused by the limited number of tuples in a certain relation R_i (different tuples in R_{i-1} that are not related can join with the same tuple in R_i by chance). This kind of overlapping should not be considered because it has nothing to do with the semantic strengths of join paths.

best join paths in the database. In this way ROSS can find good join paths and answer queries efficiently.

4.1. Graph Construction

In ROSS the strength of a join path is measured by its selectivity and length. As mentioned before, we precompute the selectivity of every pair of joins (the overlapping factor for every join path of length two). The selectivity of path $p = R_1 \bowtie \cdots \bowtie R_l$ is estimated by

$$\hat{S}(p) = \frac{\prod_{i=1}^{l-1} S(R_i \bowtie R_{i+1})}{\prod_{i=2}^{l-1} O(R_{i-1} \bowtie R_i \bowtie R_{i+1})} \quad (3)$$

The logarithm of selectivity is used as an indication for the distances between objects. If the selectivity of a join path is less than 1, it is set to 1 because a path cannot indicate negative distance between objects. The weight of a path p is defined as the weighted average of the logarithm of p's estimated selectivity and p's length $|p|$.

Definition 2 (Weight of a path). *The weight of a join path p is defined as*

$$W(p) = \log_2 \hat{S}(p) + \beta \cdot |p| \quad (4)$$

β is a parameter adjustable by users. A path with smaller weight is considered to have greater strength. To answer keyword queries, ROSS needs to be able to find the paths with the smallest weights. If $\beta = 0$, then ROSS will find most selective join paths; if β is very large, ROSS will find shortest join paths.

Consider a simple database whose schema is shown in Figure 3. Arrows go from keys to corresponding foreign-keys, and double directed edges indicate indirect joins. Suppose one wants to find the paths with

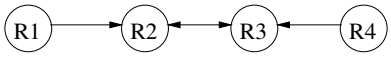


Figure 3: A simple database schema

the smallest weights from R_1 to R_4. The simplest way to convert the schema into a graph is to add a node v_i for each relation R_i, and two edges (v_i, v_j) and (v_j, v_i) for each join $R_i \bowtie R_j$. The weight of edge (v_i, v_j) is set to $\log_2 S(R_i \bowtie R_j) + \beta$. However, no overlapping information is integrated into this graph.

To utilize the pre-computed selectivity of short join paths, we may add new edges to the graph. The modified graph is shown in Figure 4, in which four edges are added for the four subpaths of length two. The weights of added edges are set according to the selectivity of the subpaths. In this graph, the shortest path from v_1

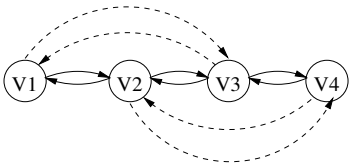


Figure 4: The graph with edges for subpaths

to v_4 may be either $v_1 \to v_2 \to v_4$ or $v_1 \to v_3 \to v_4$. However, only the overlapping on one subpath is considered, which is not correct according to Eq. (3).

To utilize the overlapping factors on all subpaths, the graph is constructed in the following way.

1. For each relation R_i, add node v_i to the graph.
2. For each join $R_i \bowtie R_j$, two edges (v_i, v_j) and (v_j, v_i) are added to the graph. The weight of (v_i, v_j) is set as $W(v_i, v_j) = \log_2 S(R_i \bowtie R_j) + \beta$. $W(v_j, v_i)$ is set in the same way.
3. For each subpath $p' = R_{i1} \bowtie R_{i2} \bowtie R_{i3}$ whose selectivity is estimated, if $O(p') > 1$, then add v'_{i2}, an extra node of v_{i2}, to the graph. (If v'_{i2} already exists, add v''_{i2} instead.) Add edges (v_{i1}, v'_{i2}) and (v'_{i2}, v_{i3}) to the graph. (It is explained below how to set the weights of these edges.)
4. For each extra node v'_{i2} which is created for subpath $p' = R_{i1} \bowtie R_{i2} \bowtie R_{i3}$, if there is an extra node v'_{i3} that is created for subpath $R_{i2} \bowtie R_{i3} \bowtie R_x$, then add an edge from v'_{i2} to v'_{i3}.

An example graph is shown in Figure 5. For simplicity, only edges added from left to right are shown, because other edges are not helpful in finding the most selective paths from R_1 to R_4. The weights of the added edges are set according to Theorem 1.

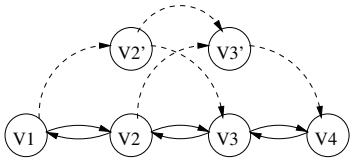


Figure 5: The augmented graph

Theorem 1 *For a path $p = R_1 \bowtie \cdots \bowtie R_l$,*

$$W(p) = \sum_{i=1}^{l-1} W(R_i \bowtie R_{i+1}) - \sum_{i=2}^{l-1} \log_2 O(R_{i-1} \bowtie R_i \bowtie R_{i+1})$$

■

In details, the weights of edges are set as follows.

1. For each extra node v'_{i2} that is added for subpath $p' = R_{i1} \bowtie R_{i2} \bowtie R_{i3}$, $W(v_{i1}, v'_{i2}) = W(v_{i1}, v_{i2})$, and $W(v'_{i2}, v_{i3}) = W(v_{i2}, v_{i3}) \quad \log_2 [O(p')]$.
2. If edge (v'_{i2}, v_{i3}) exists (v'_{i2} is an extra node for v_{i2}), then for each edge (v'_{i2}, v'_{i3}) (v'_{i3} is an extra node of v_{i3}), set $W(v'_{i2}, v'_{i3}) = W(v'_{i2}, v_{i3})$.

In the graph in Figure 5, the edge weights are set as follows. $W(v_1, v'_2) = W(v_1, v_2)$, $W(v'_2, v_3) = W(v_2, v_3) - \log_2(O(R_1 \bowtie R_2 \bowtie R_3))$, $W(v_2, v'_3) = W(v_2, v_3)$, $W(v'_3, v_4) = W(v_3, v_4) - \log_2(O(R_2 \bowtie R_3 \bowtie R_4))$, $W(v'_2, v'_3) = W(v'_2, v_3)$. Assume $O(R_1 \bowtie R_2 \bowtie R_3) > 1$ and $O(R_2 \bowtie R_3 \bowtie R_4) > 1$. Then the shortest path from v_1 to v_4 is $v_1 \to v'_2 \to v'_3 \to R_4$, whose weight is exactly $W(R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4)$. There might be edges with negative weights in the graph. Later in corollary 2 it is shown that the algorithm will not be affected by negative edges.

Definition 3 (Path correspondence). *Suppose a graph G is constructed for database D. A graph path $q = v_{k_1} \rightarrow \cdots \rightarrow v_{k_l}$ corresponds to a join path $p = R_1 \bowtie \cdots \bowtie R_l$, if and only if v_{k_1} is the node for R_1, v_{k_l} is the node for R_l, and v_{k_i} is the node or extra node for R_i $(1 < i < l)$.*

Theorem 2 *Suppose a graph G is constructed based on database D. If there is a join path p in D with $W(p) = w$, then there is path q in G with weight $W(q) = w$, and q corresponds to p. Also, q is the shortest path in G that corresponds to p.* ∎

By theorem 2 it is known that, for each join path p in the database, there are a set of paths in the graph that correspond to p. Among this set of paths, the shortest path q has the same weight with p.

Corollary 1 *Suppose a graph G is constructed based on database D. If there is a path q in G, then there is a join path p so that q corresponds to p and $W(p) \leq W(q)$.* ∎

We say two paths q_1 and q_2 in graph G are *equivalent* if they correspond to same join path in the database. q_1 and q_2 are *distinct* if they are not equivalent. Based on theorem 2 and corollary 1, the following theorem can be proved by induction, which reduces the problem of finding best join paths in a database into finding shortest paths in a graph.

Theorem 3 *Suppose graph G is built based on database D. The top-k distinct shortest paths in G correspond to the top-k join paths in D with smallest weights.* ∎

Here we analyze the number of extra nodes and edges added in the graph construction process. Suppose on average each relation R_i has f_1 k-f joins, f_2 f-k joins, and f_3 indirect joins. Then the average number of subpaths $R_j \bowtie R_i \bowtie R_k$ whose selectivity are estimated is about $(f_1+f_3)(f_2+f_3)$. Let $g = (f_1+f_3)(f_2+f_3)$. The average number of extra nodes added for v_i is about g. For each extra node v_i' for v_i, if v_i' is added for subpath $R_j \bowtie R_i \bowtie R_k$, then two edges (v_j, v_i') and (v_i', v_k) are added, and one edge is added from v_i' to any extra node of v_k created for subpath $R_i \bowtie R_k \bowtie R_x$. Therefore, the number of edges added for all extra nodes of v_i is about $g(f_2 + f_3)$. If there are n relations in the database, then there will be about gn extra nodes and $g(f_2 + f_3)n$ extra edges added. Usually f_1 and f_2 are less than 2, and f_3 is less than 1. Then g is less than 9, which keeps the new graph in small size.

By the above approach the database schema can be converted into a graph, which integrates the precomputed selectivity of every join path of length two. If the selectivity of longer join paths are pre-computed, the same approach can still be used, by adding more extra nodes and edges. It can also be proved that the negative edges will not affect such algorithms.

Corollary 2 *Suppose a graph G is constructed based on database D. Any graph path q that starts at node v_i has positive weight. If a graph path q starts at node v_i, there cannot be a negative circle on q.* ∎

4.2. Find Shortest Paths in Graphs

The problem of finding shortest paths in graphs has been studied for decades [16, 12]. Ross chooses a recent and efficient algorithm [12] that finds the k shortest paths in $\mathcal{O}(k \cdot n + k \cdot m)$ time and $\mathcal{O}(k \cdot n + m)$ space, for a graph containing n nodes and m edges. Suppose the current best path $p = (s, n_1, \ldots, n_{r-1}, t)$. The algorithm is based on the observation that the best alternative path is either based on the shortest path from source s to node x for any x pointing to destination t, or based on the best alternative path to $(s, n_1, \ldots, n_{r-1})$.

When searching for shortest graph paths, different paths in G might correspond to same join path in D. Thus to find the k best join paths, usually more than k graph paths need to be found. Let k' be the number of graph paths found. In our experiments it is shown that k' is usually only several times larger than k, even for databases with complex schemas. In general, Ross uses an effective and efficient approach to find the most selective join paths in databases, which enables it to answer keyword queries accurately.

5. System Architecture

The overall procedure of Ross is shown in Figure 6. The user submits a query to, which is converted to an *object query* by the inverted index. Then the object query is executed by the execution engine and the result objects are returned to the user.

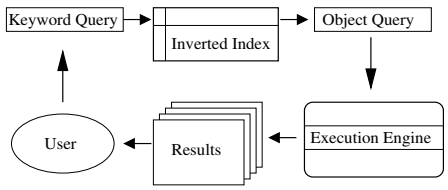


Figure 6: The query processing procedure

5.1. Backend Database

A backend database for Ross is a relational database. An inverted index is created for the values in the database. Based on the index, Ross can find the relations and attributes in which each source object appears. For example, {'John','Doe'} may be found in *Professor.name, Advise.professor, TeachCourse.instructor, WorkIn.professor*, and *Publish.author*.

Ross builds an object query for a user query. It finds out the target relation, and each source object. Then it builds a tree structure according to the logical expression. The object query for `FIND Student RELATEDTO (('John Doe') AND (Course 'Data mining' VIA Registration))` is shown in Figure 7.

One keyword (or set of keywords) in the query may correspond to multiple source objects in the database.

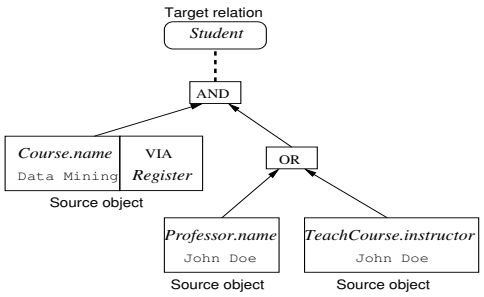


Figure 7: An example object query

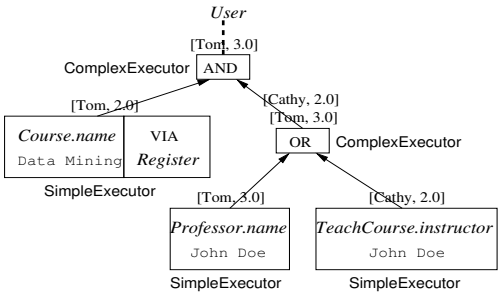


Figure 8: An example of query processing

Those source objects are connected by 'OR' operator in the object query.

5.2. Query Execution

For each node in the tree of query, ROSS creates an *Executor*, which outputs a stream of result tuples for the corresponding tree branch. A *SimpleExecutor* is created for each leaf node (source object), and a *ComplexExecutor* is created for each non-leaf node with a logical operator. The input to a SimpleExecutor is a target relation, a source object, and a set of middle relations. The input to a ComplexExecutor is the result tuples from multiple SimpleExecutors. Each executor provides only one function, *GetNextTuple()*, which returns the next best tuple in the answer of this executor. In this way it provides a stream of result tuples, ordered by their scores (low to high).

A SimpleExecutor first finds the shortest path between the target relation and source relation, and puts the tuples retrieved from this path into a queue. When asked for the next best tuple, it extracts a tuple from the queue. If the queue is empty, it finds the next shortest path and puts the tuples from that path into the queue. If one or more middle relations are specified for the source relation, then only paths containing all middle relations are considered. This does not increase the running time significantly, because of the high efficiency of the shortest-paths algorithm.

For a ComplexExecutor, when the function *GetNextTuple()* is called, it gets tuples from its child executors, and returns the next best tuple. A ComplexExecutor of 'OR' returns the next tuple with the smallest score from any of its children. A ComplexExecutor of 'AND' returns the next tuple whose maximum score on all its children is smallest. An example of query processing is shown in Figure 8. The tuples above each executor represents the tuples it returns.

Each tuple is mapped to a 64-bit integer via a hash function, so that the communication between different executors is very small. An executor may not return a tuple twice. So each executor maintains a buffer containing all tuples (integers) that have been returned.

The algorithm of *GetNextTuple()* for an ComplexExecutor of 'OR' is very simple. An executor E keeps the next best tuple from every child executor, which has not been returned by E, and selects the tuple with the smallest score to return.

The algorithm for an ComplexExecutor of 'AND' is more complex. An executor E needs to maintain the set of tuples retrieved from each child executor. When asked for the next best tuple, it looks at the next best tuple t from all children. Suppose t is from child executor E_1. If t is contained in the set of retrieved tuples of each child, then t is returned. Otherwise t is put into the set of tuples of E_1, and E looks at the next best tuple from any child, and so on. When a tuple t_j from E_j is considered as a candidate to return, all tuples with smaller scores have been put into T_i's $(1 \leq i \leq k)$. Thus any tuple t that appears in all T_i's and has smaller score than t_j has been returned. Therefore, t_j is the next best tuple that E should return.

6. Experimental Results

We perform comprehensive experiments to show the efficiency and effectiveness of our approach. Three databases are used in experiments. The first is CS Dept database whose schema is shown in Figure 1. It is collected from the website of Dept. of CS at UIUC. Relation *Registration* is randomly generated and all other relations contain real data. The second database is Northwind database from Microsoft, whose schema is shown in Figure 9. The third is TPC-H benchmark database [15], which can be generated in different sizes.

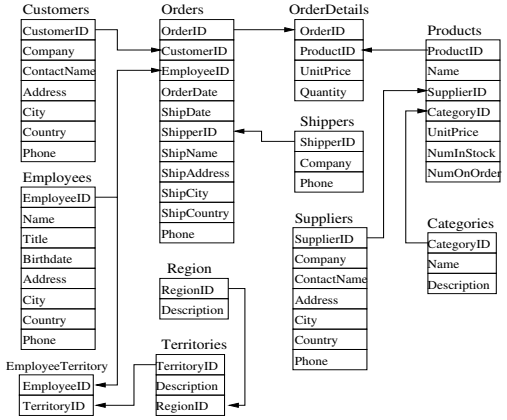


Figure 9: Northwind database

Experiments are performed on a Pentium 4 PC with

1GB RAM, running Windows 2000. Microsoft SQL server 7.0 is used as the backend database. The parameters are set as follows. The base weight of a join $\beta = 0.5$. The maximum weight of join paths is 10.0, and paths with greater weights are not considered.

6.1. Selectivity Estimation

As in Section 3, Ross pre-computes the selectivity of every join path of length k. When estimating the selectivity of a join path p, the selectivity of all pre-computed subpaths are combined. The following experiments show the accuracy of selectivity estimation for different values of k. 20 join paths are randomly generated for a database, each containing l joins. For each path p with real selectivity $S(p)$, we use different values of k ($k = 1, 2, 3$) to estimate p's selectivity $\hat{S}(p)$. The estimation error is defined as $err(p) = |\log_2(\hat{S}(p)/S(p))|$. The average error is shown in Figure 10, with $l = 2, 3, 4, 5$. It can be seen that the estimation accuracy of $k = 3$ is not much better than $k = 2$. Considering computational cost, we choose $k = 2$ in Ross, which has fairly high accuracy.

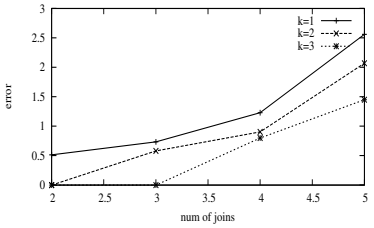


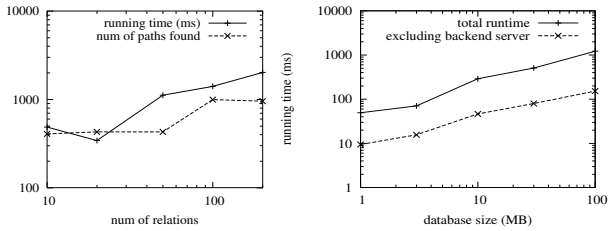
Figure 10: Selectivity estimation on CS Dept database

6.2. Scalability

In the following experiment database schemas are randomly generated to test Ross's scalability w.r.t. the size of schema. To generate a database schema with n relations, we first generate the n relations, then randomly add foreign-keys to each relation. The expected number of foreign-keys in each relation is 2. The expected selectivity of each k-f join is 4. The overlapping factor for each pair of joins is 2. All of them are randomly variables obeying exponential distribution.

We test Ross's scalability w.r.t. number of relations in databases. 10 database schemas are randomly generated for each n ($n \in \{10, 20, 50, 100, 200\}$). For each schema, 10 pairs of source and destination are randomly selected, and the 100 shortest distinct paths are found between each pair. The average running time and the total number of paths (not necessarily distinct) are shown in Figure 11 (a). One can see that Ross is highly scalable in finding the best join paths.

To test its scalability w.r.t. the number of tuples, we use TPC-H databases with size from 1MB to 100MB (raw data). We generate 20 keyword queries by randomly choosing the source and target relations, and find a meaningful path between them. Example queries include ‘‘FIND Supplier RELATEDTO (Nation 'CANADA')’’ and ‘‘FIND Customer RELATEDTO (Product 'MEDIUM ANODIZED NICKEL' VIA Order)’’.



a) w.r.t. num of relations b) w.r.t. database size

Figure 11: Scalability of Ross

Figure 11 (b) shows the average total running time and running time of Ross excluding the query processing time of backend database. One can see that Ross is highly scalable w.r.t. the database size.

6.3. Relationship Prediction

To test the performance of Ross on object search, we use it to predict for the strong relationships in the databases. Some join path in a database represents the only very strong relationship between two relations, such as professors advising students. We use this relationship as the target relationship. We remove the relationship from the database, and use Ross to predict it from other relationships. For example, we remove the *Advise* relation, and then search for students (or professors) most related to each professor (or student). Because students advised by a professor are the students most related to him, an effective object search system should be able to identify those students from information in other relations.

A target relationship must be the strongest relationship between two relations, and there must be other information involving the two relations that are related to the target relationship. We find the advising relationship is the only qualified target relationship in CS Dept database.

To compare prediction accuracy, we build another system Koss that is identical to Ross except using lengths of join paths to measure their goodness. Koss answers queries with shortest join paths, which is very similar to previous keyword query systems on relational databases (e.g. DISCOVER [9] and BANKS [3]).

We first remove the *Advise* relation. Then for each professor p, Ross and Koss are used to find the 100 students most related to p. Suppose p advises a set of students S. The raw score of a ranked list of students $L = s_1, \ldots, s_{100}$ is defined as

$$\bar{s}(L) = \sum_{s_i \in S, 1 \le i \le 100} \frac{1}{i}$$

The ratio between $\bar{s}(L)$ and the optimal score $\sum_{i=1}^{|S|} \frac{1}{i}$ is used as the score of L, which means $s(L) = \frac{\bar{s}(L)}{\sum_{i=1}^{|S|} \frac{1}{i}}$.

The accuracy of predicting students advised by each professor, and advisor of each student, are shown in Ta-

ble 1.[2] It can be seen that ROSS can effectively identify important linkages and related objects in databases, and its measure for strengths of join paths is more effective than simple measures such as join path length.

	Student advised by each prof	Advisor of student
ROSS	0.669	0.856
KOSS	0.491	0.821

Table 1: Accuracy ROSS and KOSS in relationship prediction

6.4. Answering Keyword Queries

In this experiment we test ROSS's performances in object search based on user queries. Both ROSS and KOSS are tested on the CS Dept database and Northwind database. For each database, 20 typical queries are written, which include both simple queries and complex queries on different relations. For each query, we provide the standard answer, which are one or more groups of tuples in the target relation, ranked according to their relationship to the source objects. For example, courses related to a student are courses taken by him; publications related to a professor via students are publications written by the students of the professor. The answers to these queries can be defined by one SQL query. The standard answers for some queries may contain tuples retrieved from multiple SQL queries, and need to be ranked. For example, the students related to database area are the students in database group, followed by students who take database courses.

To guarantee that the standard answers to different queries are provided in an objective and consistent way, some principles are used for ordering different relationships according to their semantic strengths. The principles are based on common senses about the tightness of relationships (e.g. strength of relationship between two persons is measured by how likely they know each other). Take CS Dept database as an example. Relationships between a person and a research area/problem are ranked as *work in group* > *publish paper* > *taking course*. Relationships between persons are ranked as *advise* > *coauthor* > *same group* > *taking course*.

The standard answer to a query usually contains a ranked list of several groups of tuples, and a measure is needed to evaluate the similarity between the answer of ROSS and the standard answer. Two answers should be considered identical if and only if they generate same groups of tuples, and give same ranks for the groups. Suppose the standard answer to is a list of tuples $t_1, \ldots t_n$, which are divided into $l+1$ groups $\{t_1, \ldots, t_{k_1}\}$, $\{t_{k_1+1}, \ldots, t_{k_2}\}$, ..., $\{t_{k_l+1}, \ldots, t_n\}$. We assign a *group score* (gs) for a group $\{t_{k_j+1}, \ldots, t_{k_{j+1}}\}$, which is the reciprocal of the highest rank of tuples in this group ($gs(t_i) = 1/(k_j+1)$ if $k_j < i \leq k_{j+1}$). $gs(t) = +\infty$ if t is not in the standard answer.

Given a query Q, ROSS or KOSS gives an answer A, which is a ranked list of tuples $t'_1, \ldots, t'_m$. The score of a tuple t'_i is defined as the the reciprocal of t'_i's rank $(1/i)$, or the group score of t'_i, whichever is lower. The score of the answer A is defined as the sum of each tuple's score, as following

$$s(A) = \sum_{i=1}^{m} \min\left(\frac{1}{i}, gs(t'_i)\right) \quad (5)$$

The optimal score is $\hat{s}(A) = \sum_{i=1}^{n} \frac{1}{i}$, which can be only achieved when tuples from different groups are ranked correctly. The accuracy of answer A is defined as the ratio between A's score and the optimal score $(s(A)/\hat{s}(A))$.

The 20 queries for CS Dept database is shown as follows. We use "target – ('source')" to represent the query "FIND target RELATEDTO ('source')". We use names like "John Doe" to represent professors and names like "Tom" to represent students. The two values after each query represent the accuracy of ROSS and KOSS.

Student – ('John Doe') [1,0.58]; Student – (('Mike Smith' via Publication) AND (area 'network')) [1,0.64]; Student – (area 'database') [0.95,1]; Student – (('John Doe') AND (area 'database')) [0.97,0.27]; Student – ('David') [0.88,0.77]; Professor – ('Tom') [1,0.71]; Professor – ('David' VIA advise) [1,1]; Professor – ('Ben' VIA course) [1,1]; Professor – (area 'artificial intelligence') [1,1]; Professor – (('Cathy') AND ('Jeff')) [1,1]; Professor – ('CS300') [1,1]; Professor – ('frequent pattern') [1,1]; Course – ('Cathy') [1,1]; Course – ('John Doe' VIA Student) [1,0.89]; Course – ((('Ben') OR ('David')) AND (area 'database')) [1,1]; Course – (Group 'algorithm') [1,0.61]; Publication – ('Steve Peterson') [1,1]; Publication – (area 'database' VIA student) [1,1]; Publication – (('John Doe') AND ('Tim Robber')) [1,1]; Publication – ('Peter Miller' VIA student) [1,0.41]

The 20 queries for Northwind database are Employee – ('Great Lakes Food Market') [1,1]; Employee – ('France' VIA Customer) [1,1]; Employee – (Product 'Ipoh Coffee') [1,1]; Employee – (('Japan') and ('Poland')) [0.64,0.64]; Employee – (Employee 'Nancy Davolio') [1,0.85]; Customer – ('Condiment') [1,1]; Customer – (Supplier 'Tokyo Traders') [1,1]; Customer – (Customer 'The Big Cheese') [0.97,0.94]; Product – (Employee 'Anne Dodsworth') [1,1]; Product – (('Around the Horn') and ('Seafood')) [1,1]; Product – (Product 'Filo Mix') [1,0.88]; Product – ('Spain') [1,1]; Product – (Customer 'Island Trading') [1,1]; Product – ((('Condiment') OR ('Confection')) AND ('London')) [1,1]; Supplier – (Customer 'Wilman Kala') [1,1]; Supplier – (Supplier 'Specialty Biscuits' VIA Category) [1,1]; Supplier – ('Berlin') [1,1]; Supplier – (('Dairy Product') and (Customer 'Old World Delicatessen')) [1,0.89]; Order – ('Seafood') [1,1]; Order – ((Country 'UK') AND (Product 'Steeleye Stout')) [1,0.31]

The performances of ROSS and KOSS are shown in Table 2. One can see that ROSS achieves good accuracy and efficiency on both databases.

2 One important reason that two systems achieve similar accuracy in predicting advisors is that, 63% of students who have advisors are in research groups with only one professor.

	CS Dept		Northwind	
Approach	Accuracy	Runtime	Accuracy	Runtime
Ross	0.990	0.54 sec	0.980	0.26 sec
Koss	0.844	0.34 sec	0.925	4.12 sec

Table 2: Performances of Ross and Koss

7. Related Work and Discussions

There have many studies on searching for different types of information (documents, XML, relational data, etc). Document search has been studied for decades in information retrieval, in which each document is considered separately. This cannot be applied on relational data, in which objects are connected together via different types of joins.

In recent years much attention has been paid to keyword-based search in relational databases. In [4, 3], a database is considered as a graph with objects as nodes and relationships as edges. The system in [4] retrieves objects from a target relation that are related to a set of source objects. In [3] a heuristical approach is proposed to find small trees that contain all keywords in the query. In [1, 9, 7] approaches are proposed to work on the schema graphs of databases, which are much smaller than the tuple graphs. They generate candidate join trees according to user query, then convert the join trees into SQL queries and retrieve results from backend databases. Similar approaches for XML are proposed in [2, 5, 8].

Although the above approaches provide good interfaces for keyword search in relational databases, they are not appropriate for searching for related objects. They use the lengths of join paths to measure the strength of relationship, which may lead to unsatisfactory results in many cases. For example, the link of coauthoring papers ($Professor \bowtie Publish \bowtie Publication \bowtie Publish \bowtie Student$) is considered to be weaker than the link of teaching student ($Professor \bowtie TeachCourse \bowtie Register \bowtie Student$). Another problem is that, the above approaches aim at finding minimum spanning trees of objects containing all keywords, without specifying the relationship among the objects. This may lead to results that do not feed the user's need.

Since we use selectivity to measure strengths of join paths, selectivity estimation becomes a major challenge. There have been thorough studies for selectivity estimation for decades [11, 10, 14, 13, 6]. However, these approaches work on given join paths and cannot be used to search for most selective paths in databases. Ross pre-computes selectivity of many short join paths, and combines them to estimate selectivity of longer paths. It converts the database schema into a graph, so that the shortest paths in graph correspond to the most selective join paths.

8. Conclusions

Object search in relational databases is an important task in knowledge discovery and informational retrieval from databases. In this paper we present the design and implementation of Ross, a relational object search system that handles complex queries. Ross adopts a new query format that enables the user to specify the relationship between the target objects and source objects. It combines both selectivity and lengths of join paths to measure the strengths of their relationships, which is more accurate than previous keyword query systems because different semantics of different joins are considered. We propose a novel approach for converting the database schema into a graph, so that the shortest paths in the graph correspond to best join paths in the database. A new stream-based system architecture is used to find the most related objects upon user's needs. We conduct comprehensive experiments to show the efficiency and effectiveness of Ross.

References

[1] S. Agrawal, S. Chaudhuri, G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. *ICDE*, 2002.

[2] S. Al-Khalifa, C. Yu, H. V. Jagadish. Querying Structured Text in an XML Database. *SIGMOD*, 2003.

[3] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. *ICDE*, 2002.

[4] R. Goldman, N. Shivakumar, S. Venkatasubramanian, H. Garcia-Molina. Proxmity Search in Databases. *VLDB*, 1998.

[5] L. Guo, F. Shao, C. Botev, J. Shanmugasundaram. XRANK: Ranked Keyword Search Over XML Documents. *SIGMOD*, 2003.

[6] L. Getoor, B. Taskar, D. Koller. Selectivity Estimation using Probabilistic Models. *SIGMOD*, 2001.

[7] V. Hristidis, L. Gravano, Y. Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. *VLDB*, 2003.

[8] V. Hristidis, Y. Papakonstantinou, A. Balmin. Keyword Proximity Search on XML Graphs. *ICDE*, 2003.

[9] V. Hristidis, Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. *VLDB*, 2002.

[10] R. Lipton, J. Naughton, D. Schneider. Practical Selectivity Estimation Through Adaptive Sampling. *SIGMOD*, 1990.

[11] C. Lynch. Selectivity Estimation and Query Optimization in Large Databases with Highly Skewed Distributions of Column Values. *SIGMOD*, 1988.

[12] E.Q.V. Martins, J.L.E. dos Santos. A New Shortest Paths Ranking Algorithm. *Investigacao Operacional*, 2000.

[13] Y. Matias, J.S. Vitter, M. Wang. Wavelet-Based Histograms for Selectivity Estimation. *SIGMOD*, 1998.

[14] V. Poosala, Y. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. *VLDB*, 1997.

[15] TPC Benchmark H. http://www.tpc.org/tpch/default.asp.

[16] J.Y. Yen. Finding the K Shortest Loopless Paths in a Network. *Management Science*, 17, 1971.

Assumption-Free Anomaly Detection in Time Series

Li Wei Nitin Kumar Venkata Lolla Eamonn Keogh Stefano Lonardi Chotirat Ann Ratanamahatana
University of California - Riverside
Department of Computer Science & Engineering
Riverside, CA 92521, USA
{wli, nkumar, vlolla, eamonn, stelo, ratana}@cs.ucr.edu

Abstract

Recent advancements in sensor technology have made it possible to collect enormous amounts of data in real time. However, because of the sheer volume of data most of it will never be inspected by an algorithm, much less a human being. One way to mitigate this problem is to perform some type of anomaly (novelty / interestingness/ surprisingness) detection and flag unusual patterns for further inspection by humans or more CPU intensive algorithms. Most current solutions are "custom made" for particular domains, such as ECG monitoring, valve pressure monitoring, etc. This customization requires extensive effort by domain expert. Furthermore, hand-crafted systems tend to be very brittle to concept drift.

In this demonstration, we will show an online anomaly detection system that does not need to be customized for individual domains, yet performs with exceptionally high precision/recall. The system is based on the recently introduced idea of time series bitmaps. To demonstrate the universality of our system, we will allow testing on independently annotated datasets from domains as diverse as ECGs, Space Shuttle telemetry monitoring, video surveillance, and respiratory data. In addition, we invite attendees to test our system with any dataset available on the web.

1. Introduction

Recent advancements in sensor technology have made it possible to collect enormous amounts of data in real time. However, because of the sheer volume of data most of it is never inspected by an algorithm, much less a human being. One way to mitigate this problem is to perform some type of anomaly (novelty / interestingness/ surprisingness) detection and to flag unusual patterns for future inspection by humans or more CPU intensive algorithms. Most current solutions are "custom made" for particular domains, such as ECG monitoring, valve pressure monitoring, etc. This customization requires extensive effort by domain experts. Furthermore hand-crafted systems tend to be very brittle to concept drift.

In this demonstration, we will show an online anomaly detection system that does not need to be customized for individual domains, yet performs with exceptionally high precision/recall. The system is based on the recently introduced idea of time series bitmaps [11]. It allows users to efficiently navigate through a time series of arbitrary length and identify portions that require further investigation. Figure 1 illustrates the graphical interface of our system[1].

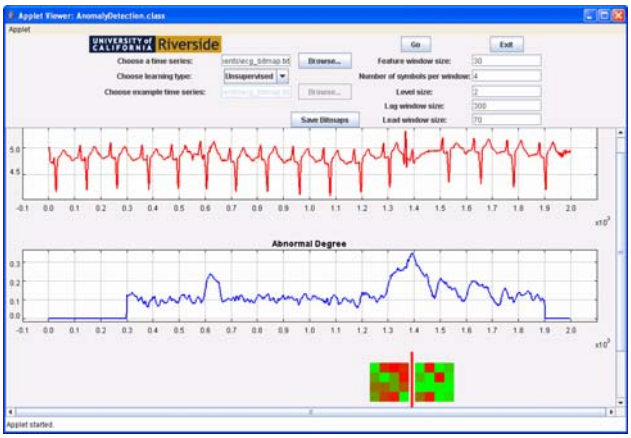


Figure 1. A snapshot of the anomaly detection tool.

To demonstrate the universality of our system, we will allow testing on independently annotated datasets from domains as diverse as ECGs, Space Shuttle telemetry monitoring, video surveillance, and respiratory data. In addition, we invite attendees to test our system with any dataset available on the web.

2. Background and Related Work

In this section, we give brief reviews of chaos games and symbolic representations of time series, which are at the heart of our anomaly detection technique.

2.1 Chaos Game Representations

Our visualization technique is partly inspired by an algorithm to draw fractals called the *Chaos game* [1]. The method can produce a representation of DNA sequences, in which both local and global patterns are displayed.

The basic idea is to map frequency counts of DNA substrings of length L into a $2L$ by $2L$ matrix as shown in Figure 2, then color-code these frequency counts. From our point of view, the crucial observation is that the CGR

[1] We encourage the interested reader to visit [5] to view full color examples of all figures in this work.

representation of a sequence allows the investigation of the patterns in sequences, giving the human eye a possibility to recognize hidden structures.

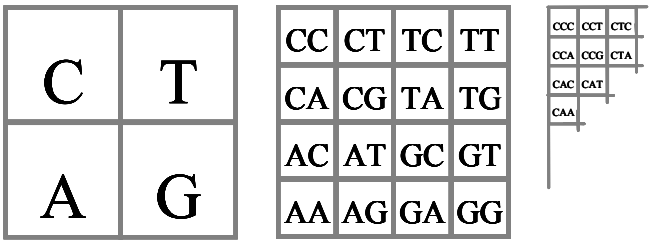


Figure 2. The quad-tree representation of a sequence over the alphabet {A,C,G,T} at different levels of resolution.

We can get a hint of the potential utility of the approach if, for example, we take the first 5,000 symbols of the mitochondrial DNA sequences of four familiar species and use them to create their own file icons. Figure 3 below illustrates this. Note that *Pan troglodytes* is the familiar Chimpanzee, and *Loxodonta africana* and *Elephas maximus* are the African and Indian Elephants, respectively. Even if we did not know these particular species, we would have no problem recognizing that there are two pairs of highly related species being considered.

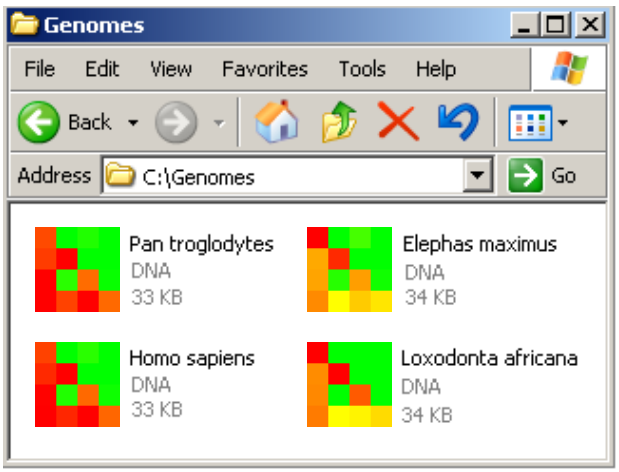


Figure 3. The bitmap representation of the gene sequences of four species.

With respect to the non-genetic sequences, Joel Jeffrey noted, *"The CGR algorithm produces a CGR for any sequence of letters"* [4]. However, it is only defined for discrete sequences, and most time series are real valued.

The results in Figure 3 encouraged us to try a similar technique on real valued time series data and investigate the utility of such a representation on the data mining task of anomaly detection. Since CGR involves treating a data input as an abstract string of symbols, a discretization method is necessary to transform continuous time series data into discrete domain. For this purpose, we used the Symbolic Aggregate approXimation (SAX) [8], which we review below.

2.2 Symbolic Time Series Representations

While there are at least 200 techniques in the literature for converting real valued time series into discrete symbols, the SAX technique of Lin *et. al.* [8] is unique and ideally suited for data mining. SAX is the only symbolic representation that allows the lower bounding of the distances in the original space.

The SAX representation is created by taking a real valued signal and dividing it into equal sized sections. The mean value of each section is then calculated. By substituting each section with its mean, a reduced dimensionality piecewise constant approximation of the data is obtained. This representation is then discretized in such a manner as to produce a word with approximately equi-probable symbols. Figure 4 shows a short time series being converted into the SAX word **baabccbc**.

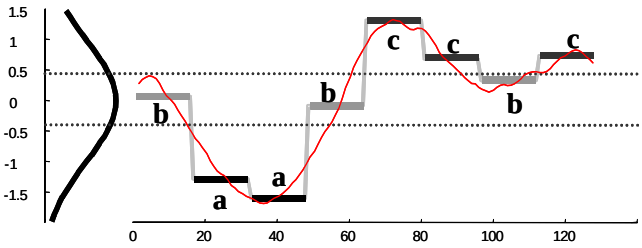


Figure 4. A real valued time series can be converted to the SAX word baabccbc.

It has been pointed out that when processing very long time series, it is not necessarily a good idea to convert the entire time series into a *single* SAX word [11]. Therefore, for long time series, we slide a shorter window, which is called feature window, across it and obtain a set of shorter SAX words.

Note that the user must choose both the length of the sliding feature window *N*, and the number *n* of equal sized sections in which to divide *N* (as we will see, there is no choice to be made for alphabet size). A good choice for *N* should reflect the natural scale at which the events occur in the time series. For example, for ECGs, this is about the length of one or two heartbeats. A good value for *n* depends on the complexity of the signal. Intuitively, one would like to achieve a good compromise between fidelity of approximation and dimensionality reduction. As we shall see, the proposed technique is not too sensitive to parameter choices.

3. Time Series Anomaly Detection

3.1 Time Series Bitmaps

At this point, we have seen that the Chaos game bitmaps can be used to visualize discrete sequences and that the SAX representation is a discrete time series representation that has demonstrated great utility for data mining. It is natural to consider combining these ideas.

The *Chaos game* bitmaps are defined for sequences with an alphabet size of four. SAX can produce strings on any alphabet sizes. As it turns out, many authors have reported a cardinality of four as an excellent choice for diverse datasets on assorted problems [2][3][6][7][8][9].

We need to define an initial ordering for the four SAX symbols **a**, **b**, **c**, and **d**. We use simple alphabetical ordering as shown in Figure 5.

After converting the original raw time series into the SAX representation, we can count the frequencies of SAX "subwords" of length *L*, where *L* is the desired level of recursion. Level 1 frequencies are simply the raw counts of the four symbols. For level 2, we count pairs of subwords of size 2 (**aa**, **ab**, **ac**, etc.). Note that we only count subwords taken from individual SAX words. For example, in the SAX representation in Figure 5 *middle right*, the last symbol of the first line is **a**, and the first symbol of the second word is **b**. However, we do not count this as an occurrence of **ab**.

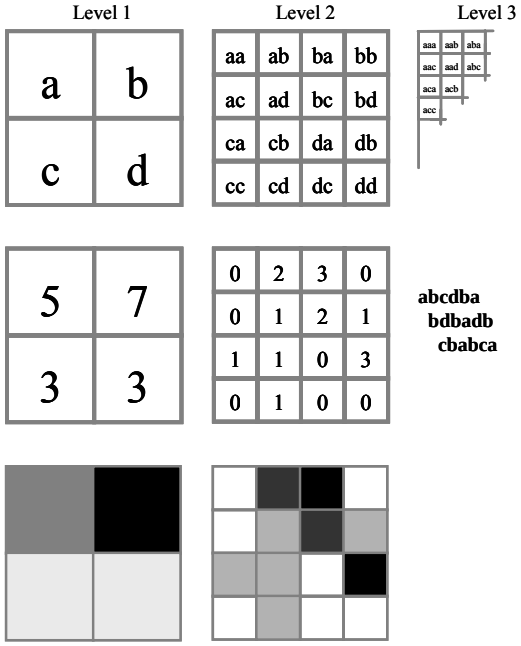


Figure 5. The generation of time series bitmaps.

Once the raw counts of all subwords of the desired length have been obtained and recorded in the corresponding pixel of the grid, we normalize the frequencies by dividing it by the largest value. The pixel values *P* thus range from 0 to 1. The final step is to map these values to colors. In the example above, we mapped to grayscale, with 0 = *white*, 1 = *black*. However, it is generally recognized that grayscale is not *perceptually uniform* [10]. A color space is said to be perceptually uniform if small changes to a pixel value are approximately equally perceptible across the range of that value. For all images in this paper, we encode the pixels values to be [*P*, 1-*P*, 0] in the RGB color space.

For bitmaps with same size, we define the distance between them as the summation of the square of the distance between each pair of pixels. More formally, for two $n \times n$ bitmaps *BA* and *BB*, the distance between them is defined as $dist(BA, BB) = \sum_{i=1}^{n}\sum_{j=1}^{n}(BA_{ij} - BB_{ij})^2$.

3.2 Anomaly Detection

We create two concatenated windows and slide them together across the sequence. The latter one is called *lead* window, showing how far to look ahead for anomalous patterns. A reasonable value would be two or three times the length of the feature window. The former one is called *lag* window, whose size represents how much memory of the past to remember. Usually, it should be at least as long as the lead window. We convert each window into the SAX representation, count the frequencies of SAX "subwords" at the desired level, and get the corresponding bitmaps. The distance between the two bitmaps is measured and reported as an anomaly score at each time instance, and the bitmaps are drawn to visualize the similarities and differences between the two windows.

There are two ways to use the tool, unsupervised (one time series) and supervised (two time series). For unsupervised use, the user must specify the size of the lag window. For supervised use, the user must specify a time series file that he/she believes contains normal behavior for the system. For example, this could be 10 minutes of ECGs that are known to be normal, or a trace from a successful space mission. In this case, the entire training time series can be imagined as the lag window.

At each "step" of the sliding window we can incrementally ingress a new data point, and egress an old data point in constant time (updating only two pixels of each bitmap). Hence, the time complexity is linear in the length of the time series.

4. Experimental Evaluation

To demonstrate the universality of our system, we tested on independently annotated datasets from domains as diverse as ECGs, Space Shuttle telemetry monitoring, video surveillance, and respiratory data. Here we show only a subset of the experimental results due to space limitations. Our approach is also effective on time series

clustering and classification [11], but we focus on its utility for anomaly detection here. We urge the interested reader to consult [5] for large-scale color reproductions and additional details.

Figure 6 illustrates a subsection of an ECG data. A cardiologist annotated two premature ventricular contractions at approximately the 0.4 and 1.1 mark, respectively, and a supraventricular escape beat at about the 1.0 mark. Our approach easily detects all the three anomalies.

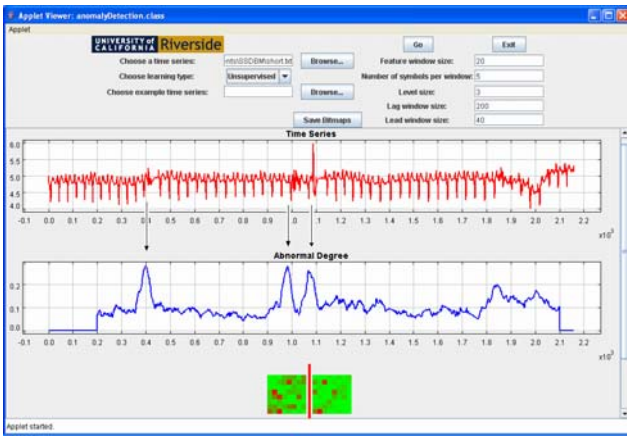


Figure 6. *Top*) A subsection of an ECG dataset. *Middle*) The abnormal score shows three strong peaks for the anomalous heartbeats. *Bottom*) The bitmaps before and after the third peak.

Figure 7 shows a very complex and noisy ECG. But according to a cardiologist, there is only one abnormal heartbeat at approximately the 0.23 mark. Our tool easily finds it.

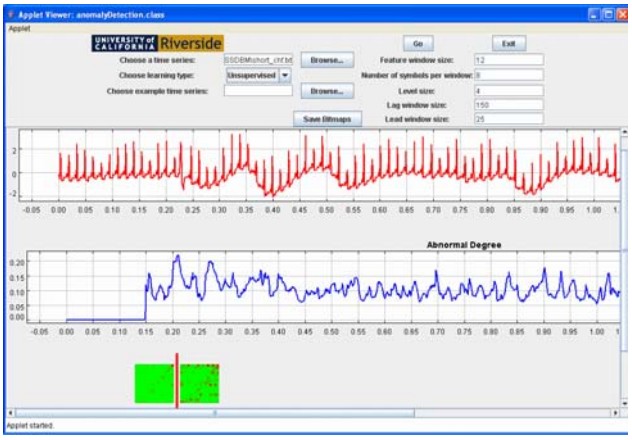


Figure 7. *Top*) A subsection of an ECG dataset. *Middle*) The abnormal score shows a strong peak for the anomalous heartbeat. *Bottom*) The bitmaps before and after the strong peak.

5. Demonstration Plan

Our demonstration will consist of the following three parts.

- First, we will present some real-world applications in which our technique can be applied. These examples will provide the audience with insights into the task of time series anomaly detection.
- Second, by using real-world datasets from diverse domains, we will show the experimental evaluation of our system.
- Finally, we will invite audience to play the tool interactively themselves. The audience will be encouraged to test their own datasets.

Reproducible Results Statement: In the interests of competitive scientific inquiry, all datasets used in this work are available at the following URL [5]. This research was partly funded by the National Science Foundation under grant IIS-0237918.

References

[1] Barnsley, M.F., & Rising, H. (1993). *Fractals Everywhere*, second edition, Academic Press.

[2] Celly, B. & Zordan, V. B. (2004). Animated People Textures. In proceedings of the 17th International Conference on Computer Animation and Social Agents. Geneva, Switzerland.

[3] Chiu, B., Keogh, E., & Lonardi, S. (2003). Probabilistic Discovery of Time Series Motifs. In the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

[4] Jeffrey, H.J. (1992). Chaos Game Visualization of Sequences. Comput. & Graphics 16, pp. 25-33.

[5] Keogh, E. http://www.cs.ucr.edu/~wli/SSDBM05/

[6] Keogh, E., Lonardi, S., & Ratanamahatana, C. (2004). Towards Parameter-Free Data Mining. In proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

[7] Lin, J., Keogh, E., Lonardi, S., Lankford, J.P. & Nystrom, D.M. (2004). Visually Mining and Monitoring Massive Time Series. In proceedings of the 10th ACM SIGKDD.

[8] Lin, J., Keogh, E., Lonardi, S. & Chiu, B. (2003) A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery.

[9] Tanaka, Y. & Uehara, K. (2004). Motif Discovery Algorithm from Motion Data. In proceedings of the 18th Annual Conference of the Japanese Society for Artificial Intelligence (JSAI). Kanazawa, Japan.

[10] Wyszecki, G. (1982). Color science: Concepts and methods, quantitative data and formulae, 2nd edition. New York, Wiley, 1982.

[11] Kumar, N., Lolla N., Keogh, E., Lonardi, S., Ratanamahatana, C. & Wei, L. (2005). Time-series Bitmaps: A Practical Visualization Tool for Working with Large Time Series Databases. SIAM 2005 Data Mining Conference.

Session 9: Summarization

A Fast Approximation Scheme for Probabilistic Wavelet Synopses

Antonios Deligiannakis
University of Maryland
adeli@cs.umd.edu

Minos Garofalakis
Bell Laboratories
minos@research.bell-labs.com

Nick Roussopoulos
University of Maryland
nick@cs.umd.edu

Abstract

Several studies have demonstrated the effectiveness of Haar wavelets in reducing large amounts of data down to compact wavelet synopses *that can be used to obtain fast, accurate approximate query answers. While Haar wavelets were originally designed for minimizing the overall root-mean-squared (i.e., L_2-norm) error in the data approximation, the recently-proposed idea of* probabilistic wavelet synopses *also enables their use in minimizing other error metrics, such as the relative error in individual data-value reconstruction, which is arguably the most important for approximate query processing. Known construction algorithms for probabilistic wavelet synopses employ probabilistic schemes for coefficient thresholding that are based on optimal Dynamic-Programming (DP) formulations over the error-tree structure for Haar coefficients. Unfortunately, these (exact) schemes can scale quite poorly for large data-domain and synopsis sizes. To address this shortcoming, in this paper, we introduce a novel, fast* approximation scheme *for building probabilistic wavelet synopses over large data sets. Our algorithm's running time is near-linear in the size of the data-domain (even for very large synopsis sizes) and proportional to $1/\epsilon$, where ϵ is the desired approximation guarantee. The key technical idea in our approximation scheme is to make exact DP formulations for probabilistic thresholding* much "sparser"*, while ensuring a maximum relative degradation of ϵ on the quality of the approximate synopsis, i.e., the desired approximation error metric. Extensive experimental results over synthetic and real-life data clearly demonstrate the benefits of our proposed techniques.*

1. Introduction

Approximate query processing over compact, precomputed data synopses has attracted a lot of interest recently as a viable solution for dealing with complex queries over massive amounts of data in interactive decision-support and data-exploration environments. For several of these application scenarios, exact answers are not required, and users may in fact prefer fast, approximate answers to their queries. Examples include the initial, exploratory drill-down queries in ad-hoc data mining systems, where the goal is to quickly identify the "interesting" regions of the underlying database; or, aggregation queries in decision-support systems where the full precision of the exact answer is not needed and the first few digits of precision suffice (e.g., the leading digits of a total in the millions or the nearest percentile of a percentage) [1, 2, 6, 12].

Background and Earlier Results. *Haar wavelets* are a mathematical tool for the hierarchical decomposition of functions with several successful applications in signal and image processing [13, 18]. A number of recent studies has also demonstrated the effectiveness of the Haar wavelet decomposition as a data-reduction tool for database problems, including selectivity estimation [14] and approximate query processing over massive relational tables [2, 19] and data streams [9, 15]. Briefly, the key idea is to apply the decomposition process over an input data set along with a thresholding procedure in order to obtain a compact data synopsis comprising of a selected small set of *Haar wavelet coefficients*. The results of the recent research studies of Matias, Vitter and Wang [14, 19], Chakrabarti et al. [2, 3], and others [5, 17] have demonstrated that fast and accurate approximate query processing engines can be designed to operate solely over such compact *wavelet synopses*.

Until very recently, a major criticism of wavelet-based approximate query processing techniques has been the fact that unlike, e.g., random samples, conventional wavelet synopses (such as those used in all the above-cited studies) cannot provide useful guarantees on the quality of approximate answers. The problem here is that coefficients for such conventional synopses are typically chosen in a greedy fashion in order to optimize the overall root-mean-squared (i.e., L_2-norm) error in the data approximation. However, as pointed out by Garofalakis and Gibbons [7], conventional, L_2-optimized wavelet synopses can result in approximate answers of widely-varying quality (even within the same data set) and approximation errors that are heavily biased towards certain regions of the underlying data domain. Their proposed solution, termed *probabilistic wavelet synopses* [7], employs the idea of *randomized coefficient rounding* in conjunction with *Dynamic-Programming-based* thresholding schemes specifically tuned for optimiz-

ing the *maximum relative error* in the approximate reconstruction of individual data values. By optimizing for relative error (with a sanity bound), which is arguably the most important metric for approximate query answers, probabilistic wavelet synopses offer drastic reductions in the approximation error over conventional deterministic techniques and, furthermore, enable unbiased data reconstruction with meaningful, non-trivial *error guarantees* for reconstructed values [7]. [1]

Our Contributions. The Dynamic-Programming (DP) algorithms of [7] for constructing probabilistic wavelet synopses are based on an *optimal, continuous DP formulation* over the error-tree structure for Haar coefficients, in conjunction with the idea of *quantizing* the possible choices for synopsis-space allocation using an integer parameter $\mathsf{q} > 1$ (in other words, fractional space is allotted to coefficients in multiples of $1/\mathsf{q}$). Unfortunately, the problem with these exact (modulo the quantization) DP techniques is that they can scale poorly for large data-domain and synopsis sizes – with a domain size of N and synopsis storage of B, the worst-case running time of the optimized algorithm presented in [7] (which uses binary-search to optimize the DP search) is $O(N\mathsf{q}^2 B \log(\mathsf{q}B))$, which becomes $O(N^2\mathsf{q}^2 \log(N\mathsf{q}))$ for large synopsis sizes $B = \Theta(N)$. (Given that today's personal computers and workstations typically come equipped with Gigabytes of main memory, it is quite realistic to expect large synopsis sizes when dealing with massive data sets.) Our own experience with the DP schemes in [7] has demonstrated that the times required for building a probabilistic wavelet synopsis can increase very rapidly for large domain sizes N and synopsis sizes B; this certainly raises some concerns with respect to the applicability of probabilistic wavelet techniques on massive, real-life data sets. Note that large domain sizes in the range of 10^5–10^7 are not at all uncommon, e.g., for massive time-series data sets where one or more readings/measurements are continuously recorded on every time-tick.

To address these concerns, we propose a novel, fast *approximation scheme* for building probabilistic wavelet synopses over large data sets. Given a quantization parameter q and a desired approximation factor ϵ, our algorithm can be used to build a probabilistic synopsis of *any size* $B \leq N$ in worst-case time of $O(N\mathsf{q} \log \mathsf{q} \min\{\log N \log R/\epsilon, B\mathsf{q}\})$ (where R is roughly proportional to the maximum absolute Haar-coefficient value in the decomposition), while guaranteeing that the quality of the final solution is within a factor of $(1+\epsilon)$ of that obtained by the (exact) techniques of Garofalakis and Gibbons [7] for the same problem instance. (Note that the running time of our algorithm actually represents an improvement over the techniques in [7] even when computing the exact, optimal solution.) In a nutshell, the key technical idea in our proposed approximation scheme is to make the original DP formulations in [7] *much "sparser"*, while ensuring a maximum relative degradation of $(1+\epsilon)$ on the quality of the approximate solution, i.e., the desired maximum error metric. This is accomplished by restricting the DP search to a carefully-chosen, logarithmically-small subset of *"breakpoints"* that cover the entire range of possible space allotments within the required error guarantee. Our results clearly validate our approach, demonstrating that our algorithm (1) exhibits significantly smaller running times, often by *more than one or even two orders of magnitude*, than the exact DP solution; and (2) typically produces significantly tighter approximations than the specified $(1+\epsilon)$ (worst-case) guarantee.

Roadmap. The remainder of this paper is organized as follows. Section 2 gives background material on wavelets, as well as conventional and probabilistic wavelet synopses. In Section 3, we discuss our approximation scheme for constructing probabilistic wavelet synopses in detail. Section 4 describes the results of our empirical study and, finally, Section 5 gives some concluding remarks.

2. Preliminaries

The Haar Wavelet Transform. Wavelets are a useful mathematical tool for hierarchically decomposing functions in ways that are both efficient and theoretically sound. Broadly speaking, the wavelet decomposition of a function consists of a coarse overall approximation along with detail coefficients that influence the function at various scales [18]. Suppose that we are given the one-dimensional data vector A containing the $N = 8$ data values $A = [2, 2, 0, 2, 3, 5, 4, 4]$. The Haar wavelet transform of A can be computed as follows. We first average the values together pairwise to get a new "lower-resolution" representation of the data with the following average values $[2, 1, 4, 4]$. In other words, the average of the first two values (that is, 2 and 2) is 2, that of the next two values (that is, 0 and 2) is 1, and so on. Obviously, some information has been lost in this averaging process. To be able to restore the original values of the data array, we need to store some *detail coefficients*, that capture the missing information. In Haar wavelets, these detail coefficients are simply the differences of the (second of the) averaged values from the computed pairwise average. Thus, in our simple example, for the first pair of averaged values, the detail coefficient is 0 since $2 - 2 = 0$, for the second we need to store -1 since $1 - 2 = -1$. Note that no information has been lost in this process – it is fairly simple to reconstruct the eight values of the original data array from the lower-resolution array containing the four av-

1 In more recent work, Garofalakis and Kumar [8] have proposed optimal *deterministic* wavelet-thresholding schemes for relative error metrics; still, their optimal algorithms are significantly more expensive computationally than the probabilistic schemes in [7], and do not directly extend to multi-dimensional data.

erages and the four detail coefficients. Recursively applying the above pairwise averaging and differencing process on the lower-resolution array containing the averages, we get the following full decomposition:

Resolution	Averages	Detail Coefficients
3	[2, 2, 0, 2, 3, 5, 4, 4]	—
2	[2, 1, 4, 4]	[0, -1, -1, 0]
1	[3/2, 4]	[1/2, 0]
0	[11/4]	[-5/4]

The *wavelet transform* (also known as the *wavelet decomposition*) of A is the single coefficient representing the overall average of the data values followed by the detail coefficients in the order of increasing resolution. Thus, the one-dimensional Haar wavelet transform of A is given by $W_A = [11/4, -5/4, 1/2, 0, 0, -1, -1, 0]$. Each entry in W_A is called a *wavelet coefficient*. The main advantage of using W_A instead of the original data vector A is that for vectors containing similar values most of the detail coefficients tend to have very small values. Thus, eliminating such small coefficients from the wavelet transform (i.e., treating them as zeros) introduces only small errors when reconstructing the original data, resulting in a very effective form of lossy data compression [18]. Furthermore, the Haar wavelet decomposition can also be extended to *multi-dimensional* data arrays through natural generalizations of the one-dimensional decomposition process described above. Multi-dimensional Haar wavelets have been used in a wide variety of applications, including approximate query answering over complex decision-support data sets [2, 19].

Error Tree and Conventional Wavelet Synopses. A helpful tool for exploring the properties of the Haar wavelet decomposition is the *error tree* structure [14]. The error tree is a hierarchical structure built based on the wavelet transform process. Figure 1 depicts the error tree for our example data vector A. Each internal node c_i ($i = 0, \ldots, 7$) is associated with a wavelet coefficient value, and each leaf d_i ($i = 0, \ldots, 7$) is associated with a value in the original data array; in both cases, the index i denotes the positions in the data array or error tree. For example, c_0 corresponds to the overall average of A. The resolution levels l for the coefficients (corresponding to levels in the tree) are also depicted. We use the terms "node" and "coefficient" interchangeably in what follows.

Given a node u in an error tree T, let $\mathtt{path}(u)$ denote the set of all proper ancestors of u in T (i.e., the nodes on the path from u to the root of T, including the root but not u) with non-zero coefficients. A key property of the Haar wavelet decomposition is that the reconstruction of any data value d_i depends only on the values of coefficients on $\mathtt{path}(d_i)$; more specifically, we have $d_i = \sum_{c_j \in \mathtt{path}(d_i)} \delta_{ij} \cdot c_j$, where $\delta_{ij} = +1$ if d_i is in the left child subtree of c_j or $j = 0$, and $\delta_{ij} = -1$ otherwise. For example, in Figure 1, $d_4 = c_0 - c_1 + c_6 = \frac{11}{4} - (-\frac{5}{4}) + (-1) = 3$.

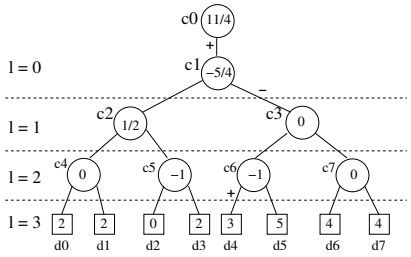


Figure 1. Error tree for example array A ($N = 8$).

Given a limited amount of storage for building a *wavelet synopsis* of the input data array A, a thresholding procedure retains a certain number $B \ll N$ of the coefficients as a highly-compressed approximate representation of the original data (the remaining coefficients are implicitly set to 0). Conventional coefficient thresholding is a deterministic process that seeks to minimize the overall root-mean-squared error (L_2 *error norm*) of the data approximation [18] by retaining the B largest wavelet coefficients in *absolute normalized value* [18]. L_2 coefficient thresholding has also been the method of choice for the bulk of existing work on Haar-wavelets applications in the data-reduction and approximate query processing domains [2, 14, 15, 19].

Probabilistic Wavelet Synopses. Unfortunately, wavelet synopses optimized for overall L_2 error using the above-described process may not always be the best choice for approximate query processing systems. As observed in a recent study by Garofalakis and Gibbons [7], such conventional wavelet synopses suffer from several important problems, including the introduction of severe bias in the data reconstruction and wide variance in the quality of the data approximation, as well as the lack of non-trivial guarantees for individual approximate answers. To address these shortcomings, their work introduces *probabilistic wavelet synopses*, a novel approach for constructing data summaries from wavelet-transform arrays. In a nutshell, their key idea is to apply a probabilistic thresholding process based on *randomized rounding* [16], that randomly rounds coefficients either up to a larger rounding value or down to zero, so that the value of each coefficient is correct *on expectation*. More formally, each non-zero wavelet coefficient c_i is associated with a *rounding value* λ_i and a corresponding *retention probability* $y_i = \frac{c_i}{\lambda_i}$ such that $0 < y_i \leq 1$, and the value of coefficient c_i in the synopsis becomes a random variable $C_i \in \{0, \lambda_i\}$, where,

$$C_i = \begin{cases} \lambda_i & \text{with probability } y_i \\ 0 & \text{with probability } 1 - y_i. \end{cases}$$

In other words, a probabilistic wavelet synopsis essentially "rounds" each non-zero wavelet coefficient c_i *independently* to either λ_i or zero by flipping a biased coin with

$$M^*[i,\beta_i] = \begin{cases} \min\limits_{\substack{y_i \in (0, \min\{1,\beta_i\}];\\ b_L \in [0, \beta_i - y_i]}} \left\{ \max\left\{ \begin{array}{l} \frac{\mathrm{Var}(i,y_i)}{\mathrm{Norm}(2i)} + M^*[2i, b_L]\,, \\ \frac{\mathrm{Var}(i,y_i)}{\mathrm{Norm}(2i+1)} + M^*[2i+1, \beta_i - y_i - b_L] \end{array} \right\} \right\} & \text{if } i < N,\ c_i \neq 0, \text{ and } \beta_i > 0 \\ \min_{b_L \in [0,\beta_i]} \{ \max\{ M^*[2i, b_L]\,,\ M^*[2i+1, \beta_i - b_L] \} \} & \text{if } i < N \text{ and } c_i = 0 \\ 0 & \text{if } i \geq N \\ \infty & \text{otherwise} \end{cases} \quad (1)$$

success probability y_i. Note that the above rounding process is *unbiased*; that is, the expected value of each rounded coefficient is $\mathrm{E}[C_i] = \lambda_i \cdot y_i + 0 \cdot (1 - y_i) = c_i$, i.e., the actual coefficient value, while its variance is

$$\mathrm{Var}(i, y_i) = \mathrm{Var}(C_i) = (\lambda_i - c_i) \cdot c_i = \frac{1 - y_i}{y_i} \cdot c_i^2 \quad (2)$$

and the expected size of the synopsis is simply $\mathrm{E}[|\text{synopsis}|] = \sum_{i|c_i \neq 0} y_i = \sum_{i|c_i \neq 0} \frac{c_i}{\lambda_i}$. Thus, since each data value can be reconstructed as a simple linear combination of wavelet coefficients, and by linearity of expectation, it is easy to see that probabilistic wavelet synopses guarantee unbiased approximations of individual data values as well as range-aggregate query answers [7].

Garofalakis and Gibbons [7] propose several different algorithms for building probabilistic wavelet synopses. The key, of course, is to select the coefficient rounding values $\{\lambda_i\}$ such that some desired error metric for the data approximation is minimized while not exceeding a prescribed space limit B for the synopsis (i.e., $\mathrm{E}[|\text{synopsis}|] \leq B$). Their winning strategies are based on formulating appropriate *Dynamic-Programming (DP)* recurrences over the Haar error-tree that explicitly minimize either (a) the maximum normalized standard error (MinRelVar), or (b) the maximum normalized bias (MinRelBias), for each reconstructed value in the data domain. As explained in [7], the rationale for these probabilistic error metrics is that they are directly related to the *maximum relative error* (with an appropriate *sanity bound* S)[2] in the approximation of individual data values based on the synopsis; that is, both the MinRelVar and MinRelBias schemes try to (probabilistically) control the quantity $\max_i\{\frac{|\hat{d}_i - d_i|}{\max\{|d_i|, \mathsf{S}\}}\}$, where $\hat{d}_i$ denotes the data value reconstructed based on the wavelet synopsis. Note, of course, that $\hat{d}_i$ is again a *random variable*, defined as the ± 1 summation of all (independent) coefficient random variables on $\mathtt{path}(d_i)$. Bounding the maximum relative error in the approximation also allows for meaningful *error guarantees* to be provided on reconstructed data values [7].

As an example, Equation (1) depicts the DP recurrence in [7] for minimizing the maximum squared Normalized Standard Error (NSE^2) in the data reconstruction, defined as

$$\max_i \mathrm{NSE}^2(\hat{d}_i) = \max_i \frac{\mathrm{Var}(\hat{d}_i)}{\max\{d_i^2, \mathsf{S}^2\}},$$

where $\mathrm{Var}(\hat{d}_i) = \sum_{c_j \in \mathtt{path}(d_i)} \mathrm{Var}(j, y_j)$. $M^*[i, \beta_i]$ here denotes the minimum value of the maximum *squared* NSE (i.e., NSE^2) among all data values in the subtree of the error-tree rooted at coefficient c_i assuming a space budget of β_i, and $\mathrm{Norm}(i) = \max\{d_{\min}(i)^2, \mathsf{S}^2\}$, where $d_{\min}(i)$ is the minimum absolute data value under c_i's subtree, is a normalization term for that subtree. (Indices $2i$ and $2i+1$ in the recurrence correspond to the left and right child (respectively) of c_i in the error-tree structure (Figure 1).) Intuitively, the DP recurrence in Equation (1) states that, for a given space budget β_i at c_i, the optimal fractional-storage allotments $\{y_i\}$ and the corresponding maximum NSE^2 are fixed by minimizing the larger of the costs for paths via c_i's two child subtrees (including the root in all paths), where the cost for a path via a subtree is the sum of: (1) the variance penalty incurred at c_i itself, assuming a setting of y_i, divided by the normalization term for that subtree, and (2) the optimal cost for the subtree, assuming the given space budget. This minimization, of course, is over all possible values of y_i and, given a setting of y_i, over all possible allotments of the remaining $\beta_i - y_i$ space "units" amongst the two child subtrees of c_i. Of course, if $c_i = 0$ then no space budget needs to be allocated to node i, which results in the simpler recurrence in the second clause of Equation (1). Finally, data-value nodes (characterized by indices $i \geq N$, see Figure 1) cost no space and incur no cost, and the "otherwise" clause handles the case where we have a non-zero coefficient but zero budget ($c_i \neq 0$ and $\beta_i = 0$).

As demonstrated in [7], the DP recurrence in Equation (1) characterizes the optimal solution to the maximum NSE^2 minimization problem for the case of *continuous* fractional-storage allotments $y_i \in (0, 1]$ (modulo certain technical conditions that may require small "perturbations" of zero coefficients [7]). A similar DP recurrence can also be given for the maximum normalized bias metric. Their MinRelVar and MinRelBias algorithms then proceed by *quantizing the solution space*; that is, they assume the storage allotment variables y_i and b_L in Equation (1) to take values from a discrete set of choices corresponding to integer multiples of $1/\mathsf{q}$, where $\mathsf{q} > 1$ is an input integer parameter to the algorithms. (For instance, $y_i \in \{0, \frac{1}{\mathsf{q}}, \frac{2}{\mathsf{q}}, \ldots, 1\}$ – larger values of q imply results

2 The role of the sanity bound is to ensure that relative-error numbers are not unduly dominated by small data values [11, 19].

closer to the optimal, continuous solution.) Furthermore, both MinRelVar and MinRelBias cap the variance of a coefficient c at c^2, thus allowing for zero-space allotments to unimportant coefficients (this also implies that non-zero allotments of size $\leq \frac{1}{2}$ are useless, as they result in larger variance (Equation (2)) while utilizing more space).The running time of their (quantized) MinRelVar and MinRelBias algorithms is $O(N\mathsf{q}^2 B \log(\mathsf{q}B))$ with an overall space requirement of $O(N\mathsf{q}B)$ (and an in-memory working-set size of $O(\mathsf{q}B \log N)$); furthermore, their techniques also naturally extend to multi-dimensional data and wavelets, with a reasonable increase in time and space complexity [7]. Experimental results over synthetic and real-life data in [7] have demonstrated the superiority of MinRelVar and MinRelBias probabilistic synopses as an approximate query answering tool over conventional wavelet synopses. In our discussion, we use $M^*_{\mathsf{q}}[i, \beta_i]$ to denote the result of the quantized (exact) algorithms of [7] (e.g., maximum NSE^2 for MinRelVar) for the error subtree rooted at coefficient c_i assuming a space budget of β_i.

3. Our Approximation Scheme

In this section, we present our efficient approximation scheme, termed ϵ-ApproxRV, for constructing probabilistic wavelet synopses over large data sets. Our ϵ-ApproxRV is a guaranteed $(1+\epsilon)$ approximation algorithm for the MinRelVar scheme of Garofalakis and Gibbons [7]; that is, it focuses on minimizing the maximum NSE^2 in the data reconstruction. Our techniques can easily be extended to handle other error metrics, such as the maximum normalized bias employed by MinRelBias [7]. Our presentation here focuses primarily on the case of one-dimensional Haar wavelets – the details of the extension to multiple dimensions can be found in the full paper[4].

3.1. The One-Dimensional ϵ-ApproxRV Algorithm

Consider the error-tree structure for a one-dimensional Haar wavelet decomposition, and let R denote the *maximum absolute normalized value* of any coefficient in the tree, defined as

$$R = \max_i \frac{|c_i|}{\max\{d_{\min}(i), \mathsf{s}\}},$$

where, as previously, $d_{\min}(i)$ denotes the minimum absolute data value in the subtree of node i. (Typically, e.g., for frequency-count vectors, the denominator in the above expression is > 1, which implies that R is in the order of the maximum absolute coefficient value.) Our ϵ-ApproxRV algorithm runs in $O(N\mathsf{q} \log \mathsf{q} \min\{\frac{\log N \log R}{\epsilon}, B\mathsf{q}\})$ time and computes an approximate solution for any synopsis space budget $B \leq N$. Note that, for large synopsis sizes ($B = \Theta(N)$), the corresponding time complexity of the exact MinRelVar algorithm is significantly higher: $O(N^2\mathsf{q}^2 \log(N\mathsf{q}))$ [7]. Of course, our ϵ-ApproxRV algorithm is actually faster than the MinRelVar algorithm even for very small synopsis sizes ($B = o(\log N)$) and when the optimal solution is sought. Again, the key idea in our ϵ-ApproxRV algorithm is to speed up the DP search by making it much "sparser"[3] – in a nutshell, our approximate "sparse" DP algorithm will only search over a few possible space allotments for each error subtree, which are carefully chosen to guarantee a maximum deviation of $(1+\epsilon)$ from the optimal solution. Our ϵ-ApproxRV algorithm proceeds in a bottom-up fashion over the input error tree – to simplify the exposition in this section, we assume that levels in the error tree are numbered bottom-up, with leaf-node coefficients at level 0 and the root (overall average) at level $\log N - 1$.

The Sparse DP Approximation Scheme. Fix a quantization parameter q, and let $M_{\mathsf{q}}[v, b]$ denote the approximate maximum squared NSE (NSE^2) computed by ϵ-ApproxRV for any data value in the error subtree rooted at node v. As earlier, $M^*_{\mathsf{q}}[v, b]$ is the corresponding optimal NSE^2 value computed by MinRelVar. Note that, for any node v, the $M^*_{\mathsf{q}}[v, b]$ values are clearly *monotonically decreasing* in b; that is, $M^*_{\mathsf{q}}[v, x] \leq M^*_{\mathsf{q}}[v, y]$ for $x > y$ [7].

For the base case, consider a leaf-node coefficient v (at level 0) – clearly, in this case

$$M^*_{\mathsf{q}}[v, b] = \frac{\text{Var}(c_v, \min\{1, b\})}{\min\{\text{Norm}(2v), \text{Norm}(2v+1)\}}$$

i.e., the maximum normalized variance of the corresponding random variable with a success probability of b ($b \in \{0, \frac{1}{\mathsf{q}}, \frac{2}{\mathsf{q}}, \ldots, 1\}$ – any $b \geq 1$ obviously results in zero normalized variance). It is easy to see that all possible values for $M^*_{\mathsf{q}}[v, b]$, for any value of b, can be computed in time $O(\mathsf{q})$, where q is the designated quantization parameter. Out of these $O(\mathsf{q})$ variance values and possible allotments to c_v, our ϵ-ApproxRV algorithm picks a subset of allotments $b_1 > \ldots > b_h$ such that: (1) for each allotment $x \in [b_i, b_{i-1})$ we have $M^*_{\mathsf{q}}[v, b_i] \leq (1+\epsilon) M^*_{\mathsf{q}}[v, x]$; and, (2) b_1 through b_h cover the entire possible range of space allotments to c_v, i.e., $b_1 = 1$ and $b_h = 0$. This can obviously be done in $O(\mathsf{q})$ time by simply going over all M^*_{q} values and selecting b_{i+1} as the first allotment $\leq b_i$ such that $M^*_{\mathsf{q}}[v, b_{i+1}] > (1+\epsilon) M^*_{\mathsf{q}}[v, b_i]$. Since the maximum normalized variance for a coefficient value c_v is at most (see Section 2) $\frac{c_v^2}{\min\{\text{Norm}(2v), \text{Norm}(2v+1)\}} \leq R^2$, is easy to see that the number h of allotment "breakpoints" selected in this fashion is at most $O(\log_{1+\epsilon} R^2) = O(\frac{\log R}{\log(1+\epsilon)}) \approx O(\frac{\log R}{\epsilon})$ (for small values of $\epsilon < 1$). The approximate error values

3 Guha et al. [10] also discuss sparse DP algorithms in an entirely different context, namely building approximate V-optimal histograms over linear, time-series data; in contrast, our solution focuses on Haar wavelets and works over the hierarchical error-tree structure of the wavelet decomposition.

determined by our ϵ-ApproxRV algorithm for coefficient c_v are defined only by these h breakpoints $b_1, \ldots, b_h$ – specifically, $M_{\mathsf{q}}[v, b_i] = M^*_{\mathsf{q}}[v, b_i] = \frac{\text{Var}(c_v, b_i)}{\min\{\text{Norm}(2v), \text{Norm}(2v+1)\}}$ for $i = 1, \ldots, h$, and for any other possible allotment $x \in [b_i, b_{i-1})$, we define $M_{\mathsf{q}}[v, x] = M_{\mathsf{q}}[v, b_i]$. Thus, it is easy to see that, by construction, the approximation error of our ϵ-ApproxRV algorithm is bounded by a factor of $(1 + \epsilon)$ at leaf coefficients (at level 0); in other words, all dropped allotments are "covered" by a logarithmic number of breakpoints to within a (1 + ϵ) factor.

Now, proceeding inductively, consider an internal error-tree node v at level j, with children u and w (at level $j - 1$), and assume that the subtree rooted at u (w) has determined a collection of l_u (resp., l_w) error-function breakpoints $a_1 > \ldots > a_{l_u}$ (resp., $b_1 > \ldots > b_{l_w}$), and corresponding approximate NSE^2 values $M_{\mathsf{q}}[]$, that cover the range of allotments to each subtree and such that, for each $x \in [a_i, a_{i-1})$ ($i = 2, \ldots, l_u$), we have $M_{\mathsf{q}}[u, a_i] \leq (1+\epsilon)^j M^*_{\mathsf{q}}[u, x]$ (and similarly for w). Our ϵ-ApproxRV algorithm computes the allotment breakpoints and approximate error values $M_{\mathsf{q}}[]$ at the parent node v by iterating over all possible space allotments to node v and the breakpoints determined by the u and w subtrees (rather than all possible allotments to child subtrees), and retaining the minimum M_{q} values for each total allotment. The following lemma shows that, for each fixed space allotment to the coefficient at node v, it actually suffices to look at only $l_u + l_w$ combinations (a_i, b_k) for the subtree allotments rather than all possible $l_u \cdot l_w$ combinations.

Lemma 1: *When minimizing the maximum (approximate)* NSE^2 *error at node v, for any fixed space allotment to node v, it suffices to consider only $l_u + l_w$ combinations of allotments (a_i, b_k) to the child subtrees rooted at u, w.* ∎

Proof: Assume a fixed space allotment to the coefficient at node v, and let $leftVar$ ($rightVar$) denote the variance of node v (for the given allotment) divided by the normalization factor of its left (resp., right) subtree. Let L_u denote the sorted list of approximate NSE^2 values $M'_{\mathsf{q}}[u, a_i] = M_{\mathsf{q}}[u, a_i] + leftVar$, i.e., $M_{\mathsf{q}}[u, a_1] + leftVar < \ldots < M_{\mathsf{q}}[u, a_{l_u}] + leftVar$, with L_w defined similarly using the $rightVar$ quantity and the $M_{\mathsf{q}}[w, b_k]$ entries. Let $L = \text{merge}(L_u, L_w)$, i.e., $M'_{\mathsf{q}}[y_1] \leq \ldots \leq M'_{\mathsf{q}}[y_{l_u+l_w}]$, where $y_i \in \{(u, a_k) : k = 1, \ldots, l_u\} \cup \{(w, b_k) : k = 1, \ldots, l_w\}$. Now assume that a_i space is allocated to the u-subtree of v. Then, it is easy to see that, when considering the allotment to the w-subtree, out of all the b-values that lie to the left of a_i in L we really only need to consider the rightmost b-value, say b_k – the reason of course is that lower values of $M'_{\mathsf{q}}[w, b]$ (i.e., allotments $b > b_k$) result in configurations that use more total space without improving the error at v (since that is dominated by the u-subtree). These configurations are clearly useless in our error-minimization procedure. For the b-values to the right of a_i in L, a similar argument again applies: when a value b_k is assumed, only the closest a-value to its left in L needs to be considered. ∎

Thus, our approximate error-minimization procedure at v only needs to consider, for each fixed space allotment $s = 0, 1/\mathsf{q}, \ldots, 1$ to node v, $l_u + l_w$ breakpoint combinations (a_i, b_k) for the u and w subtrees, which can be determined easily in $O(l_u + l_w)$ time based on the proof of Lemma 1. Let $\mathcal{S}(s)$ denote the list of the obtained $l_u + l_w$ (a_i, b_k) combinations for each space allotment s to node v. The sorted list of approximate error values at node v can be computed in $O(\mathsf{q}(l_u + l_w) \log \mathsf{q})$ time by merging these lists using a heap structure or, alternatively, pairwise merging them in $\log \mathsf{q}$ steps. Thus, an initial list of $O(\mathsf{q}(l_u + l_w))$ breakpoints for the v subtree is determined based on the "useful" space-allocation configurations found through the above lemma – clearly, configurations that give the same (or, larger) NSE^2 values for the same (or, larger) amount of total space are useless and should be discarded. In other words, we define the initial set of space-allotment breakpoints for the v subtree as $\mathcal{C} = \{c = a_i + b_k + s : M_{\mathsf{q}}[v, a_i + b_k + s] \leq M_{\mathsf{q}}[v, a + b + t]$ for all $a + b + t \leq a_i + b_k + s\}$. (Useless configurations and configurations with space larger than B can easily be discarded in the merging pass for the $O(\mathsf{q})$ sub-lists described above.) It is easy to verify that, based on our inductive assumption, this set of breakpoints $\mathcal{C}$ covers the entire range of possible allotments for the v subtree; furthermore, the following lemma shows that it also preserves the approximation properties guaranteed by the individual subtrees.

Lemma 2: *Let $s_1 > \ldots > s_h$ denote the sorted list of space-allotment breakpoints $\mathcal{C}$ for the v subtree, computed as described above, and let $x \in [s_i, s_{i-1})$ for any i. Then, $M_{\mathsf{q}}[v, s_i] \leq (1 + \epsilon)^j M^*_{\mathsf{q}}[v, x]$, where j denotes the level of node v.* ∎

Proof: Assume that the optimal error value $M^*_{\mathsf{q}}[v, x]$ is obtained through the allotment configuration (y_u, y_w, s), that is:

$$M^*_{\mathsf{q}}[v, x] = \max \begin{cases} \frac{\text{Var}(c_v, s)}{\text{Norm}(u)} + M^*_{\mathsf{q}}[u, y_u] \\ \frac{\text{Var}(c_v, s)}{\text{Norm}(w)} + M^*_{\mathsf{q}}[w, y_w] \end{cases}$$

where, of course, $x \geq y_u + y_w + s$. Since the breakpoints for the u and w subtrees cover all possible allotments, let $y_u \in [a_i, a_{i-1})$ and $y_w \in [b_k, b_{k-1})$. By our inductive hypothesis, it is easy to see that the configuration (a_i, b_k, s) (which is obviously examined by the ϵ-ApproxRV algorithm) will give $M_{\mathsf{q}}[v, a_i + b_k + s] \leq (1+\epsilon)^j M^*_{\mathsf{q}}[v, x]$ and, clearly, $a_i + b_k + s \leq s_i \leq x$. Since $M_{\mathsf{q}}[v, s_i] \leq M_{\mathsf{q}}[v, a_i + b_k + s]$, the result follows. ∎

A potential problem with our approximation scheme, as described so far, is that the list of space-allotment breakpoints $\mathcal{C}$ would appear to grow exponentially as the DP moves up the error-tree levels. (So, starting with r breakpoints at the leaf nodes, we get $O(\mathsf{q}^j 2^j r)$ breakpoints at level j of the tree.) However, not all s_i's in $\mathcal{C}$ are necessary – we can actually "trim" $\mathcal{C}$ to a small number of breakpoints, while incurring an additional $(1+\epsilon)$ worst-case factor degradation on our approximation error. We perform this trimming process at every node of the error tree (except for the final, root node). More specifically, assume a chain of computed breakpoints $s_{i-k} > s_{i-k+1} > \ldots > s_i$ such that, for each $l = i-k, \ldots, i-1$ we have $M_{\mathsf{q}}[v, s_i] \leq (1+\epsilon) M_{\mathsf{q}}[v, s_l]$. Then, clearly, $M_{\mathsf{q}}[v, s_i]$ can "cover" all the points that are covered by $s_{i-k}, \ldots, s_{i-1}$ at an additional $(1+\epsilon)$ degradation, since, for any $l = i-k, \ldots, i-1$:

$$\begin{aligned} M_{\mathsf{q}}[v, s_i] &\leq (1+\epsilon) M_{\mathsf{q}}[v, s_l] \\ &\leq (1+\epsilon)^{j+1} M^*_{\mathsf{q}}[v, x] \quad \forall x \in [s_l, s_{l-1}). \end{aligned}$$

Thus, in this situation, the allotment points $s_{i-k}, \ldots, s_{i-1}$ can be eliminated and s_i can cover their ranges to within a $(1+\epsilon)^{j+1}$ factor. Now, note that the maximum value of the overall NSE^2 value at level j (and, thus, the range of values for the $M_{\mathsf{q}}[]$ array) is certainly upper-bounded by $(j+1)R^2$. This means that the total number of breakpoints obtained in the manner described above is at most $\log_{1+\epsilon}((j+1)R^2) \approx O(\frac{\log(j+1)+\log R}{\epsilon})$, which is an upper bound for the size of our breakpoint-list constructed at level j of the error tree. Thus, with an overall computational effort of:

$$\begin{aligned} &\sum_{j=0}^{\log N - 1} O(\frac{N}{2^{j+1}} \frac{\mathsf{q} \log \mathsf{q} (\log(j+1) + \log R)}{\epsilon}) \\ &\leq O(\frac{N\mathsf{q} \log \mathsf{q} \log R}{\epsilon} \sum_{j=0}^{\log N - 1} \frac{1}{2^{j+1}}) + O(\frac{N\mathsf{q} \log \mathsf{q}}{\epsilon} \sum_{j=0}^{\log N - 1} \frac{j+1}{2^{j+1}}) \\ &= O(\frac{N\mathsf{q} \log \mathsf{q} \log R}{\epsilon}), \end{aligned}$$

we get a (collection of) approximate solutions at the root node of the error tree that are guaranteed to cover the optimal MinRelVar solutions to within a $(1+\epsilon)^{\log N}$ factor. Then, it is easy to see that, setting $\epsilon' = \epsilon / \log N$, we get a guaranteed $(1+\epsilon)$ approximation in time $O(\frac{N\mathsf{q} \log \mathsf{q} \log N \log R}{\epsilon})$. Note that, to find the approximate solution for any specific choice of the allotment space B, we simply start out at the root and re-trace the steps of the algorithm for the largest root breakpoint s_i that is $\leq B$; to do that, we just need to keep track of the (a_i, b_k, s) configuration that generated each of the breakpoints at each error tree node and proceed recursively down the tree. It is easy to verify that the overall space required by the ϵ-ApproxRV algorithm is

$$\sum_{j=0}^{\log N - 1} O(\frac{N}{2^{j+1}} \frac{\log(j+1) + \log R}{\epsilon'}) = O(\frac{N \log N \log R}{\epsilon}),$$

while the working set size (maximum amount of memory-resident data) is only $O(\frac{\mathsf{q}(\log\log N + \log R)}{\epsilon'}) = O(\frac{\mathsf{q} \log N (\log\log N + \log R)}{\epsilon})$. To see this, note that ϵ-ApproxRV works in a bottom-up fashion and, when computing the breakpoint-list for a given node v, we only need access to: (1) the $l_u + l_w$ breakpoints of its child nodes in the error tree; and, (2) the $O(\mathsf{q}(l_u + l_w))$ initial breakpoints for node v that are computed just before the trimming process. Thus, the maximum working set will occur in the top-level of the error tree (level $\log N$-1), where $l_u + l_w = O(\frac{\log R + \log\log N}{\epsilon'})$. Finally, note that, given a space budget B, we cannot have more than $\mathsf{q}B$ different breakpoints at any error-tree node; in other words, the size of the breakpoint list at any level-j node is at most $O(\min\{\frac{\log(j+1)+\log R}{\epsilon}, \mathsf{q}B\})$. This easily implies that the overall space required by our ϵ-ApproxRV algorithm can never exceed the space requirements of MinRelVar (i.e., $O(N\mathsf{q}B)$). Similarly, the list of (at most $\mathsf{q}B$) breakpoints at each node can be computed in time $O(\mathsf{q} \times (\mathsf{q}B) \log \mathsf{q}) = O(B\mathsf{q}^2 \log \mathsf{q})$; thus, the overall running time complexity is also upper-bounded by $O(NB\mathsf{q}^2 \log \mathsf{q})$, giving an improvement over MinRelVar, even for very small values of B. The following theorem summarizes the results of our analysis for the one-dimensional ϵ-ApproxRV algorithm.

Theorem 3: *The ϵ-ApproxRV algorithm correctly computes a list of breakpoints at the root node such that, for any space budget $B \leq N$ and approximation factor ϵ, the estimated maximum NSE^2 value is within a factor of $(1+\epsilon)$ from the optimal MinRelVar solution. The overall ϵ-ApproxRV computation requires $O(N\mathsf{q} \log \mathsf{q} \min\{\frac{\log N \log R}{\epsilon}, B\mathsf{q}\})$ time and $O(N \min\{\frac{\log N \log R}{\epsilon}, \mathsf{q}B\})$ space, with a working set size of $O(\mathsf{q} \min\{\log N \frac{\log\log N + \log R}{\epsilon}, B\})$.* ∎

It is important to note that the above (worst-case) running-time and space complexities of our ϵ-ApproxRV algorithm are based on a pathological case where all the produced coefficients have an absolute normalized value of R. However, in most real-life data sets the wavelet decomposition process produces few coefficients of large magnitude, while the remaining coefficient values are significantly smaller. This, in turn, implies that, for most error-tree nodes, the maximum value of the overall NSE^2 value at level j will be significantly smaller than $(j+1)R^2$. This results not only in reduced space requirements for ϵ-ApproxRV (smaller breakpoint-lists stored at each node), but also in reduced running times (smaller breakpoint-lists scanned and merged). Our experimental results in Section 4 clearly validate our claims, with ϵ-ApproxRV demonstrating consistent and very significant gains over the exact MinRelVar scheme for a wide

range of input parameters and data sets.

Optimizations and Extensions. For very large data sets, it is possible that the breakpoint-lists produced by our ϵ-ApproxRV algorithm may not fit in main memory, resulting in substantial I/O during the algorithm's execution. In the full paper[4], we propose a simple optimization to address this concern. Briefly, the key idea is to compute the breakpoint-lists for error-tree nodes in one pass using a postorder traversal, with a working set size of $O(\min\{\frac{(\log N)^2(\log R+\log\log N)}{\epsilon}, \mathsf{q}B\log N\})$.

The full paper[4] also discusses in detail the extension of our ϵ-ApproxRV algorithm for multi-dimensional data sets. For the case of D-dimensional data, the running time of ϵ-ApproxRV becomes $O(\min\{\frac{N_z\mathsf{q}4^D\log M_{max}(\log\mathsf{q}+D)(D+\log R+\log\log M_{max})}{\epsilon}, N_z\mathsf{q}2^D\times(\mathsf{q}B+D2^D)\})$, where N_z denotes the number of nodes in the error tree that contain at least one non-zero coefficient, and M_{max} is the maximum domain size among all dimensions. The corresponding space requirements are $O(\min\{\frac{4^D N_z\log M_{max}(D+\log R+\log\log M_{max})}{\epsilon}, \mathsf{q}N_zB\})$. Note, of course, that in most real-life scenarios employing wavelet-based data reduction, the number of dimensions D is typically a small constant (e.g., 2–5) [2, 6, 7].

4. Experimental Study

In this section, we present an extensive experimental study of our proposed ϵ-ApproxRV algorithm for constructing probabilistic wavelet synopses over large data sets. The objective of this study is to evaluate both the scalability and the obtained accuracy of our proposed ϵ-ApproxRV algorithm when compared to the dynamic programming algorithm MinRelVar of [7] for a large variety of real-life and synthetic data sets. For the later DP solution, we used the significantly faster version of the algorithm that was very recently proposed in [7]. The main findings of our study for the ϵ-ApproxRV algorithm include:

• **Near Optimal Results.** The ϵ-ApproxRV algorithm consistently provides near-optimal solutions. Moreover, the actual deviation of the ϵ-ApproxRV solution from the optimal one is typically significantly smaller (usually by a factor larger than 5) than the specified ϵ value.

• **Significantly Faster Solution.** Our ϵ-ApproxRV algorithm provides a fast and scalable solution for constructing probabilistic synopses over large data sets. Compared to the MinRelVar algorithm of [7], the running time of the ϵ-ApproxRV algorithm is often more than an order of magnitude (and in some times more than two orders of magnitude) smaller, while at the same time providing highly-accurate answers. In fact, the ϵ-ApproxRV algorithm is significantly faster even for cases when the optimal solution is required ($\epsilon = 0$).

4.1. Testbed and Methodology

Techniques and Parameter Settings. Our experimental study compares the ϵ-ApproxRV and MinRelVar algorithms for constructing probabilistic wavelet synopses. Both algorithms utilize the quantization parameter q, which is assigned a value of 10, as suggested in [7], in our experiments. Larger values of this quantization parameter improved the running time performance of the ϵ-ApproxRV algorithm when compared to the MinRelVar algorithm, as expected by the running time complexities of the two algorithms. Finally, the sanity bound of each data set is set to its 5%-quantile data value.

Data Sets. We experiment with several one-dimensional synthetic and real-life data set. Due to space constraints we only present here the results for the real-life data sets (the performance of the algorithms in the synthetic data sets is qualitatively similar). The `Weather` data set contains meteorological measurements obtained by a station at the university of Washington (`www-k12.atmos.washington.edu/k12/grayskies`). This is a one-dimensional data set for which we extracted the following 6 measured quantities: wind speed, wind peak, solar irradiance, relative humidity, air temperature and dew-point temperature, and present here the results for the wind speed (AirSpeed) and the air temperature (AirTemp), which represent a noisy and a smooth signal, correspondingly. The `Phone` data set includes the total number of long distance calls per minute originating from several states in USA. We here present the results for the states of New York (NY) and Indiana (IN), with NY having large numbers of calls per minute and IN being a state with significantly fewer calls. The presented results for each real-life data set are also indicative of the results for the other measured quantities in these data sets.

Approximation Error Metrics. To compare the accuracy of the studied algorithms we focus on the maximum relative error of the approximation, since it can provide guaranteed error-bounds for the reconstruction of any individual data value. Since the objective function that both studied algorithms try to minimize is the maximum NSE^2 of any data value, for a more direct and clear comparison we present the results for this metric and for both algorithms. The results for the maximum relative error are qualitatively similar to the presented ones.

4.2. Experimental Results

Sensitivity to ϵ. We now evaluate the accuracy and running time of the ϵ-ApproxRV algorithm in comparison to the MinRelVar algorithm, using the real-life data sets. In Figures 2 and 3 we plot the running times for the two algorithms and for the two data sets, correspondingly, as we vary the value

of ϵ from 0 to 0.3. We set the domain size for all data sets to 65536, and the synopsis space to 5% of the input size. The ϵ-ApproxRV algorithm is consistently faster than the MinRelVar algorithm in both real-life data sets, often by more than an order of magnitude, and is considerably faster even when the optimal solution is required ($\epsilon = 0$). Unlike the MinRelVar algorithm which may perform multiple lookups of each computed entry,[4] the ϵ-ApproxRV algorithm processes all node entries in a single pass, therefore resulting in significantly faster running times. We also observe in these figures that, with the increase of ϵ, the running time of ϵ-ApproxRV decreases, as the algorithm effectively prunes a larger number of breakpoints.

The corresponding NSE^2 values for both algorithms are presented in Figures 4 and 5. In order for the reader to be able to observe the difference in the accuracy of the two algorithms, in each figure we plot the ratio of the maximum NSE^2 values produced by the ϵ-ApproxRV algorithm to the corresponding results of the MinRelVar algorithm. The ϵ-ApproxRV algorithm, as expected, always provides solutions that are within the specified error factor ϵ from the optimal solution. It is interesting to note though that in all cases the produced solution is significantly closer to the optimal one (by more than a factor of 5), than the specified ϵ value. This is not surprising, as ϵ represents a worst-case error bound.

Sensitivity to the Domain Size. We now evaluate the accuracy and running time of the ϵ-ApproxRV algorithm in comparison to the MinRelVar algorithm, using the real-life data sets, when we vary the domain size of the data sets from 128 to 65536, and plot the resulting running times for the two algorithms in Figures 6 and 7. The synopsis space is set to 5% of the input size, while the value of ϵ is set to 0.10. Again, the ϵ-ApproxRV algorithm significantly outperforms the MinRelVar algorithm, with the savings in running time increasing rapidly as the domain size increases. For large domain sizes, the ϵ-ApproxRV algorithm is up to 23.8 times faster than the MinRelVar algorithm.

In Figures 8 and 9 we plot the corresponding ratios of the maximum NSE^2 values obtained by the two algorithms. Again, the ϵ-ApproxRV algorithm always produced solutions that are significantly closer to the optimal solution (less than 1.7% and 1.4% difference, correspondingly, for the two data sets), than the specified error factor ϵ.

Sensitivity to the Synopsis Space. In Figures 10 and 11 we present the running times for both algorithms and for the real-life data sets, as the synopsis space is varied from 1% to 30% of the size of the input. The domain size is set to 65536, while the value of ϵ is set to 0.10. The running time of the MinRelVar algorithm increases rapidly with the increase in the used synopsis space, while the corresponding running time of the ϵ-ApproxRV algorithm remains practically unaffected. For large synopsis spaces, the ϵ-ApproxRV algorithm is more than two orders of magnitude faster than the MinRelVar algorithm. However, the solutions obtained from the ϵ-ApproxRV algorithm are again very close to the optimal ones (less than 1.7% and 1.5% difference for the two data sets), as we can see in Figures 12 and 13.

4 In the MinRelVar algorithm, the optimal solution of allocating space B to any node v may be probed for any space allotment $\geq B$ to v's parent node in the error tree.

5. Conclusions

We have proposed a novel, fast approximation scheme for constructing probabilistic wavelet synopses over large data sets. Our proposed techniques employ a much "sparser" version of previously proposed Dynamic-Programming (DP) solutions, which restricts its search to a carefully chosen, logarithmically-small subset of *"breakpoints"* that cover the entire range of possible space allotments, while always ensuring a maximum relative degradation of $(1 + \epsilon)$ in the quality of the obtained solution. Our experimental evaluation has demonstrated that our approximation algorithm typically provides significantly tighter solutions than the maximum $(1 + \epsilon)$ error factor, while at the same time providing running times that are up to two orders of magnitude smaller than known exact DP solutions.

References

[1] S. Acharya, P.B. Gibbons, V. Poosala, and S. Ramaswamy. "Join Synopses for Approximate Query Answering". In *ACM SIGMOD*, 1999.

[2] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. "Approximate Query Processing Using Wavelets". In *VLDB*, 2000.

[3] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. "Approximate query processing using wavelets". *The VLDB Journal*, 10(2-3):199–223, September 2001.

[4] A. Deligiannakis, M. Garofalakis and N. Roussopoulos. "A Fast Approximation Scheme for Probabilistic Wavelet Synopses". Technical Report UMCP-CSD-CS TR #4643, 2005.

[5] A. Deligiannakis and N. Roussopoulos. "Extended Wavelets for Multiple Measures". In *ACM SIGMOD*, 2003.

[6] M. Garofalakis and P.B. Gibbons. "Approximate Query Processing: Taming the Terabytes". Tutorial in *VLDB*, 2001.

[7] M. Garofalakis and P.B. Gibbons. "Probabilistic Wavelet Synopses". *ACM Transactions on Database Systems*, 29(1), March 2004.

[8] M. Garofalakis and A. Kumar. "Deterministic Wavelet Thresholding for Maximum-Error Metrics". In *PODS*, 2004.

[9] A.C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M.J. Strauss. "Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries". In *VLDB*, 2001.

[10] S. Guha, N. Koudas and K. Shim. "Data-Streams and Histograms". In *STOC*, 2001.

[11] P.J. Haas and A.N. Swami. "Sequential Sampling Procedures for Query Size Estimation". In *ACM SIGMOD*, 1992.

[12] J.M. Hellerstein, P.J. Haas, and H.J. Wang. "Online Aggregation". In *ACM SIGMOD*, 1997.

[13] B. Jawerth and W. Sweldens. "An Overview of Wavelet Based Multiresolution Analyses". *SIAM Review*, 36(3):377–412, 1994.

[14] Y. Matias, J.S. Vitter, and M. Wang. "Wavelet-Based Histograms for Selectivity Estimation". In *ACM SIGMOD*, 1998.

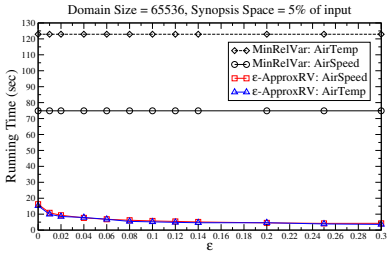


Figure 2: Running Time vs ϵ in Weather data

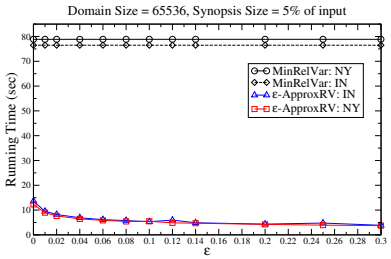


Figure 3: Running Time vs ϵ in Phone data

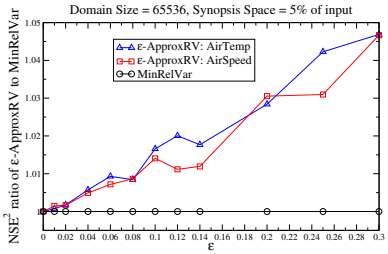


Figure 4: NSE^2 ratio vs ϵ in Weather data

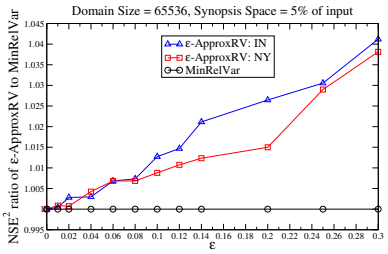


Figure 5: NSE^2 ratio vs ϵ in Phone data

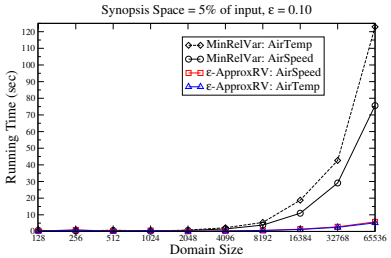


Figure 6: Running Time vs Domain Size in Weather data

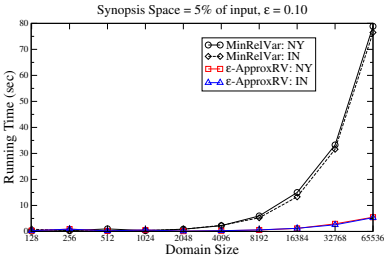


Figure 7: Running Time vs Domain Size in Phone data

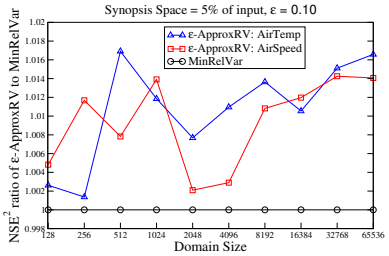


Figure 8: NSE^2 ratio vs Domain Size in Weather data

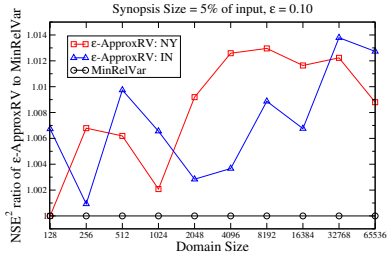


Figure 9: NSE^2 ratio vs Domain Size in Phone data

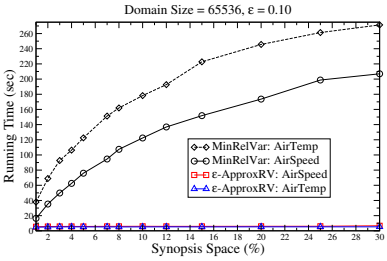


Figure 10: Running Time vs Space in Weather data

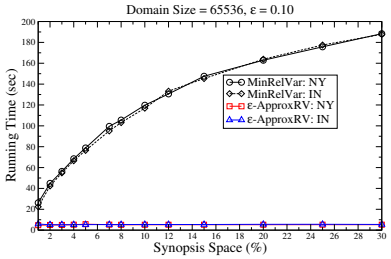


Figure 11: Running Time vs Space in Phone data

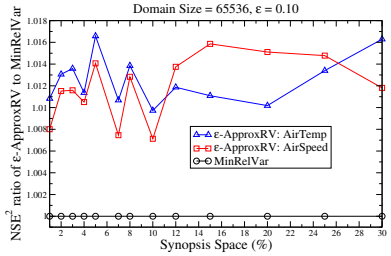


Figure 12: NSE^2 ratio vs Space in Weather data

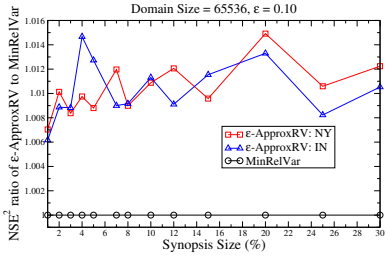


Figure 13: NSE^2 ratio vs Space in Phone data

[15] Y. Matias, J.S. Vitter, and M. Wang. "Dynamic Maintenance of Wavelet-Based Histograms". In *VLDB*, 2000.

[16] R. Motwani and P. Raghavan. "*Randomized Algorithms*". Cambridge University Press, 1995.

[17] R.R. Schmidt and C. Shahabi. "ProPolyne: A Fast Wavelet-Based Algorithm for Progressive Evaluation of Polynomial Range-Sum Queries". In *EDBT*, 2002.

[18] E.J. Stollnitz, T.D. DeRose, and D.H. Salesin. "*Wavelets for Computer Graphics – Theory and Applications*". Morgan Kaufmann Publishers, 1996.

[19] J.S. Vitter and M. Wang. "Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets". In *ACM SIGMOD*, 1999.

The "Best K" for Entropy-based Categorical Data Clustering

Keke Chen Ling Liu
College of Computing, Georgia Institute of Technology
{kekechen, lingliu}@cc.gatech.edu

Abstract

With the growing demand on cluster analysis for categorical data, a handful of categorical clustering algorithms have been developed. Surprisingly, to our knowledge, none has satisfactorily addressed the important problem for categorical clustering – how can we determine the best K number of clusters for a categorical dataset? Since categorical data does not have the inherent distance function as the similarity measure, traditional cluster validation techniques based on the geometry shape and density distribution cannot be applied to answer this question. In this paper, we investigate the entropy property of the categorical data and propose a BkPlot *method for determining a set of candidate "best Ks". This method is implemented with a hierarchical clustering algorithm* ACE. *The experimental results show that our approach can effectively identify the significant clustering structures.*

1 Introduction

Data clustering is an important method in data analysis. Clustering algorithms use the similarity measure to group the most similar items into clusters [23]. Clustering techniques for categorical data are very different from those for numerical data in terms of the definition of similarity measure. Most numerical clustering techniques use distance functions, for example, Euclidean distance, to define the similarity measure, while there is no inherent distance meaning between categorical values.

Traditionally, categorical data clustering is merged into numerical clustering through the data preprocessing stage [23], where numerical features are extracted/constructed from the categorical data, or the conceptual similarity between data records is defined based on the domain knowledge. However, meaningful numerical features or conceptual similarity are usually difficult to extract at the early stage of data analysis because we have little knowledge about the data. It has been widely recognized that clustering directly on the raw categorical data is important for many applications. Examples include environmental data analysis [29], market basket data analysis [1], DNA or protein sequence analysis [8], and network intrusion analysis [5]. Therefore, there are increasing interests in clustering categorical data recently [21, 19, 17, 18, 6, 15, 3, 25].

Cluster Validation Different clustering algorithms hardly generate the same clustering result for the same dataset, and we need the cluster validation methods to evaluate the quality of the clustering results [27, 22, 20]. Formally, there are two main issues in cluster validation: 1) how to evaluate the quality of different partition schemes generated by different clustering algorithms for certain dataset, given the fixed K number of clusters; 2) how to determine the best number of clusters (the "best K"), which indicates the inherent significant clustering structures of the dataset.

For numerical data, the clustering structure is usually validated by the geometry and density distribution of the clusters. When a distance function is given for the numerical data, it is natural to introduce the density-based methods [16, 4] into clustering. As a result, the distance functions and density concepts play the unique roles in validating the numerical clustering result. Various statistical cluster validation methods and visualization-based validation methods have been proposed for numerical data [22, 20, 12], all of which are based on the geometry and density property. The intuition behind the geometry and density distribution justifies the effectiveness of these cluster validation methods. A good example commonly seen in clustering literature is evaluating the clustering result of 2D experimental datasets by visualizing it – the clustering result is validated by checking how well the clustering result matches the geometry and density distribution of points through the cluster visualization.

While lack of the distance meaning for the categorical data, the techniques used in cluster validation for numerical data are not applicable for categorical data. Without reasonable numerical feature extraction/construction for a given categorical dataset, the general distance functions are usually inapplicable and unintuitive. As a result, no geometry/density-based validation method is appropriate in validating the clustering result for categorical data.

Entropy Based Similarity Instead of using distance function to measure the similarity between any pair of data records, similarity measures based on the "purity" of a set of records seem more intuitive for categorical data. As a well-defined and accepted concept, entropy [14] can be used to

formally measure the purity of partition. Originally from information theory, entropy has been applied in both pattern discovery [10] and numerical clustering [13]. Due to the lack of intuitive distance definition for categorical values, recently, there have been efforts in applying the entropy criterion in clustering categorical data [6, 25]. The initial results show that entropy criterion can be very effective in clustering categorical data. Li et al [25] also proved that the entropy criterion can be formally derived from the framework of probabilistic clustering models, which further supports that the entropy criterion is a meaningful and reliable similarity measure for categorical data.

In entropy-based categorical clustering, the quality of clustering result is naturally evaluated by the entropy criterion [6, 25], namely, the *expected entropy* for a partition. However, the other cluster validation problem – determining the "best K", has not been sufficiently addressed yet. In this paper, we present a novel method based on entropy to address this problem.

Our Approach We first develop an entropy-based categorical clustering algorithm "*ACE*"(Agglomerative Categorical clustering with Entropy criterion). The algorithm works in a bottom-up manner. Beginning with each individual record as a cluster, it merges the most similar pair of clusters in each step, where the similarity is evaluated with the *incremental entropy*. An agglomerative hierarchical clustering algorithm typically generates a clustering tree that contains the different clustering structures that have different K. We use these clustering structures to analyze the best K problem.

Based on the intuition behind the merging operation in ACE algorithm, we investigate the relation between the pairs of neighboring partition schemes (having K clusters and $K+1$ clusters, respectively). We use "*Entropy Characteristic Graph* (ECG) " to sketch the entropy property of the clustering structures, and use "*Best-K Plot* (BkPlot)", which is built on ECG, to identify the candidates of the best K. The initial experimental result shows that the proposed validation method, concretely, using the BkPlots generated by ACE to identify the best Ks, works effectively in finding the significant K(s) for categorical data clustering.

The rest of the paper is organized as follows. Section 2 sets down the notations and gives the definition of the traditional entropy-based clustering criterion. Section 3 presents the agglomerative hierarchical clustering algorithm ACE. Section 4 investigates the relation between the neighboring partitioning schemes with the entropy criterion, and proposes the validation method for identifying the best Ks. We present the experimental result in section 5 and review the related categorical clustering work in section 6. Finally, we conclude our work in section 7.

2 Notations and Definitions

We first give the notations used in this paper and then introduce the traditional entropy-based clustering criterion. Several basic properties about the entropy criterion will be presented later.

Consider that a dataset $\mathbb{S}$ with N records and d columns, is a sample set of the discrete random vector $X = (x_1, x_2, \ldots, x_d)$. For each component x_j, $1 \leqslant j \leqslant d$, x_j takes a value from the domain A_j. A_j is conceptually different from $A_k (k \neq j)$. There are a finite number of distinct categorical values in domain A_j and we denote the number of distinct values as $|A_j|$. Let $p(x_j = v)$, $v \in A_j$, represent the probability of $x_j = v$, we have the classical entropy definition [14] as follows.

$$H(X) = -\sum_{j=1}^{d} \sum_{v \in A_j} p(x_j = v) \log_2 p(x_j = v)$$

When $H(X)$ is estimated with the sample set $\mathbb{S}$, we define the estimated entropy as $\hat{H}(X) = H(X|\mathbb{S})$, i.e.

$$\hat{H}(X) = -\sum_{j=1}^{d} \sum_{v \in A_j} p(x_j = v|\mathbb{S}) \log_2 p(x_j = v|\mathbb{S})$$

Suppose the dataset $\mathbb{S}$ is partitioned into K clusters. Let $C^K = \{C_1, \ldots, C_K\}$ represent a partition, where C_k is a cluster and n_k represent the number of records in C_k. The classical entropy-based clustering criterion tries to find the optimal partition, C^K, which maximizes the following entropy criterion [9, 11, 25].

$$O(C^K) = \frac{1}{d}\left(\hat{H}(X) - \frac{1}{n}\sum_{k=1}^{K} n_k \hat{H}(C_k)\right)$$

Since $\hat{H}(X)$ is fixed for a given dataset $\mathbb{S}$, maximizing $O(C^K)$ is equivalent to minimize the item $\frac{1}{n}\sum_{k=1}^{K} n_k \hat{H}(C_k)$, which is named as the "expected entropy" of partition C^K. Let us notate it as $\bar{H}(C^K)$. For convenience, we also name $n_k \hat{H}(C_k)$ as the "weighted entropy" of cluster C_k.

Li et al [25] showed that the minimization of expected-entropy is equivalent to many important concepts in information theory, clustering, and classification, such as Kullback-Leibler Measure, Maximum Likelihood [24], Minimum Description Length [26], and dissimilarity coefficients [7]. Entropy criterion is especially good for categorical clustering due to the lack of intuitive definition of distance for categorical values. While entropy criterion can also be applied to numerical data [13], it is not the best choice since it cannot describe the cluster shapes and other numerical clustering features of the dataset.

3 ACE:Agglomerative Categorical clustering with Entropy criterion

In this section, we define the proposed similarity measure, *incremental entropy*, for any two clusters. With incremental entropy, we design the algorithm ACE. ACE and its working mechanism is the tool used to explore the significant clustering structures in the next section.

3.1 Incremental Entropy

In this section, we investigate the mergence of any two clusters to explore the similarity between the two clusters. Intuitively, merging the two clusters that are similar in the inherent structure will not increase the disorderliness (expected-entropy) of the partition, while merging dissimilar ones will inevitably bring larger disorderliness. We observed that this increase of expected entropy has some correlation with the similarity between clusters. Therefore, it is necessary to formally explore the entropy property of merging clusters. Let $C_p \cup C_q$ represent the mergence of two clusters C_p and C_q in some partition scheme, and C_p and C_q have n_p and n_q members, respectively. By the definition of expected entropy, the difference between $\hat{H}(K)$ and $\hat{H}(K+1)$ is only the difference between the weighted entropies, $(n_p+n_q)\hat{H}(C_p \cup C_q)$ and $n_p\hat{H}(C_p)+n_q\hat{H}(C_q)$. We have the following relation for the weighted entropies.

Proposition 1. $(n_p+n_q)\hat{H}(C_p \cup C_q) \geqslant n_p\hat{H}(C_p) + n_q\hat{H}(C_q)$

PROOF. The about relation can be expanded as follows.

$$-\sum_{j=1}^{d}\sum_{v\in A_j}(n_p+n_q)p(x_j=v|C_p\cup C_q)\cdot \log_2 p(x_j=v|C_p\cup C_q) \geqslant$$
$$-\sum_{j=1}^{d}\sum_{v\in A_j} n_p p(x_j=v|C_p)\log_2 p(x_j=v|C_p) -$$
$$-\sum_{j=1}^{d}\sum_{v\in A_j} n_q p(x_j=v|C_q)\log_2 p(x_j=v|C_q) \quad (1)$$

It is equivalent to check if the following relation is satisfied for each value v in each $domain(A_j)$.

$$n_p p(x_j=v|C_p)\log_2 p(x_j=v|C_p) + n_q p(x_j=v|C_q)\log_2 p(x_j=v|C_q) \geqslant (n_p+n_q)p(x_j=v|C_p\cup C_q)\cdot \log_2 p(x_j=v|C_p\cup C_q) \quad (2)$$

Without loss of generality, suppose C_p having x items and C_q having y items in value v at j-th attribute. The formula 2 can be transformed to $x\log_2\frac{x}{n_p}+y\log_2\frac{y}{n_q} \geqslant (x+y)\log_2\frac{x+y}{n_p+n_q}$. Since x, y, n_p, n_q are positive integers, let $x = s\cdot y$ and $n_p = r\cdot n_q, (s, r > 0)$, and then we can eliminate $\log_2$ to get a simpler form: $\frac{r^s}{(1+r)^{s+1}} \leqslant \frac{s^s}{(1+s)^{1+s}}$. It is easy to prove that $\frac{s^s}{(1+s)^{1+s}}$ is the maximum value of the function $f(r) = \frac{r^s}{(1+r)^{s+1}}(r, s > 0)$. Therefore, formula (2) is true, thus (1) is true and Proposition 1 is proved. □

Let $I_m(C_p, C_q) = (n_p+n_q)\hat{H}(C_p\cup C_q) - (n_p\hat{H}(C_p)+n_q\hat{H}(C_q))$ be the "*incremental entropy*" by merging the clusters C_p and C_q. Note that $I_m(C_p, C_q) = 0$ most likely suggests that the two clusters have the *identical structure* – for every categorical value v_i in every attribute x_j, $1 \leqslant i \leqslant |A_j|$, $1 \leqslant j \leqslant d$, we have $p(x_j = v_i|C_p) = p(x_j = v_i|C_q)$. A simple example in table 1 demonstrates the identical structure.

ds1	ds2
1 1 0 1	1 1 0 1
1 1 0 1	0 0 1 1
0 0 1 1	
0 0 1 1	

Table 1. Identical structure

Incremental entropy brings the important heuristic about the dissimilarity between any two clusters, i.e., when the two clusters are similar in structure, merging them will not bring large disorderliness into the partition, thus, $I_m(C_p, C_q)$ will be small; when the two clusters are very different, merging them will bring great disorderliness, thus, $I_m(C_p, C_q)$ will be large. Therefore, incremental entropy intuitively serves as the similarity measure between any two clusters.

3.2 ACE Algorithm

While the traditional hierarchical algorithms for numerical clustering needs to explicitly define the inter-cluster similarity with "single-link", "multi-link" or "complete-link" methods [22]. Incremental entropy is a natural inter-cluster similarity measure, ready for constructing a hierarchical clustering algorithm. Having incremental entropy as the measure of inter-cluster similarity, we developed the following entropy-based agglomerative hierarchical clustering algorithm – (ACE).

ACE algorithm is a bottom-up process to construct a clustering tree. It begins with the scenario where each record is a cluster. Then, an iterative process is followed – in each step, the algorithm finds a pair of clusters C_p and C_q that are the most similar, i.e. $I_m(C_p, C_q)$ is minimum among all possible pair of clusters. We use $I_m^{(K)}$ to denote the I_m value in forming the K-cluster partition from the K+1-cluster partition.

Maintaining the minimum incremental entropy in each step is the most costly part. In order to efficiently implement the ACE algorithm, we maintain three main data structures: *summary table* for conveniently counting the occurrences of values, I_m-*table* for bookkeeping $I_m(C_p, C_q)$ of any pair of clusters C_p and C_q, and a I_m *heap* for maintaining the minimum I_m value in each step.

Summary table is used to maintain the fast calculation of cluster entropy $\hat{H}(C_k)$ and each cluster has one summary table (Figure 1). Since computing cluster entropy is based on counting the occurrences of categorical values in each column, we need the summary table to keep the counters for each cluster. If the average column cardinality is m, a summary table keeps dm counters. Such a summary table

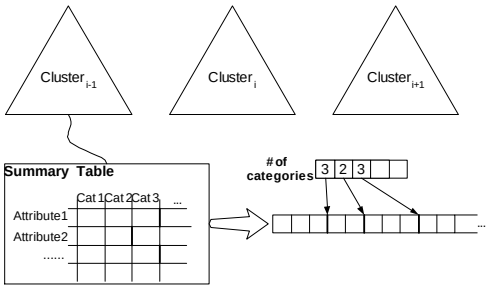


Figure 1. Summary table and physical structure

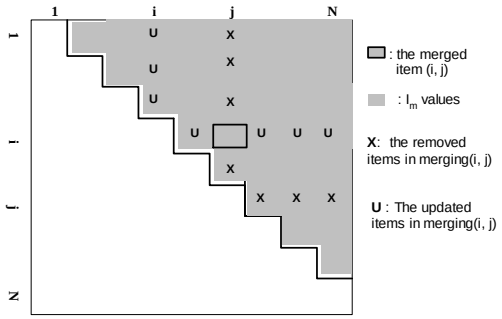


Figure 2. Operation schedule after a merging operation

enables fast merging operation – when merging two clusters, the two summary tables are added up to form a new summary table for the new cluster.

We use I_m-table to keep track of the incremental entropy between any pair of clusters, which is then used to maintain the minimum-I_m in each round of merging. The I_m-table is a symmetric table (thus, only a half of entries are used in practice), where the cell (i, j) keeps the value of $I_m(C_i, C_j)$ Figure 2.

I_m heap is used to keep track of the globally minimum incremental entropy. We define the most similar cluster of cluster u as $u.similar = \arg\min_v\{I_m(u, v), v \neq u\}$. Let $u.I_m$ represent the corresponding incremental entropy of merging u and $u.similar$, we define $< u, u.I_m, u.similar >$ as the *feature vector* of cluster u. The feature vectors are inserted into the heap, sorted by $u.I_m$, for fast locating the most similar pair of clusters.

Algorithm 1 shows the sketch of the main procedure. When merging u and $u.similar$ happens, their summary tables are added up to form the new summary table. Consider u as the main cluster, i.e., $u.similar$ is merged to cluster u, we need to find the new $u.similar$ and insert the new feature vector $< u, u.I_m, u.similar >$ into the heap. Then, there comes the important procedure for updating the bookkeeping information after merging operation. Let v denote the old $u.similar$. The bookkeeping information for v is discarded and any entries in I_m-table related to u or v should be updated. For any cluster w, if the $w.similar$ is changed due to the update of I_m-table, its location at the heap needs to be updated too. The detailed update algorithm is described in Algorithm 2 and demonstrated by Figure 2.

3.3 Complexity of ACE

Updating the I_m-table is the most costly part, consisting several incremental-entropy calculations. Each incremental-entropy calculation involves the summation of the two summary tables and computing the weighted entropy with the new summary table. The cost of computing weighted entropy is $O(dm)$, when an auxiliary array in length of N is used to buffer the $\log_2$ values as the following

Algorithm 1 ACE.main()

```
T_s[] ← initialize summary tables
T_{I_m} ← initialize I_m table
h ← heap
for Each record u do
  h.push (< u, u.I_m, u.similar >)
end for
while not empty(h) do
  < u, u.I_m, u.similar > ← h.top()
  T_s[u] ← T_s[u] + T_s[u.similar]
  update < u, u.I_m, u.similar >
  h.push (< u, u.I_m, u.similar >)
  updating_after_merging() //Algorithm 2
end while
```

equation shows.

$$
\begin{aligned}
& n_p \hat{H}(C_p) \\
= & -\sum_{j=1}^{d} \sum_{\substack{v_{jk} \in A_j \\ c_{jk} = freq(v_{jk}) | C_p}} c_{jk} (\log_2 c_{jk} - \log_2 n_p)
\end{aligned}
$$

The cost is dominated by updating I_m-table after each merging, which will totally need $O(N^2)$ incremental-entropy calculations in the worst case. Therefore, the overall time complexity is $O(dmN^2)$. The summary tables require $O(dmN)$ space, both the log_2 buffer and the heap costs $O(N)$ space, and I_m-table costs $O(N^2)$ space.

Algorithm 2 ACE.updating_after_merging()

```
C_i ← master cluster, C_j ← merged cluster
release T_s[C_j]
invalidate I_m table entries (C_j, *)
update I_m table entries (*, C_i) and (*, C_j)
for Each valid cluster u, if u.similar == C_i or C_j do
  update < u, u.I_m, u.similar >;
  relocate < u, u.I_m, u.similar > in h
end for
```

4 Exploring the Significant Clustering Structures

Traditionally, statistical validity indices based on geometry and density distribution are applied in clustering numerical data [20]. A typical index curve consists of the statistical index values for different K number of clusters. The Ks at the peaks, valleys, or distinguished "knees" on the index curve, are regarded as the candidates of the optimal number of clusters (the best K). Are there such index curves indicating the significant clustering structures for categorical data as well? The first thought might be investigating the curve of the expected entropy of the optimal partition of K clusters, notated as $\bar{H}_{opt}(C^K)$.

Our result shows that the curve of optimal expected-entropies is usually a smoothly decreasing curve without any distinguished peaks, valley, or knees (Figure 3). However, we find some special meaning behind the neighboring partition schemes (with K and $K+1$ clusters respectively). The differential of expected-entropy curve, which we name as "Entropy Characteristic Graph (ECG)" (Figure 4), has some substantial meaning indicating the significant clustering structures. An ECG shows that the similar partition schemes with different K are at the same "plateau". From plateau to plateau there are the critical points implying the significant change of clustering structure, which could be the candidates for the best Ks. These critical points are highlighted in the second-order differential of ECG, named "Best-K Plot (BkPlot)".

4.1 Property of Optimal Partition Schemes

In this section, we first give the Proposition 2 describing the relationship between the optimal expected-entropies with varying K, which is then used to introduce the "Entropy Characteristic Graph" and "BkPlot".

Since the significant clustering structures are the globally optimal selections, we begin with the investigation of optimal partitions with varying K. We describe the property of the optimal expected entropies as follows.

First of all, $\bar{H}_{opt}(C^K)$ is bounded. It was proved in [25] that $\bar{H}(C^K)$ is bounded by the maximum value $\hat{H}(X)$. We also have $\bar{H}(C^K) \geqslant 0$ as the entropy definition implies. The zero entropy of $\bar{H}(C^k)$ is reached at $k = N$, when each vector is a cluster.

Second, for any different number of clusters, K and L, $K < L$, we introduce the following property.

Proposition 2. *$\bar{H}_{opt}(C^K) \geqslant \bar{H}_{opt}(C^L)$, when $K < L$*

PROOF. Let some L-cluster partition C_0^L be formed by splitting the clusters in the optimal K-cluster partition. With Proposition 1, we have $\bar{H}_{opt}(C^K) \geqslant \bar{H}(C_0^L) \geqslant \bar{H}_{opt}(C^L)$ □

Proposition 2 shows that the optimal expected-entropy decreases with the increasing of K, which meets the intuition very well. It is hard to describe the curve with a formal function with varying K. However, as our experimental result shows, it is often a negative logarithm-like curve (Figure 3). The expected-entropy curve seems not help us to clearly identify the significant clustering structures. However, there is some important implication behind the expected-entropy curve when we consider the *similarity between the neighboring partitions*, where the neighboring partitions refer to the K-cluster partition and $K+1$-cluster partition.

4.2 Understanding the Similarity of Neighboring Partition Schemes

There are two aspects to capture the similarity of neighboring partition schemes. One is the increasing rate of entropy, defined as $I(K) = \bar{H}_{opt}(C^{K+1}) - \bar{H}_{opt}(C^K)$, which indicates how much the clustering structure is changed. The other aspect is the difference between $I(K)$ and $I(K+1)$, which indicates whether the consecutive changes to the clustering structure are similar. Since it is hard to describe the relation between the optimal partitions, we use the merging of clusters described in ACE algorithm to intuitively illustrate the two aspects of similarity. In the consecutive partition schemes generated by ACE, the increasing rate is equivalent to incremental entropy: $I(K) = \frac{1}{Nd} I_m^{(K)}$.

First, we consider the meaning of small increasing rate of entropy. As we discussed, merging identical clusters introduces zero increasing rate, which implies that the merging does not introduce any impurity to the clusters and the clustering structure is not changed. Similarly, small increasing rate implies small impurity, for which we consider the clustering structure is not significantly changed; and large increasing rates should introduce considerable impurity into the partitions and thus the clustering structure can be changed significantly. For large increasing rates, we need to further investigate the *relative changes* to determine if a globally significant clustering structure emerges, which is described as follows.

Consider $I(K)$ as the amount of impurity introduced from $K+1$-cluster scheme to K-cluster scheme. If $I(K) \approx I(K+1)$, i.e. K-cluster scheme introduces similar amount of impurity as K+1-cluster scheme does, we define that the clustering structure is not *relatively* changed from K+1-cluster scheme to K-cluster scheme. An conceptual demonstration of "similar mergence" in Figure 6 can help to understand the similarity of clustering structure at $I(K) \approx I(K+1)$. Here, we use icons to conceptually represent the categorical clusters. The shape and the size of an icon represent the structure and size of the cluster, respectively. Clusters in the identical or similar structure are preferred to be merged as the "identical structure" in section 3.1 shows, regardless of the cluster size. The four clusters ($C_1 \sim C_4$) in Figure 6 are very similar. They are selected in two consecutive merging operations. Thus, the changes to the resulting clustering structures are similar and not quite distinguishable from each other.

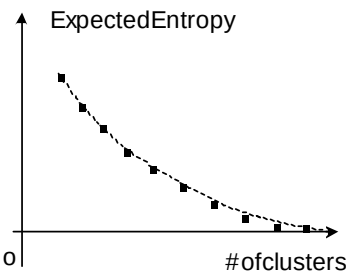


Figure 3. Sketch of expected entropy curve.

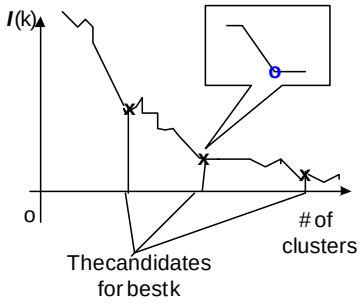


Figure 4. Sketch of ECG graph.

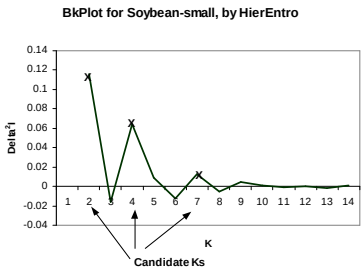


Figure 5. Finding the best k with BkPlot.

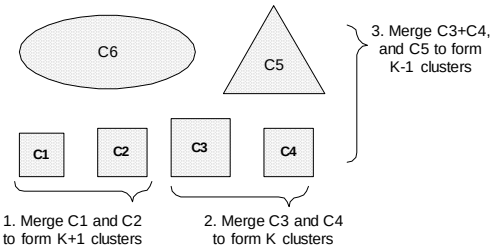


Figure 6. $I(K) \approx I(K+1)$, but $I(K-1) > I(K)$ significantly

However, the third merging operation, which merges $C_3 \cup C_4$ and C_5, might change the clustering structure greatly, and thus $I(K-1)$ can increase dramatically. This indicates that the second merging operation has resulted in a representative clustering structure for cluster analysis.

In practice, if a dataset has significant clustering structures, we can find a series of neighboring "stable" schemes, which have similar increasing rate of entropy, and we may also find the *critical points* where a series of "stable" schemes become "less stable" – the increasing rate changes dramatically (Figure 4). Each of such critical points indicates some significant change in clustering structure and distinguishes a set of "stable" schemes from another set. All of the critical points should be the candidates for the best Ks and could be interesting to cluster analysis.

We name the $I(K)$ plot as *Entropy Characteristic Graph (ECG)*. If a dataset has significant clustering structures, its ECG should be a curve with some distinguished "knees". An ECG curve showing no distinguished knees implies that the clustering structure is smoothly changed when K changes from N to 1, and thus clustering structures at all Ks have the same importance – in other words, there is no significant clustering structure.

The common way to mathematically identify such critical knees on a curve is to find the peaks/valleys at the second-order differential of the curve. Since an ECG consists of a set of discrete points, we define the second-order differential of ECG as $\delta^2 I(K) : \delta I(K) = I(K) - I(K+1)$ and $\delta^2 I(K) = \delta I(K-1) - \delta I(K)$ to make K aligned with the critical points. We can clearly identify the best Ks at the $\delta^2 I(K)$ plot, and thus name it as the "Best-k Plot (BkPlot)" (Figure 5).

4.3 Entropy Characteristic Graph Generated by ACE

ECGs generated by ACE have a special property. We use $I_m^{(K)}$ to denote the I_m value in forming K-cluster partition from $K+1$-cluster partition. Since $I(K) = \frac{1}{Nd} I_m^{(K)}$, it is equivalent to investigate the property of $I_m^{(K)}$. We will prove that $I_m^{(K)} \geqslant I_m^{(K+1)}$, so that the critical points always happen at the peaks of BkPlot.

Proposition 3. $I_m^{(K)} \geqslant I_m^{(K+1)}$

PROOF. Let $I_m(C_o, C_p, C_q)$ denote the incremental entropy in merging any three clusters. It is trivial to prove that the sequence of the three clusters does not matter in calculating the I_m and

$$I_m(C_o, C_p, C_q) \geqslant I_m(C_{(1)}, C_{(2)}) \tag{3}$$

where $C_{(1)}$ and $C_{(2)}$ are any two of the three clusters.

We maintain the ascending list of I_m for each merging operation in ACE algorithm. Suppose that the two clusters C_p and C_q are selected to merge and thus form the $K+1$-cluster scheme. We have $I_m^{(K+1)} = I_m(C_p, C_q)$. After the merge operation, the incremental entropy between the pairs of any cluster C_o, $o \neq p, q$, and the new cluster $C_p \cup C_q$, should be updated to $I_m(C_o, C_p, C_q)$. Since $I_m(C_p, C_q)$ is the minimum value at the stage $K+1$ and the relation (3) shows the updates to I_m table only increase the values, the selected I_m value for stage K will definitely be greater or equal to that of stage $K+1$, i.e. $I_m^{(K)} \geqslant I_m^{(K+1)}$. □

The BkPlots of such ECGs ($I(K) \geqslant I(K+1)$) always exhibit the critical Ks at peaks. This could reduce the number of possible noisy Ks and help the users to clearly identify the best K. We will demonstrate that the BkPlots generated by ACE are the most robust and efficient ones, compared to those generated by other algorithms.

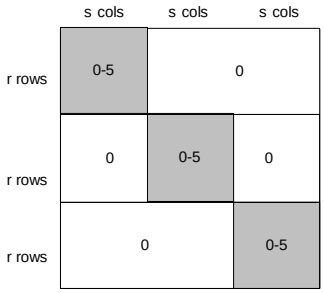


Figure 7. Synthetic Data DS1

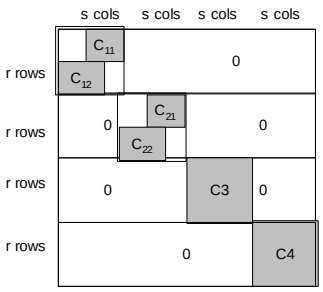


Figure 8. Synthetic Data DS2

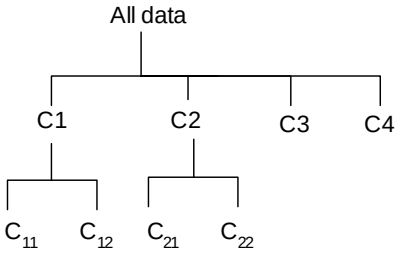


Figure 9. The significant clustering structures in DS2

5 Experimental Results

The goal of the experiments is twofold. 1)We want to show that BkPlot can be used to find the critical Ks. 2) We want to show that the BkPlots generated by ACE are the most robust and efficient, compared to those generated by the other two popular entropy-based clustering algorithms: Monte-Carlo method (MC) [25] and Coolcat [6].

5.1 Datasets

We construct two types of synthetic datasets with the following way, so that the clustering structure can be intuitively identified and manually labeled before running the experiments. The first type of datasets has a one-layer clustering structure (Figure 7) with 30 attributes and 1000 rows. It has three clusters in the same size (about 333 rows for each). Each cluster has random categorical values selected from {'0','1','2','3','4', '5'} in a distinct set of attributes, while the rest attributes are set to '0'. The second type of datasets has a two-layer clustering structure also with 30 attributes and 1000 rows. The top layer has four clusters, two of which have sub-clusters as Figure 8 shows. Both types have the clearly defined clustering structure, and each record in a generated dataset distinctly belongs to one cluster. We generate ten datasets for each type of structure, named DS1-i and DS2-i, $1 \leqslant i \leqslant 10$, respectively.

We also use three "real" datasets, "Soybean-small", "Congressional votes" and "Zoo" in the experiments. All of the three are from UCI KDD Archive [1]. *Soybean-small data* is a dataset used to classify the soybean diseases. The dataset has 47 records and each record has 35 attributes describing the features of the plant. There are four classes in the dataset. *Congressional votes* is also a Boolean dataset containing US Congressional Voting Records for the year 1984. The dataset has 435 records. Each record has a Congressman's votes on 16 issues (i.e. 16 attributes). We use the 16 attributes to classify the Congressman to "Democrat" or "Republican". *Zoo data* contains the feature description of the animals in a zoo. There are 101 animal instances, classified to 7 categories. Each record has 17 attributes describing different features of animal, such as hair and the number of legs, most of which are boolean.

[1] http://www.ics.uci.edu/∼mlearn/MLRepository.html

5.2 Compared Algorithms

Literally, any categorical clustering algorithm that employs the same entropy minimization criterion can possibly generate a valid BkPlot. However, the quality of the BkPlots can be easily influenced by the algorithms. We briefly introduce another two algorithms, Monte-Carlo algorithm and Coolcat algorithm in this section. Both use expected entropy to evaluate the quality of partition and try to minimize the expected entropy in order to achieve an approximately optimal partition.

Monte-Carlo Method [25] is a top-down partitioning algorithm. With a fixed K, it begins with all records in one cluster and follows an iterative process. In each iteration, the algorithm randomly picks one record from one of the K clusters and puts it into another randomly selected cluster. If the change of assignment does not reduce the expected entropy, the record is put back to the original cluster. Theoretically, given a sufficiently large s, the algorithm will eventually terminate at an optimal or near-optimal solution. In the experiments, we set $s = 5000$ for running MC on the synthetic datasets.

Coolcat [6] algorithm begins with selecting K records, which maximize the K-record entropy, from a sample of the dataset as the initial K clusters. It sequentially processes the rest records and assigns each to one of the K cluster. In each step, the algorithm finds the best fitted one of the K clusters for the new record – adding the new record to the cluster will result in minimum increase of expected entropy. The data records are processed in batches. Because the order of processing points has a significant impact on the quality of final clusters, there is a "re-clustering" procedure at the end of each batch. This procedure picks m percentage of the worst fitted records in the batch and re-assigns them to the K clusters in order to maintain relatively low expected entropy.

We run the algorithm on each dataset with a large sample

size (50% of the datasets) and $m = 20\%$, which is sufficient for improvement through re-clustering [6]. In order to reduce the effect of ordering, we run Coolcat 20 times for each datasets. Each run processes the data in a randomly generated sequence and we select the result having the lowest expected entropy.

5.3 Performance Measures

We use four measures to evaluate the quality of BkPlots generated by different algorithms.

- *Coverage Rate.* We evaluate the robustness of BkPlot with "Coverage Rate (CR)", i.e., the percentage of inherent significant Ks are indicated by the BkPlot. There could be more than one significant clustering structures for a particular dataset. For example, four-cluster and six-cluster structures can be all significant for DS2. A robust BkPlot should always include all of the significant Ks.

- *False Discovery Rate.* There could be some Ks, which are actually not critical but suggested by some BkPlots. In order to efficiently find the most significant ones, we prefer a BkPlot to have less false indicators as possible. We use "*False Discovery Rate*(FDR)" to represent the percentage of the noisy indicators in the BkPlot.

- *Expected Entropy.* Since the BkPlot is indirectly related to expected entropy through ECG, it is also reasonable to check the quality of expected entropy for the partitions generated by different algorithms at the particular Ks. The quality of expected entropy can be evaluated by two parts [24]: the deviation to the optimal expected entropy, and the variance of the estimated expected entropy. If an algorithm generates BkPlots with the lowest expected entropy as well as the minimum variance among the three algorithms, we can firmly conclude that this is the best one on the three.

- *Purity.* For the real datasets, there is no documented clustering structure, but the class definition is given. The purity of a cluster [30], $P(C_k)$, measures the extent to which the cluster contains data points primarily from a single class. The purity of clustering result is the weighted sum of the purity of individual cluster, given by $Purity = \sum_{k=1}^{K} \frac{n_k}{n} P(C_k)$

5.4 Discussion

The BkPlots generated by ACE algorithm for DS1 (Figure 10 clearly indicate '3' is the only significant K. The datasets having the same clustering structure should have almost the identical BkPlots. The identical BkPlots on ten different DS1-i, $0 \leqslant i \leqslant 10$, shows that ACE is a robust algorithm for generating BkPlot.

The peaks of BkPlots for DS2-i (Figure 13) include the two inherent significant Ks – '4' and '6', but '2' is also given as the third significant K. However, we notice that the peak values at 'K=4' or 'K=6' for different DS2 datasets are almost same, while those at 'K=2' have more variation. This solicits us to consider a more reliable method to estimate the most significant K for a considerably large dataset. We can uniformly generate a bunch of sample sets, which should have the identical clustering structure with the original dataset. The most stable peaks in the BkPlots of the sample sets correspond to the most significant Ks.

The BkPlots generated by Monte-Carlo algorithm for DS1 (Figure 11) also clearly identify that '3' is the best K with very small variation. However, the BkPlots for DS2 show large variation on Ks. In order to clearly observe the difference, we only show five BkPlots for DS2-i, $1 \leqslant i \leqslant 5$, respectively. Overall, the Ks distribute from '2' to '10' for different DS2-i. Some BkPlots include the significant Ks - '4' and '6', while others miss one or both, which implies that MC algorithm might not be robust enough for datasets having complicated clustering structure. The reason is MC algorithm becomes more likely to trap in local minima with the increasing complexity of clustering structure and the increasing number of clusters, since the corresponding search space increases exponentially.

Coolcat algorithm is the least robust one for generating BkPlots. It brings large variation for both datasets (Figure 12 and 15). Coolcat algorithm is originally designed for fast processing of categorical data while the quality of result is not well guaranteed. Therefore, it is not suitable for generating robust BkPlots for precisely analyzing the clustering structure.

We summarize the result with the discussed measures, Coverage Rate (CR), False Discovery Rate (FDR), and expected entropy (EE) in Table 2 and 3. The higher the coverage rate, the more robust the BkPlot is. The lower the false discovery rate the more efficient the BkPlot is. The numbers are the average over the 10 datasets. For both types of dataset, ACE shows the minimum expected entropy and minimum standard deviation, as well as the highest CR and lowest FDR. Therefore, the BkPlots generated by ACE are the most robust and efficient ones.

	CR	FDR	EE
ACE	100%	0%	0.732 ± 0.001
MC	100%	0%	0.733 ± 0.001
Coolcat	60%	85%	1.101 ± 0.026

Table 2. Summary for DS1-i

	CR	FDR	EE $K = 4$	EE $K = 6$
ACE	100%	33%	0.562 ± 0.002	0.501 ± 0.001
MC	80%	53%	0.565 ± 0.009	0.521 ± 0.008
Coolcat	60%	70%	0.852 ± 0.023	0.761 ± 0.021

Table 3. Summary for DS2-i

We run experiments on real datasets with ACE only and the results match the domain knowledge very well. We are not clear about the best K for the inherent clustering struc-

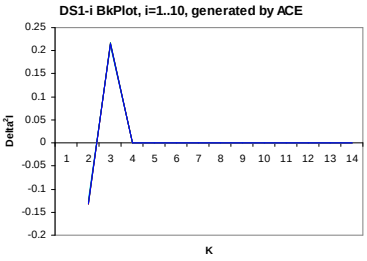


Figure 10. ACE for DS1

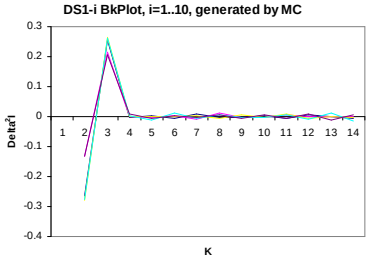


Figure 11. MC for DS1

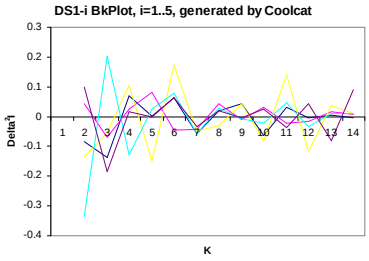


Figure 12. Coolcat for DS1

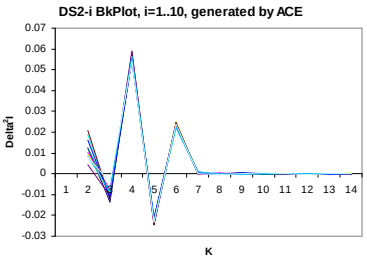


Figure 13. ACE for DS2

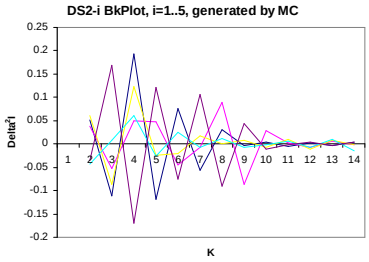


Figure 14. MC for DS2

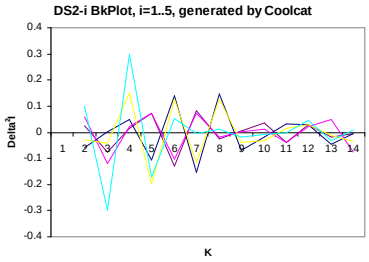


Figure 15. Coolcat for DS2

dataset	N	d	# class	Best Ks	Purity
soybean-small	47	35	4	{2,4,7}	100%
votes	435	16	2	{2}	83%
zoo	101	17	7	{2,4,7}	93.1%

Table 4. ACE result for real datasets

ture, but we can use the documented number of classes as the reference number. Interestingly, the BkPlots of ACE shows that these numbers are all included in the best Ks, which implies that the inherent structure is consistent with the domain knowledge. In fact, the additional best Ks can be investigated further to explore more hidden knowledge. For example, 'K=2' and 'K=4' for zoo dataset might be other meaningful categorizations for the animals. The high purity also shows that the entropy-based categorical clustering can generate results highly consistent with the domain knowledge, which have been supported by other literatures [6, 25]. The result encourages us to believe that BkPlots with ACE can actually work effectively for the real datasets.

6 Related Work

While many numerical clustering algorithms [22, 23] have been published, only a handful of categorical clustering algorithms appear in literature. Although it is unnatural to define a distance function between categorical data or to use the statistical center (the mean) of a group of categorical items, there are some algorithms, for example, K-Modes [21] algorithm and ROCK [19] algorithm, trying to fit the traditional clustering methods into categorical data. However, since the numerical similarity/distance function may not describe the categorical properties properly and intuitively, it leaves little confidence to the clustering result.

Gibson et al. introduced STIRR [18], an iterative algorithm based on non-linear dynamical systems. STIRR represents each attribute value as a weighted vertex in a graph. Starting with the initial conditions, the system is iterated until a "fixed point" is reached. When the fixed point is reached, the weights in one or more of the "basins" isolate two groups of attribute values on each attribute. Due to the complexity and unintuitive mechanism, the users may hesitate to use it.

CACTUS [17] adopts the linkage idea from ROCK and names it "strong connection". However, the similarity is calculated by the "support". A cluster is defined as a region of attributes that are pair-wise strongly connected.Similarly, the concept of "support" or linkage is still indirect in defining the similarity of categorical data, and unnecessarily makes the clustering process complicated.

Cheng et al. [13] applied the entropy concept in numerical subspace clustering, and Coolcat [6] introduced the entropy concept into categorical clustering. We have briefly introduced Coolcat in section 5. Some closely related work also borrows concepts from information theory, including Co-clustering [15], Information Bottleneck [28] and LIMBO [3].

C. Aggarwal [1] demonstrated that localized associations are very meaningful to market basket analysis. To find the localized associations, they introduced a categorical clustering algorithm CLASD to partition the basket data. A new similarity measure is defined for any pair of transactions. CLASD is still a kind of traditional clustering algorithm –

the special part is only the definition of similarity function for categorical data. Thus, it has the similar problem as we described.

Most of the recent research in categorical clustering is focused on clustering algorithms. Surprisingly, there is little research concerning about the cluster validation problems for categorical datasets.

7 Conclusion

Most of the recent research about categorical clustering has only contributed to categorical clustering algorithms. In this paper, we proposed an entropy-based cluster validation method for identifying the best Ks for categorical data clustering. Our method suggests to find the best Ks by observing the "Entropy Characteristic Graph (ECG)", which describes the entropy property of partitions with varying K and is significant in characterizing the clustering structure of categorical data. The "Best-K plot (BkPlot)" is used to find the significant points conveniently from the Entropy Characteristic Graph. In order to find the robust BkPlot, we also develop an entropy-based agglomerative algorithm ACE. Our experiments show that, ACE can generate the most robust BkPlots for various experimental datasets, compared to the other two typical entropy-based algorithms. Meanwhile, ACE can also find high quality clustering results in terms of the entropy criterion. Therefore, BkPlot validation method with ACE algorithm can serve as an effective tool for analyzing the significant clustering structures in categorical datasets.

8 Acknowledgement

This research is partially supported by NSF CNS, NSF CCR, NSF ITR, DoE SciDAC, DARPA, CERCS Research Grant, IBM Faculty Award, IBM SUR grant, HP Equipment Grant, and LLNL LDRD.

References

[1] C. C. Aggarwal, C. Magdalena, and P. S. Yu. Finding localized associations in market basket data. *IEEE Trans. on Knowledge and Data Eng.*, 14(1):51–62, 2002.

[2] A. Agresti. *Categorical Data Analysis*. Wiley-Interscience, 1990.

[3] P. Andritsos, P. Tsaparas, R. J. Miller, and K. C. Sevcik. Limbo:scalable clustering of categorical data. *Proc. of Intl. Conf. on Extending Database Technology (EDBT)*, 2004.

[4] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. *Proc. of ACM SIGMOD Conference*, 1999.

[5] D. Barbara and S. Jajodia, editors. *Applications of Data Mining in Computer Security*. Klumer Academic Publishers, 2002.

[6] D. Barbara, Y. Li, and J. Couto. Coolcat: an entropy-based algorithm for categorical clustering. *Proc. of ACM Conf. on Information and Knowledge Mgt. (CIKM)*, 2002.

[7] F. Baulieu. Two variant axiom systems for presence/absence based dissimilarity coefficients. *Journal of Classification*, 14, 1997.

[8] A. Baxevanis and F. Ouellette, editors. *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins, 2nd edition*. Wiley-Interscience, 2001.

[9] H. Bock. Probabilistic aspects in cluster analysis. *Conceptual and Numerical Analysis of Data*, 1989.

[10] M. Brand. An entropic estimator for structure discovery. In *Proc. Of Neural Information Processing Systems (NIPS)*, pages 723–729, 1998.

[11] G. Celeux and G. Govaert. Clustering criteria for discrete data and latent class models. *Journal of Classification*, 1991.

[12] K. Chen and L. Liu. VISTA: Validating and refining clusters via visualization. *Information Visualization*, 3(4), 2004.

[13] C. H. Cheng, A. W.-C. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. *Proc. of ACM SIGKDD Conference*, 1999.

[14] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.

[15] I. S. Dhillon, S. Mellela, and D. S. Modha. Information-theoretic co-clustering. *Proc. of ACM SIGKDD Conference*, 2003.

[16] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[17] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS-clustering categorical data using summaries. *Proc. of ACM SIGKDD Conference*, 1999.

[18] D. Gibson, J. Kleinberg, and P. Raghavan. Clustering categorical data: An approach based on dynamical systems. *Proc. of Very Large Databases Conference (VLDB)*, 8(3–4):222–236, 2000.

[19] S. Guha, R. Rastogi, and K. Shim. ROCK: A robust clustering algorithm for categorical attributes. *Proc. of IEEE Intl. Conf. on Data Eng. (ICDE)*, 1999.

[20] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Cluster validity methods: Part I and II. *SIGMOD Record*, 31(2):40–45, 2002.

[21] Z. Huang. A fast clustering algorithm to cluster very large categorical data sets in data mining. *Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1997.

[22] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice hall, 1988.

[23] A. K. Jain and R. C. Dubes. Data clustering: A review. *ACM Computing Surveys*, 31, 1999.

[24] E. L. Lehmann and G. Casella. *Theory of Point Estimation*. Springer-Verlag, 1998.

[25] T. Li, S. Ma, and M. Ogihara. Entropy-based criterion in categorical clustering. *Proc. of Intl. Conf. on Machine Learning (ICML)*, 2004.

[26] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[27] S. Sharma. *Applied Multivariate Techniques*. Wiley&Sons, 1995.

[28] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. *Proc. of the 37-th Annual Allerton Conference on Communication, Control and Computing*, 1999.

[29] N. Wrigley. *Categorical Data Analysis for Geographers and Environmental Scientists*. Longman, 1985.

[30] Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55:311–331, 2004.

Local Computation of Answers to Table Queries on Summary Databases

Francesco M. Malvestuto
Computer Science Department
"La Sapienza" University of Rome, Italy
malvestuto@di.uniroma1.it

Elaheh Pourabbas
IASI "Antonio Ruberti"
National Research Council, Rome, Italy
pourabbas@iasi.cnr.it

Abstract

We address the problem of evaluating table queries from a summary database formed by a collection of pre-computed tables on certain measure variables. We assume that every table query asks for the distribution of a measure variable of interest, and that the summary database contains tables on the variable of interest as well as on other measure variables. If the requested distribution is none of the base tables and cannot be exactly derivable from none of them, then the answer to the query will be the result of an estimation procedure, which may bring up another measure variable that is correlated to the measure variable of interest. We give an estimation procedure that combines the "divide-and-conquer" principle with tree computations.

1. Introduction

A recent querying paradigm, called *On-Line Analytical Processing* (OLAP) [6], often involves complex queries over very large multidimensional relations or datacubes with category (or dimensional) and measure (or summary) attributes. Obtaining the exact answer to an OLAP query can be prohibitively expensive in terms of time and/or storage space in data warehouse environment. In order to reduce the computational effort, a promising approach is to store some aggregate data (as " materialized views") in a summary database, which is used to answer OLAP queries.

In this paper, we consider the problem of evaluating table queries from a summary database formed by a collection of pre-computed tables on certain measure variables. We suppose that each table query asks for the distribution of a measure variable of interest, called the *target variable* (e.g., Personnel 2001), by a set of category attributes (e.g., *gender*, *state*,...), and that the summary database contains tables on the target variable as well as on other measure variables. If the table requested by a query is none of the base tables and cannot be exactly derived from them [13] [5], then the query can be answered in an approximate (and, hopefully, accurate) way using some information-theoretic estimation criterion which, on demand of the user, may bring up an additional measure variable (e.g., Personnel 2000, Total-Income 2001), called the *auxiliary variable*. We can use as estimation criterion the principle of *minimum cross-entropy* or the principle of *maximum entropy* [17], depending on whether or not the target and auxiliary variables can be viewed as terms of a time series (see the following example). Both are based on the proportionality principle and have been successfully applied to the *small-area estimation* [9] and to the analysis of inter-industry transactions with *input-output matrices* [1] [10][18].

Example 1 A summary database contains four tables: two on the measure variable *Personnel 2001*, one table on the measure variable *Total-Income 2001*, and one table on the measure variable *Personnel 2000*. The two tables on *Personnel 2001* report the distributions $p_1(g, a)$ and $p_2(d, a)$ of employees in 2001 by *gender* and *age-class*, and by *department* and *age-class*, respectively. The table on *Total-Income 2001* reports the distribution $t(g, l)$ of total income in 2001 by *gender* and *level*. The table on *Personnel 2000* reports the distribution $q(g, d, a, l)$ of employees in 2000 by *gender*, *department*, *age-class* and *level*.
Suppose that a user asks for the distribution of employees in 2001 by *age-class* and *level*. Then, the query system advices the user that he will receive an estimate of the requested table and that he may tune the answer to some auxiliary variable. We now discuss three typical cases:

Case 1: the user selects no auxiliary variable. Let $\hat{p}(g, d, a)$ be the maximum entropy extension of $p_1(g, a)$ and $p_2(d, a)$. Then, the query will be answered by issuing the marginal on a and l of the distribution $\dfrac{\hat{p}(g, d, a)}{L}$, where L is the number of possible levels.

Case 2: the user selects *Total-Income 2001* as auxiliary variable. Let $\hat{p}(g,d,a)$ be as above and let $t(g)$ be the marginal on g of $t(g,l)$. Then, the query will be answered by issuing the marginal on a and l of the distribution $\frac{\hat{p}(g,d,a)t(g,l)}{t(g)}$.

Case 3: the user selects *Personnel 2000* as auxiliary variable. Let $\tilde{p}(g,d,a)$ be the minimum cross-entropy extension of $p_1(g,a)$ and $p_2(d,a)$ relative to the marginal $q(g,d,a)$ of the distribution $q(g,d,a,l)$. Then, the query will be answered by issuing the marginal on a and l of the distribution $\frac{\tilde{p}(g,d,a)q(g,d,a,l)}{q(g,d,a)}$.

In this paper we show how to solve the problem of answering a table query using only tables stored in a summary database, referred to as the *Table-Query Problem*. The proposed procedure is inspired by the "divide-and-conquer" principle and generalizes that given in [17], which applies only to the query that asks for the distribution of the target variable by all the category attributes of the target tables and the auxiliary table.

The paper is structured as follows. In the next section, we state the two Proportional Estimation Models PEM1 and PEM2 and the Table-Query Problem. In Sections 3 and 4, we solve the Table-Query Problem under the models PEM1 and PEM2, respectively. Finally, Section 5 concludes.

2. The Table-Query Problem

Henceforth, we assume that all measure variables are of nonnegative-real type and of additive nature. Let *X* be a set of (category) attributes. The *domain* of *X*, written *dom*(X), is the set of all semantically possible tuples on *X*; by *size*(X) we denote the cardinality of *dom*(X). Let $p(x)$ be a nonnegative real-valued function defined on *dom*(X); the *support* of $p(x)$ is the relation with scheme X containing all tuples x with $p(x) \neq 0$. The pair $T = \langle X, p(x)\rangle$ defines a *(summary) table*, of which X is the *scheme* and $p(x)$ the *distribution*. Without loss of generality, we always assume that the data reported in every table are normalized to one. Let *Y* be a subset of X; the *marginal* of T with respect to Y is the table $T(Y) = \langle Y, p(y)\rangle$ where $p(y)$ is the marginal of the distribution $p(x)$, that is, $p(y) = \sum_x p(x)$ the summation being extended over all tuples *x* in $dom(X)$ whose restrictions to *Y* coincide with *y*. Note that the support of $p(y)$ is the (relation-theoretic) projection onto Y of the support of $p(x)$. We also admit the case Y is empty; then, $p(y)$ is the unity. Let $\mathcal{T} = \{T_1, ..., T_n\}$ be a set of tables, where T_i has scheme X_i, for all i. The set X given by the union of the schemes X_i of the tables T_i and their collection **H** will be referred to as the *base set* and the *scheme* of $\mathcal{T}$, respectively. The table set $\mathcal{T}$ is *consistent* if there exists at least one table T with scheme X such that the marginal of T with respect to X_i coincides with T_i, for all i. Such a table is called a *universal table* of $\mathcal{T}$.

Suppose we are given a table query and that the user has selected a certain auxiliary variable. Let us assume that the summary database contains the set of tables $\mathcal{T} = \{T_1, ..., T_n\}$ on the target variable, where T_i has scheme X_i, and the table $\langle Y, q(y)\rangle$ on the auxiliary variable. Consider the following two Proportional Estimation Models where $p(x,y)$ denotes the distribution of an unknown table with scheme $X \cup Y$.

PEM 1
Marginal constraints: $p(x_i) = p_i(x_i),\ \ i = 1, \ldots, n$ *Proportionality criterion:* Let $Z = X \cap Y$. There exist real-valued functions $g_1(x_1), \ldots, g_n(x_n)$ such that the factorization $p(x,y) = g_1(x_1)\cdots g_n(x_n)\frac{q(y)}{q(z)}$ holds for every tuple (x,y) in the support of $p(x,y)$.

PEM 2
Marginal constraints: $p(x_i) = p_i(x_i),\ \ i = 1, \ldots, n$ *Proportionality criterion:* There exist real-valued functions $g_1(x_1), \ldots, g_n(x_n)$ such that the factorization $p(x,y) = g_1(x_1)\cdots g_n(x_n)q(y)$ holds for every tuple (x,y) in the support of $p(x,y)$.

Using the results proven in our previous paper [17], one has that: PEM1 has a unique solution, we denote by $\hat{p}(x,y)$, and PEM2 has a solution if and only if there exists a universal table $T = \langle X, p(x)\rangle$ of $\mathcal{T}$ such that the support of the marginal of $p(x)$ with respect to $Z = X \cap Y$ is contained in the support of the marginal of $q(y)$ with respect to Z, and if this is the case then PEM2 has a unique solution, we denote by $\tilde{p}(x,y)$. At this point, we can state the *Table-Query Problem* we want to solve:

Given a nonempty subset U of $X \cup Y$, find the marginal with respect to U of $\hat{p}(x,y)$ or $\tilde{p}(x,y)$ depending on whether PEM1 or PEM2 is in use. In [17] the Table-Query Problem was solved for the special case that $U = X \cup Y$. We now state some useful formulas for solving the Table-Query Problem in the general case. By summing out the variables in $Y - Z$ in the functional expressions of $\hat{p}(x,y)$ and $\tilde{p}(x,y)$, we obtain

$$\hat{p}(x) = g_1(x_1)\ldots g_n(x_n) \tag{1}$$

for every tuple x in the support of $\hat{p}(x)$, and

$$\tilde{p}(x) = g_1(x_1)\ldots g_n(x_n)q(z) \tag{2}$$

for every tuple x in the support of $\tilde{p}(x)$. Formulas (1) and (2) lead to the following expressions for the solutions to PEM1 and PEM2:

$$\hat{p}(x,y) = \hat{p}(x)\frac{q(y)}{q(z)} \qquad \tilde{p}(x,y) = \tilde{p}(x)\frac{q(y)}{q(z)} \tag{3}$$

Using formulas (3), we can easily find the expressions for $\hat{p}(u)$ and $\tilde{p}(u)$. Let $X' = (X-Y)\cap U$, $Y' = (Y-X)\cap U$, $Z' = Z \cap U$ and $Z'' = Z - U$. Then, $U = X' \cup Y' \cup Z'$ and we have:

$$\begin{aligned} \hat{p}(u) &= \textstyle\sum_{z''} \dfrac{\hat{p}(x',z',z'')q(y',z',z'')}{q(z',z'')} \quad \text{(i)} \\ \tilde{p}(u) &= \textstyle\sum_{z''} \dfrac{\tilde{p}(x',z',z'')q(y',z',z'')}{q(z',z'')} \quad \text{(ii)} \end{aligned} \tag{4}$$

Example 2 Suppose that a user asks for the table on a certain measure variable with scheme $abdhk$. Let $\mathcal{T} = \{T_1, \ldots, T_{12}\}$ be the set of the base tables on the target variable (see Figure 1).

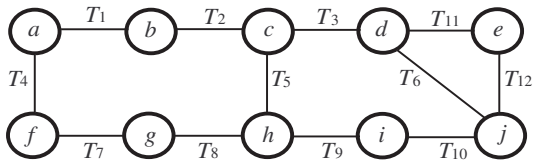


Figure 1. The table set $\mathcal{T}$ on the target variable

Note that the scheme $\mathbf{H} = \{ab, af, bc, cd, ch, de, dj, ej, fg, gh, hi, ij\}$ of $\mathcal{T}$ does not contain the attribute k. Suppose that the user selects an auxiliary variable for which there exists a base table with scheme $bcgkl$ and distribution q. Then, $4(i)$ and $4(ii)$ read:

$$\hat{p}(a,b,d,h,k) = \textstyle\sum_{c,\,g} \dfrac{\hat{p}(a,b,c,d,g,h)q(b,c,g,k)}{q(b,c,g)}$$
$$\tilde{p}(a,b,d,h,k) = \textstyle\sum_{c,\,g} \dfrac{\tilde{p}(a,b,c,d,g,h)q(b,c,g,k)}{q(b,c,g)}$$

Suppose that we are able to compute the distributions $\hat{p}(x',z',z'')$ and $\tilde{p}(x',z',z'')$. Then, the procedure below yields $\hat{p}(u)$ and $\tilde{p}(u)$.

MARGINAL
(1) Find the distribution $\hat{p}(x',z',z'')$ (respectively, and $\tilde{p}(x',z',z'')$).
(2) Marginalize $q(y)$ with respect to $Y' \cup Z$ and Z.
(3) Compute $\hat{p}(u)$ (respectively, $\tilde{p}(u)$) using $4(i)$ (respectively, $4(ii)$).

The execution of Steps 2 and 3 of MARGINAL is a matter of routine; so, we focus on Step 1. Let $V = X' \cup Z$. Of course, the distributions $\hat{p}(v)$ and $\tilde{p}(v)$ are also the marginals with respect to V of $\hat{p}(x)$ and $\tilde{p}(x)$, respectively. Therefore, Step 1 requires solving the following problem:

Find the marginal with respect to V of $\hat{p}(x)$ (or of $\tilde{p}(x)$). A brute-force approach to solving this problem consists in first finding $\hat{p}(x)$ (or $\tilde{p}(x)$) and, then, marginalizing it. We now show how to compute $\hat{p}(x)$ and $\tilde{p}(x)$. First of all, observe that they are the distributions of two universal tables of $\mathcal{T}$, we denote by $\hat{T}$ and $\tilde{T}$, both of which, by formulas (1) and (2), have the following form:

$$f_1(x_1)\ldots f_n(x_n)\pi(x) \tag{5}$$

for some real-valued functions $f_1(x_1), \ldots, f_n(x_n)$ and for some distribution $\pi(x)$ over X. Explicitly, $\pi(x) = \dfrac{1}{size(X)}$ for $\hat{p}(x)$, and $\pi(x) = \dfrac{q(z)}{size(X-Z)}$ for $\tilde{p}(x)$. Now, it is well-known [3] [7] that, if $p(x)$ is the distribution of a universal table of $\mathcal{T}$ having the form (5), then $p(x)$ can be computed using the iterative procedure, called *Iterative Proportional Fitting Procedure* (IPFP) [8], which starts with the *zero approximation* $p^{[0]}(x) = \pi(x)$ and determines the *higher-order approximations* to $p(x)$ according to the following computation scheme:

first iteration cycle	$p^{[1]}(x)$	$\ldots$	$p^{[n]}(x)$
second iteration cycle	$p^{[n+1]}(x)$	$\ldots$	$p^{[2n]}(x)$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
h-th iteration cycle	$p^{[hn+1]}(x)$	$\ldots$	$p^{[hn+n]}(x)$
$\ldots$	$\ldots$	$\ldots$	$\ldots$

where the approximation $p^{[hn+i]}(x)$ in the $(h+1)$-th iteration cycle, $1 \le i \le n$, is obtained by fitting the approximation $p^{[hn+i-1]}(x)$ to the distribution $p_i(x_i)$ of the base table T_i:

$$p^{[hn+i]}(x) = \frac{p_i(x_i)}{p^{[hn+i-1]}(x_i)} p^{[hn+i-1]}(x).$$

From an information-theoretic point of view, the distribution $p(x)$ minimizes the cross-entropy relative to $\pi(x)$ (see Section A of the Appendix for information-theoretic definitions). So, the distribution of the universal table $\hat{T}$ minimizes the cross-entropy relative to the distribution $\dfrac{1}{size(X)}$ or, equivalently, maximizes the entropy (see Section A of the Appendix), and the distribution of the universal table $\tilde{T}$ minimizes the cross-entropy relative to the distribution $\dfrac{q(z)}{size(X-Z)}$. Accordingly, $\hat{T}$ will be referred to as the

maximum entropy universal table [11] [12] (the *ME universal table*, for short) of $\mathcal{T}$, and $\tilde{T}$ as the *minimum cross-entropy universal table* relative to $\frac{q(z)}{size(X-Z)}$ (the *q-mCE universal table*, for short) of $\mathcal{T}$. Efficient procedures for computing the distributions of $\hat{T}$ and $\tilde{T}$ can be found in [11] [12] [2] and in [17], respectively. Once $\hat{p}(x)$ (or $\tilde{p}(x)$) have been computed, its marginal with respect to V can be easily obtained. However, as is shown in Sections 3 and 4, in most cases both $\hat{p}(v)$ and $\tilde{p}(v)$ can be computed without passing through the computation of $\hat{p}(x)$ and $\tilde{p}(x)$.

3 Marginalizing $\hat{T}$ with respect to V

An efficient procedure for computing the distribution $\hat{p}(v)$ from $\mathcal{T}$ rests on the notion of "collapsibility" [14] we now recall. Let $\mathbf{H} = \{X_1, \ldots, X_n\}$ be the scheme of $\mathcal{T}$ and X its base set; the *projection* of $\mathcal{T}$ onto a subset W of X is the table set $\mathcal{T}(W) = \{T_1(X_1 \cap W), \ldots, T_n(X_n \cap W)\}$, where redundant tables are omitted. The ME universal table $\hat{T}$ of $\mathcal{T}$ is *collapsible* onto W if the marginal of $\hat{T}$ with respect to W coincides with the ME universal table of the projection of $\mathcal{T}$ onto W. So, if W is a (possibly improper) superset of V that the ME universal table of $\mathcal{T}$ is collapsible onto, then $\hat{p}(v)$ can be obtained by first computing the ME universal table $\hat{T}(W)$ of $\mathcal{T}(W)$ and, then, marginalizing the distribution of $\hat{T}(W)$ with respect to V. The best choice for W will fall upon a minimal superset of V that the ME universal table of $\mathcal{T}$ is collapsible onto. Such a superset of V is unique and is called the *closed hull* of V in $\mathbf{H}$ [14]; furthermore, it coincides with the "canonical closure" [15] [16] of $\mathbf{H}$ when $\mathbf{H}$ is viewed as a hypergraph (see Section B of the Appendix for hypergraph-theoretic definitions). The procedure for finding the closed hull of V in $\mathbf{H}$ is based on the notion of the *compaction* of $\mathbf{H}$ [15] [16], which is the finest of the acyclic covers $\mathbf{K}$ such that the ME universal table of $\mathcal{T}$ is collapsible onto each edge of $\mathbf{K}$. It has a number of nice properties, two of which read: (a) the separators of $\mathbf{H}$ and of the compaction of $\mathbf{H}$ are the same; (b) if $\mathbf{H}$ is acyclic, then (and only then) the compaction of $\mathbf{H}$ coincides with $\mathbf{H}$. Let $\mathbf{K}$ be the compaction of $\mathbf{H}$. For each edge C of $\mathbf{K}$, we call the table set $\mathcal{T}(C)$ a *component* of $\mathcal{T}$.

Example 2 (*continued*). The compaction of $\mathbf{H}$ is $\mathbf{K} = \{abcfgh, cdhij, dej\}$. The components of the table set of Figure 1 are shown in Figure 2.

Given $\mathbf{H}$, the compaction $\mathbf{K} = \{C_1, \ldots, C_m\}$ of $\mathbf{H}$ and the set V, the closed hull of V in $\mathbf{H}$, say W, can be determined using the following algorithm [15], whose Step 1 performs the *selective reduction* of $\mathbf{K}$ with *sacred set* V [19]:

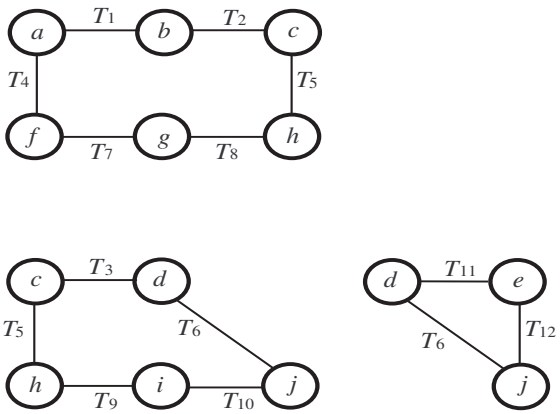


Figure 2. The components of the table set $\mathcal{T}$

CLOSED HULL

(1) Repeatedly apply the following two operations until neither can be longer applied:
(i) Delete a vertex of $\mathbf{K}$ if it is not in V and belongs to exactly one edge;
(ii) Delete an edge of $\mathbf{K}$ if it is contained in another edge.
(2) Let $\mathbf{K}' = \{C'_{j_1}, \ldots, C'_{j_k}\}$ be the resulting hypergraph, where C'_{j_h} is the "residual part" of the edge C_{j_h} of $\mathbf{K}$. For each h, $1 \leq h \leq k$, set E_h to C'_{j_h} if C'_{j_h} is contained in some edge X_i of $\mathbf{H}$, and to C_{j_h} otherwise.
(3) Set W to the empty set. For each h, $1 \leq h \leq k$, set $W := W \cup E_h$

It should be noted that the sets $E_1, \ldots, E_k$ of Step 2 of CLOSED HULL are exactly the edges of the subhypergraph $\mathbf{K}(W)$ of $\mathbf{K}$ induced by W and that $\mathbf{K}(W)$ is an acyclic hypergraph. Moreover, since by property (b) the compaction of $\mathbf{K}$ is $\mathbf{K}$ itself, the set W is the closed hull of itself not only in $\mathbf{H}$ but also in $\mathbf{K}$. Finally, if $\mathbf{H}$ is acyclic, then $\mathbf{H} = \mathbf{K}$ and $\mathbf{K}(W) = \mathbf{H}(W)$. After determining the closed hull W of V in $\mathbf{H}$, the distribution $\hat{p}(w)$ can be obtained without passing through the computation of $\hat{p}(x)$ simply by applying the IPFP to $\mathcal{T}(W)$ with zero approximation the distribution $\frac{1}{size(W)}$.

Example 2 (*continued*). With $V = abcdgh$, the selective reduction of $\mathbf{K}$ with sacred set V (see Step 1 of CLOSED HULL) is the hypergraph $\mathbf{K}' = \{abcgh, cdh\}$, where $abcgh$ and cdh are the residual parts of the edges $abcfgh$ and $cdhij$ of $\mathbf{K}$, respectively. Since neither $abcgh$ nor cdh is contained in any edge of $\mathbf{H}$, the result of Step 3 of CLOSED HULL is $W = abcdfghij$, which hence is the closed hull of V. So, $\hat{p}(w)$ can be obtained as the ME universal table

of the projection of $\mathcal{T}$ onto W (see Figure 3), that is, by applying the IPFP to $\mathcal{T}(W)$ with zero approximation the distribution $\frac{1}{size(abcdfghij)}$.

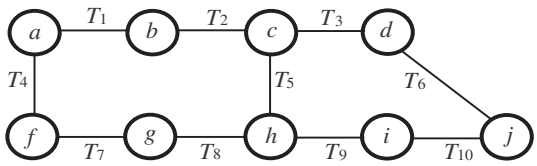


Figure 3. The projection of $\mathcal{T}$ onto W

However, we can furthermore reduce the time and space costs using the implementation of the IPFP given in [11] [12] for computing the ME universal table $\hat{P}$ of any consistent table set $\mathcal{P}$ with scheme $\mathbf{R}$, we now recall. The implementation is based on the following two properties of $\hat{P}$ with respect to any acyclic cover $\mathbf{S}$ of $\mathbf{R}$:

(i) $\hat{P}$ coincides with the ME universal table of the set of the marginals of $\hat{P}$ with respect to the edges of $\mathbf{S}$, and

(ii) the ME universal table of the set of the marginals of $\hat{P}$ with respect to the edges of $\mathbf{S}$ has a closed-form expression which, for any *join tree* J of $\mathbf{S}$ (see Section B of the Appendix), consists of a ratio whose numerator is the product of the marginals of $\hat{P}$ with respect to the labels of nodes of J (i.e., with respect to the edges of $\mathbf{S}$) and whose denominator is the product of the marginals of $\hat{P}$ with respect to the labels of arcs of J. Such a closed-form expression of $\hat{P}$ will be referred to as the *tree formula* generated by $\mathbf{S}$.

A first consequence is that, with $\mathcal{P} = \mathcal{T}$, $\mathbf{R} = \mathbf{H}$ and $\mathbf{S} = \mathbf{K}$, where $\mathbf{K}$ is the compaction of $\mathbf{H}$, the entries in the tree formula for $\hat{T}$ generated by $\mathbf{K}$ can be computed locally. More precisely, since $\hat{T}$ is collapsible onto each edge of $\mathbf{K}$, the marginal of $\hat{T}$ with respect to each edge of $\mathbf{K}$ can be computed as the ME universal table of the corresponding component of $\mathcal{T}$; moreover, by property (a) of $\mathbf{K}$, the separators of $\mathbf{K}$ are the same as $\mathbf{H}$, so that the marginal of $\hat{T}$ with respect to each separator of $\mathbf{K}$ can be obtained by marginalizing some table T_i in $\mathcal{T}$.

Example 2 (*continued*). Recall that the compaction of $\mathbf{H}$ is $\mathbf{K} = \{abcfgh, cdhij, dej\}$. The separators of $\mathbf{K}$ (and, hence, of $\mathbf{H}$) are ch and dj. A join tree of $\mathbf{K}$ is shown in Figure 4.

Therefore, the tree formula for $\hat{T}$ generated by $\mathbf{K}$ reads:

$$\frac{\hat{p}(abcfgh)\hat{p}(cdhij)\hat{p}(dej)}{p_5(ch)p_6(dj)}$$

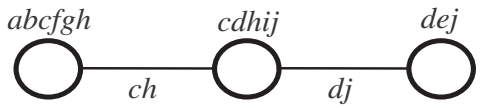


Figure 4. The join tree of $\mathbf{K}$

where $\hat{p}(abcfgh)$, $\hat{p}(cdhij)$ and $\hat{p}(dej)$ can be computed as the distributions of the ME universal tables of the components $\mathcal{T}(abcfgh)$, $\mathcal{T}(cdhij)$ and $\mathcal{T}(dej)$ of $\mathcal{T}$ (see Figure 2), respectively, that is, by applying the IPFP procedure to $\mathcal{T}(abcfgh)$, $\mathcal{T}(cdhij)$ and $\mathcal{T}(dej)$ with zero approximations $\frac{1}{size(abcfgh)}$, $\frac{1}{size(cdhij)}$ and $\frac{1}{size(dej)}$, respectively.

We now apply the technique above to compute the ME universal table of the table set $\mathcal{T}(W)$ with scheme $\mathbf{H}(W)$. With $\mathcal{P} = \mathcal{T}(W)$, $\mathbf{R} = \mathbf{H}(W)$ and $\mathbf{S} = \mathbf{K}(W)$, the entries in the tree formula for $\hat{T}(W)$ generated by $\mathbf{K}(W)$ can be computed locally. Explicitly, with the notation of CLOSED HULL, the marginal of $\hat{T}(W)$ with respect to each edge E_h of $\mathbf{K}(W)$ can be computed as the ME universal table of the corresponding component $\mathcal{T}(E_h)$ of $\mathcal{T}(W)$, and the marginal of $\hat{T}(W)$ with respect to each separator of $\mathbf{K}(W)$ can be computed as the marginal of some table in $\mathcal{T}(W)$.

Example 2 (*continued*). Recall that $W = abcdfghij$. The compaction of $\mathbf{H}(W)$ is $\mathbf{K}(W) = \{abcfgh, cdhij\}$. The separator of $\mathbf{K}(W)$ is ch. A join tree of $\mathbf{K}(W)$ is shown in Figure 5.

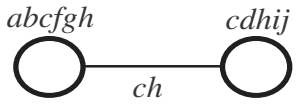


Figure 5. The join tree of $\mathbf{K}(W)$

Therefore, the tree formula for $\hat{T}(W)$ generated by $\mathbf{K}(W)$ reads:

$$\frac{\hat{p}(abcfgh)\hat{p}(cdhij)}{p_5(ch)}$$

where the distributions $\hat{p}(abcfgh)$ and $\hat{p}(cdhij)$ are computed as above.

Finally, once $\hat{T}(W)$ has been computed using its tree formula generated by $\mathbf{K}(W)$, $\hat{T}(V)$ can be obtained by marginalization. However, we can do better. Suppose that

we have already found the entries in the tree formula for $\hat{T}(W)$ generated by $\mathbf{K}(W)$; at this point, instead of computing $\hat{T}(W)$, we soon marginalize the factors $\hat{p}(e_h)$ of the numerator, for each edge E_h of $\mathbf{K}(W)$, with respect to the edge C'_{j_h} of $\mathbf{K}'$. What we obtain is a tree formula generated by $\mathbf{K}'$, which provides a closed-form expression of $\hat{T}(W')$, where W' is the vertex set of $\mathbf{K}'$, that is, $W' = W - \bigcup_{h=1,\ldots,k}(E_h - C'_{j_h})$. Finally, after computing $\hat{T}(W')$, we, marginalize $\hat{T}(W')$ with respect to V.

Example 2 (*continued*). Recall that $\mathbf{K}' = \{abcgh, cdh\}$ and $\mathbf{K}(W) = \{abcfgh, cdhij\}$. After computing the distributions $\hat{p}(abcfgh)$ and $\hat{p}(cdhij)$, we soon marginalize them with respect to the edges $abcgh$ and cdh of $\mathbf{K}'$, respectively. The vertex set of of $\mathbf{K}'$ is $W' = abcdgh$ and the tree formula for $\hat{T}(W')$ generated $\mathbf{K}'$ reads:

$$\hat{p}(abcdgh) = \frac{\hat{p}(abcgh)\hat{p}(cdh)}{p_5(ch)}$$

4 Marginalizing $\tilde{T}$ with respect to V

The notion of collapsibility of the ME universal table of $\mathcal{T}$ naturally generalizes to the q-mCE universal table of $\mathcal{T}$ as follows. The table $\tilde{T}$ is *collapsible* onto W if the marginal of $\tilde{T}$ with respect to W coincides with the minimum cross-entropy universal table of $\mathcal{T}(W)$ relative to the marginal of $\dfrac{q(z)}{size(X-Z)}$ with respect to W. Unfortunately, at the present the uniqueness of a minimal superset of V that $\tilde{T}$ is collapsible onto is an open problem. Nevertheless, we can get an effective procedure for computing the distribution $\tilde{p}(v)$ as follows. Given $\mathcal{T} = \{T_1, \ldots, T_n\}$ with scheme $\mathbf{H} = \{X_1, \ldots, X_n\}$, let us consider the (partially specified) table set $\mathcal{T}^* = \{\tilde{T}(Z), T_1, \ldots, T_n\}$, where redundant tables are omitted. It has scheme $\mathbf{H}^* = \{Z, X_1, \ldots, X_n\}$ where redundant edges are omitted. Then, it can be proven [17] that $\tilde{T}$ coincides with the ME universal table of $\mathcal{T}^*$. However, owing to the incompleteness of $\mathcal{T}^*$, we can seldom apply the technique developed in Section 3 to compute $\tilde{T}$ from $\mathcal{T}^*$. To see it, let $\mathbf{K}^*$ be the compaction of $\mathbf{H}^*$. Note that Z is a partial edge of $\mathbf{K}^*$, that is, Z is contained in at least one edge of $\mathbf{K}^*$. Of course, the closed hull W of V in $\mathbf{H}^*$ can be computed as in Section 3; but, like $\mathcal{T}^*$, also the projection of $\mathcal{T}^*$ onto W is partially specified unless the intersection of Z with each edge of $\mathbf{K}^*$ is contained in some edge X_i of $\mathbf{H}$. In what follows, we assume that this is not the case for, otherwise, $\tilde{T}$ does coincide with $\hat{T}$ [17].

Example 2 (*continued*). Recall that $Z = bcg$. The table set $\mathcal{T}^*$ is obtained from $\mathcal{T}$ by replacing the table T_2 by $\tilde{T}(Z)$ (see Figure 6).

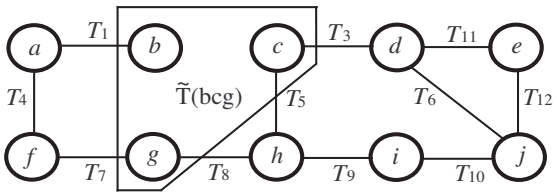


Figure 6. The table set $\mathcal{T}^*$

Analogously, the hypergraph $\mathbf{H}^*$ is obtained from $\mathbf{H}$ by replacing the edge $X_2 = bc$ by $Z = bcg$. The compaction of $\mathbf{H}^*$ is $\mathbf{K}^* = \{abfg, bcg, cgh, cdhij, dej\}$ and the components of $\mathcal{T}^*$ are shown in Figure 7.

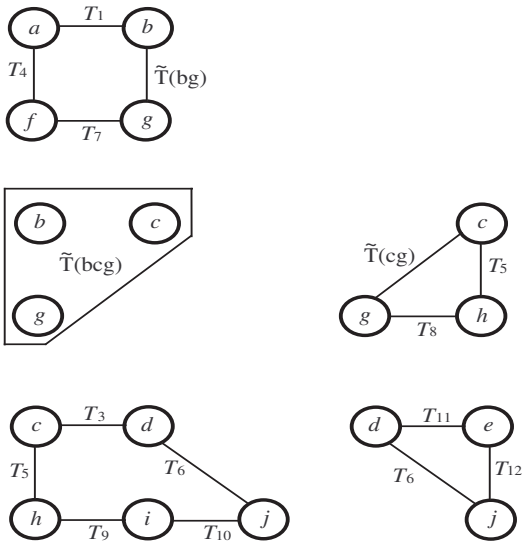


Figure 7. The components of $\mathcal{T}^*$

With input $\mathbf{K}^* = \{abfg, bcg, cgh, cdhij, dej\}$ and $V = abcdgh$, CLOSED HULL yields $W = abcdfghij$. The projection of $\mathcal{T}^*$ onto W is shown in Figure 8.

In order to overcome the above-mentioned difficulty, we now introduce a suitable acyclic cover of $\mathbf{K}^*$. Let $\mathbf{K}^* = \{C_1, \ldots, C_m\}$ and let us assume that for some k, $1 \leq k \leq m-1$, $C_{k+1}, \ldots, C_m$ are the edges of $\mathbf{K}^*$ for which the set $Z \cap C_j$ is neither empty nor contained in any edge X_i of $\mathbf{H}$. Let $C = \bigcup_{j=k+1,\ldots,m} C_j$. It is easy to see that the hypergraph $\mathbf{K} = \{C, C_1, \ldots, C_k\}$ is an acyclic cover of $\mathbf{K}^*$, and each separator of $\mathbf{K}$ is contained in some edge X_i of $\mathbf{H}$. Moreover, by properties (i) and (ii) of acyclic schemes, one has that: $\tilde{T}$ coincides with the ME universal table of the set of the marginals of $\tilde{T}$ with

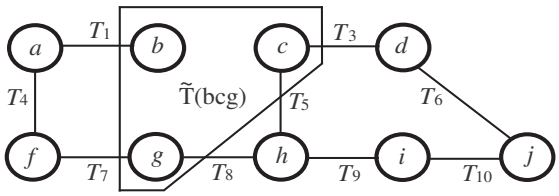


Figure 8. The projection on table set $\mathcal{T}^*$ onto W

respect to the edges of $\mathbf{K}$, and there is a tree formula for $\tilde{T}$ generated by $\mathbf{K}$. Finally, the marginal of $\tilde{T}$ with respect to each edge of $\mathbf{K}$ is the ME universal table of the corresponding projection of $\mathcal{T}^*$. Note that, for each j ($j = 1, \ldots, k$), the projections of $\mathcal{T}^*$ and $\mathcal{T}$ onto the edge C_j of $\mathbf{K}$ are the same (up to redundant tables) and, hence, their ME universal tables do coincide so that $\tilde{p}(C_j)$ can be computed by applying the IPFP procedure to $\mathcal{T}(C_j)$ with zero approximation $\frac{1}{size(C_j)}$. On the other hand, $\tilde{T}$ is collapsible onto the edge C of $\mathbf{K}$ [17] so that $\tilde{p}(C)$ can be computed by applying the IPFP procedure to $\mathcal{T}(C)$ with zero approximation $\frac{q(z)}{size(C-Z)}$.

Example 2 (*continued*). Recall that $Z = bcg$. The only edges C_j of $\mathbf{K}^*$ for which the set $Z \cap C_j$ is empty or is contained in some edge X_i of $\mathbf{H}$ are $cdhij$ and dej. Therefore, $C = abfg \cup bcg \cup cgh = abcfgh$ and $\mathbf{K} = \{abcfgh, cdhij, dej\}$. A join tree of $\mathbf{K}$ is shown in Figure 4 and the tree formula for $\tilde{T}$ generated $\mathbf{K}$ reads:

$$\frac{\tilde{p}(abcfgh)\tilde{p}(cdhij)\tilde{p}(dej)}{p_5(ch)p_6(dj)}$$

where the distribution $\tilde{p}(abcfgh)$ is computed by applying the IPFP procedure to $\mathcal{T}(abcfgh)$ with zero approximation $\frac{q(bcg)}{size(afh)}$, and the distributions $\tilde{p}(cdhij)$ and $\tilde{p}(dej)$ are computed in the same way as $\hat{p}(cdhij)$ and $\hat{p}(dej)$ (see above).

Turning to our problem of computing $\tilde{p}(v)$, it is sufficient to note that, since Z is contained in V, if we compute the closed hull of V in $\mathbf{K}$, it will be a superset of C. So, after detemining the closed hull of V in $\mathbf{K}$, we can compute $\tilde{p}(v)$ using the marginalization technique employed for $\hat{p}(v)$ (see above).

Example 2 (*continued*). The closed hull of $V = abcdgh$ in $\mathbf{K}$ is $W = abcdghij$. The tree formula for $\tilde{T}(W)$ generated $\mathbf{K}(W)$ reads:

$$\frac{\tilde{p}(abcfgh)\tilde{p}(cdhij)}{p_5(ch)}$$

where the distributions $\tilde{p}(abcfgh)$ and $\tilde{p}(cdhij)$ are computed as above. Instead of computing $\tilde{p}(w)$ using the tree formula above, we soon marginalize $\tilde{p}(abcfgh)$ with respect to $abcgh$ and $\tilde{p}(cdhij)$ with respect to cdh. Thus, we obtain the tree formula for $\tilde{T}(abcdgh)$:

$$\frac{\tilde{p}(abcgh)\tilde{p}(cdh)}{p_5(ch)},$$

After computing the distribution of $\tilde{T}(abcdgh)$ using the tree formula above, we can finally obtain $\tilde{p}(v)$ by marginalization.

5 Conclusions

We have considered the problem of estimating the answer to a table query using a table set on the target variable and a table on an auxiliary variable selected by the user. We have shown that such a query can be answered with "local" computation, that is, using a (hopefully minimal) subset of the table set on the target variable and applying the principle of "divide-and-conquer". A direction for future research is the generalization of this approach to the case that also the information on the auxiliary variable is stored in a table set.

APPENDIX

Section A

The *entropy* of a distribution $p(x)$ is the nonnegative functional

$$H[p] = -\sum_x p(x)\ log(p(x))$$

the summation being extended over all tuples x in the support of $p(x)$. It is well-known that $H[p]$ is always less than or equal to $log\ size(X)$. Given a distribution $\pi(x)$ whose support contains the support of $p(x)$, the *cross-entropy* (or "I-divergence"or "discrimination information" or "Kullback-Leibler distance") between $p(x)$ and $\pi(x)$ is the nonnegative functional

$$D[p, \pi] = \sum_x p(x)\ log\frac{p(x)}{\pi(x)}$$

the summation being extended over all tuples x in the support of $p(x)$. It is well-known that $D[p, \pi] = 0$ if and only if $p(x) = \pi(x)$. Let us assume that, for a subset Z of X,

$q(z)$ is a distribution whose support contains the support of $p(z)$. Then, for $\pi(x) = \dfrac{q(z)}{size(X-Z)}$, we have

$$D[p,\pi] = logsize(X-Z) - H[p] - \sum_z p(z)\ logq(z).$$

Finally, suppose that $p(x)$ is an extension of a consistent set of distributions $p_1(x_1), \ldots, p_n(x_n)$. If Z is a (possibly empty) subset of X_i for some i, then

$$\sum_z p(z)\ logq(z) = \sum_z p_i(z)\ logq(z) = const$$

and, hence, minimizing $D[p,\pi]$ is the same as maximizing $H[p]$.

Section B

A *hypergraph* with vertex set X is a nonempty collection $\mathbf{H}$ of nonempty subsets of X, which are called *edges* of $\mathbf{H}$ [4] and whose union recovers X. A *partial edge* of $\mathbf{H}$ is a nonempty set of vertices that is contained in some edge of $\mathbf{H}$. A *cover* of $\mathbf{H}$ is a hypergraph $\mathbf{K}$ with vertex set X such that each edge of $\mathbf{H}$ is a partial edge of $\mathbf{K}$. Let W be a nonempty subset of X. The *subhypergraph* of $\mathbf{H}$ *induced*by W, denoted by $\mathbf{H}(W)$, is the hypergraph with vertex set W, whose edges are exactly the maximal (with respect to set-inclusion) intersections of W with the edges of $\mathbf{H}$. A *path* is a sequence of egdes such that every two consecutive edges have a nonempty intersection. Two vertices are *connected* if they belong respectively to the first edge and to the last edge of a path. The *connected components* of a hypergraph are its subhypergraphs induced by maximal sets of pairwise-connected vertices. A hypergraph is *connected* if it has exactly one connected component. Two connected vertices are *separated* by a set S of vertices if neither belongs to S and they belong to distinct connected components of $\mathbf{H}(X-S)$. A partial edge S is a *separator* if there exist two vertices that are separated by S but are not separated by any proper subset of S. Let $\mathbf{H}$ be a connected hypergraph. The *intersection graph* of $\mathbf{H}$ is the ordinary graph whose nodes correspond one-to-one to and are labelled by the edges of $\mathbf{H}$, and two distinct nodes of G are joined by an arc if their labels have a nonempty intersection. Moreover, if (u,v) is an arc of G and A and B are the labels of the nodes u and v, then the arc (u,v) is labelled by $A \cap B$. A spanning tree J of G is a *join tree* of $\mathbf{H}$ if, for every two nodes of J, the intersection of its labels is contained in the label of each node along the (unique) path in J that connects the two nodes. The hypergraph $\mathbf{H}$ is acyclic if there exists a join tree of $\mathbf{H}$ [4]. If this is the case, then for every join tree J of $\mathbf{H}$, the separators of $\mathbf{H}$ are exactly the labels of arcs of J. Several other equivalent definitions of acyclicity exist [4].

References

[1] M. Bacharach. *Biproportional Matrices and Input-Output Change*. University Press, Cambridge, 1970.

[2] J.-H. Badsberg and F. M. Malvestuto. An implementation of the iterative proportional fitting procedure by propagation trees. *Computational Statistics and Data Analysis*, 37:297–322, 2001.

[3] S. F. Bishop and P. Holland. *Discrete Multivariate Analysis*. MIT-Press, 1975.

[4] D. M. C. Beeri, R. Fagin and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of ACM*, 30:479–513, 1983.

[5] H. V. J. C. Faloutsos and N. D. Sidiropoulos. Recovering information from summary data. *Proceedings of the* 23*rd VLDB Conference*, pages 36–45, 1997.

[6] S. Chauduri and U. Dayal. An overview of data warehousing and olap technology. *ACM SIGMOD Record*, 26:65–74, 1997.

[7] I. Csiszár. *I*-divergence geometry of probability distributions and minimization problems. *The Annals of Probability*, 3:146–158, 1975.

[8] W. E. Deming and F. F. Stephan. On a least square adjustment of a sampled frequency table when the expected marginal totals are known. *Annals of Mathematical Statistics*, 11:427–444, 1940.

[9] M. Ghosh and J. N. K. Rao. Small area estimation: An appraisal. *Journal of Statistical Science*, 9:55–93, 1994.

[10] W. W. Leontief and A. Strout. Multiregional input-output analysis. *Structural Interdependence and Economic Development. T. Bama (Ed.)*, pages 119–169, 1963.

[11] F. M. Malvestuto. Answering queries in categorical databases. *Proc. of the* 6*th ACM Symp. on Principles of Database Systems*, pages 87–96, 1987.

[12] F. M. Malvestuto. A universal table model for categorical databases. *Information Sciences*, 49:203–223, 1989.

[13] F. M. Malvestuto. A universal - scheme approach to statistical databases containing homogeneous summary tables. *ACM Trans. on Database Systems*, 18:678–708, 1993.

[14] F. M. Malvestuto. A hypergraph-theoretic analysis of collapsibility and decomposability for extended log-linear models. *Statistics and Computing*, 11:155–169, 2001.

[15] F. M. Malvestuto and M. Moscarini. A fast algorithm for query optimization in universal-relation databases. *J. Computer and System Sciences*, 56:299–309, 1998.

[16] F. M. Malvestuto and M. Moscarini. Decomposition of a hypergraph by partial-edge separators. *Theoretical Computer Science*, 237:57–79, 2000.

[17] F. M. Malvestuto and E. Pourabbas. Customized answers to summary queries via aggregate views. *Proceedings of the* 16*th SSDBM Conference*, pages 193–202, 2004.

[18] R. Stone and A. Brown. *A Computable Model for Economic Growth, A Programme for Growth No.1*. Chapman and Hall, London, 1962.

[19] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce hypergraphs. *SIAM J. on Computing*, 13:566–579, 1984.

Session 10: Spatial Queries

Probabilistic Data Modeling and Querying for Location-Based Data Warehouses

Igor Timko
Aalborg University

Curtis E. Dyreson
Washington State University

Torben Bach Pedersen
Aalborg University

Abstract

Motivated by the increasing need to handle complex, dynamic, uncertain multidimensional data in location-based warehouses, this paper proposes a novel probabilistic data model that can address the complexities of such data. The model provides a foundation for handling complex hierarchical and uncertain data, e.g., data from the location-based services domain such as transportation infrastructures and the attached static and dynamic content such as speed limits and vehicle positions. The paper also presents algebraic operators that support querying of such data. The work is motivated with a real-world case study, based on our collaboration with a leading Danish vendor of location-based services.

1. Introduction

Corporate and personal use of location-based services (LBSs), e.g., traffic or tourist related services, is increasing. LBSs generate massive amounts of location-based data that must be analyzed in order to optimize and personalize the services. Of particular interest are aggregation queries that involve the transportation infrastructure and attached content, e.g., "How many users (in their cars) of age less that 21 will be in the eastbound lane of Main Street five minutes from now?". Current OLAP and data warehouse (DW) technology [11, 13, 18] supports aggregation queries based on a *multidimensional* data model capturing hierarchies of dimensional data. Unlike other types of data models, multidimensional models provide first-class support for interactive, investigative aggregate queries on complex data, e.g., roll-up and drill-down queries [20]. This creates a need for location-based data warehouses (LBDWs) that can offer the benefits of traditional DW technology for LBS data. Current DW technology, both in research and industry, can provide support for many kinds of LBDW queries, but LBDWs have additional complexities that are not well-supported by traditional DWs such as uncertain data. For example, in a transportation infrastructure, cars are moving dynamically, so the future location of a car is uncertain. Moreover, the current location is sometimes also uncertain (e.g., known only to a wireless phone grid). Since the problem domain is very complex, a formal foundation for LBDWs is needed.

The contributions of this paper are as follows. First, the paper presents a probabilistic multidimensional data model (extension of the deterministic model from [26]) that can manage uncertain LBS data. The probabilities appear both in dimension hierarchies (a dimension value may *partially* contain another value) and fact characterizations (facts are characterized by dimension values with certain probabilities). Second, the paper presents a set of algebraic operators for querying the modeled uncertain data (extension of the operators from [20]). Third, the paper defines different types of probabilistic fact groupings for aggregation. Finally, the paper defines different types of probabilistic aggregation functions applied to the groups. The paper thus extends current OLAP/DW technology with means for supporting LBDWs. The concepts presented in the paper are illustrated using a real-world case study from the LBS domain. The work is based on an on-going collaboration with a leading Danish LBS vendor, Euman A/S [7].

Previous related work has generally fallen into three categories: probabilistic databases, spatio-temporal databases, and multidimensional OLAP databases. The work on probabilistic data management in general [1, 2, 6, 8] handles basic uncertainty in the data, but does not support dimensional data with hierarchies and LBS specifics such as transportation infrastructures and attached content. Research in general-purpose spatio-temporal data management considers "operational" queries on certain [19, 22, 24] or uncertain [4, 5, 28, 29] spatio-temporal data in 2D spaces or transportation infrastructures, but do not consider aggregation/analysis queries. Some papers [21, 25, 30, 31] have considered aggregation of spatial or spatio-temporal data, but have not considered transportation infrastructures.

Previous research has also covered the modeling of moving objects [9], transportation infrastructures [16, 17], or both [23], for "operational", i.e., non-analytical, purposes. The Euman data model [10] handles multiple infrastructure representation based on a *segment-based* model that is a generalization of the popular *linear referencing* technique [17]. However, none of this work captures data in a multidimensional framework, and thus does not provide optimal support for DW-like analytical querying, nor does it address the inherent uncertainties in LBS data.

Previous work on modeling multidimensional data, e.g., [20], does not handle the complexities of LBDWs. On the one hand, the data model and algebra presented in [12] support LBDW to a certain extent by allowing partial containment dimension hierarchies, while [26] improves on [12] by additionally handling transportation infrastructures and complex content. However, neither [12] nor [26] handles uncertainty in the data. On the other hand, a probabilistic multidimensional data

model [15] does handle uncertain data (by assigning a *single* probability to a *whole* row of a fact table) but does not support the other mentioned features of LBDWs. Moreover, our data model takes a more general and more flexible approach to handling uncertainty (a probability is assigned to *each* attribute of a fact table row).

The remainder of the paper is structured as follows. Section 2 presents the case study, and describes content and queries. Section 3 briefly introduces the model we use as the foundation, namely the [OLAP$_{LBS}$] model from [26]. Section 4 describes our approach to handling spatial hierarchies using expected degrees of containment, while Section 5 deals with probabilistic fact characterizations. Section 6 describes the formal query algebra. Finally, Section 7 concludes the paper and points to future work.

An extended version of this paper, which among other things contains a section on pre-aggregation issues, can be found in [27].

2. Case Study

We now discuss the requirements for an LBDW by presenting a real-world case study for which a UML diagram can be seen in Figure 1(a).

Content We start with discussing LBS content. LBDW have both *point* and *interval* content [10]. *Point content* concerns entities that are located at a specific geographic location, have no relevant spatial extent, and are attached to specific points in the transportation infrastructure, e.g., traffic accidents, gas stations, and (users' and other's) vehicle positions. *Interval content* concerns data that is considered to relate to a *road section* and is thus attached to intervals of given roads. Examples include speed limits and road surfaces. Our model must capture both point and interval content.

Content can be further classified as *dynamic* (frequently evolving) or *static* (rarely evolving). *Static* content, e.g., gas stations or speed limits, remains attached to a point or an interval of a road for a relatively long period of time. In this paper, we focus on very dynamic (*hyper-dynamic*) content, e.g., vehicle positions and their predicted trajectories (which evolve continuously). Positions of static content are usually certain, while positions of dynamic content are usually uncertain, e.g., a vehicle position is approximated by a wireless phone cell. Furthermore, any position prediction algorithm will have some degree of uncertainty. Thus, in our model, we must capture content of any degree of dynamism, as well as uncertainty.

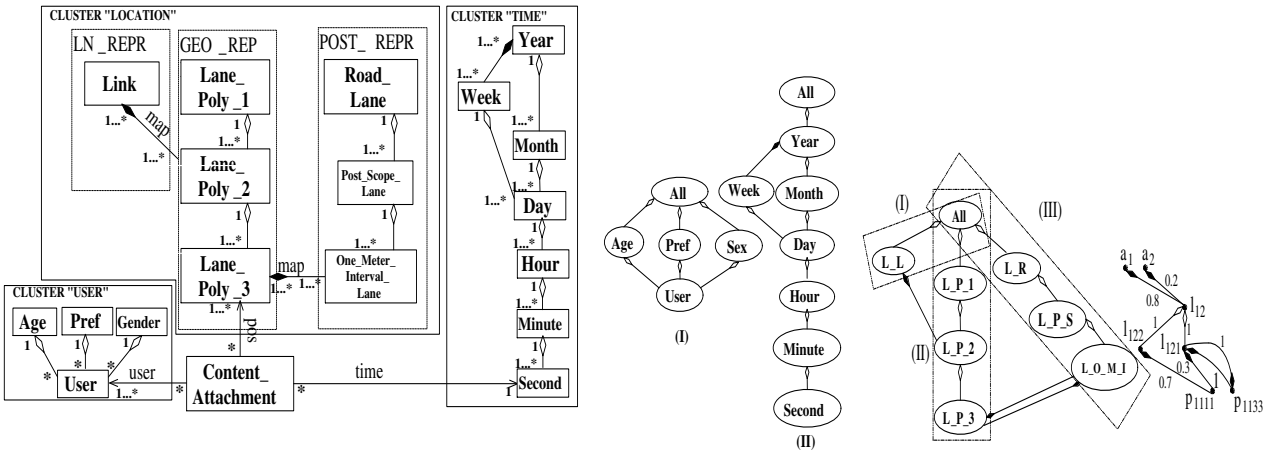
In Figure 1(a), hyper-dynamic content is modeled by the "USER" cluster, where the "User" class represents users and (implicitly) their vehicles, The "User" class participates in three *full containment relationships* capturing user age, preference, and gender. The users' (vehicle) positions in the infrastructure is modeled by the "LOCATION" cluster. The positions are captured at certain times, represented by the "TIME" cluster. This content positioning/attachment, is modeled as a "Content_Attachment" class which is linked to users, positions, and times. In an OLAP multidimensional model, the "Content_Attachment" class would be a *fact* characterized by "USER", "LOCATION", and "TIME" *dimensions*.

Transportation Infrastructure We now discuss the aspects of the transportation infrastructure relevant to data aggregation. Different, purpose-dedicated infrastructure representations, may be used, but most modern types of infrastructure representations, e.g., kilometer-post and geographic, are (1) *segment-based* and (2) *hierarchical* [10]. Thus, our data model must capture different types of segment-based and, possibly, hierarchical representations.

The "LOCATION" cluster from the UML diagram in Figure 1(a) contains three segment-based representations, "LN_REPR", "GEO_REPR", and "POST_REPR", which are link-node, geographic, and kilometer post representations, respectively. All three are a refinement of real-world representations used by the LBS company Euman A/S [7], obtained by representing *lanes* instead of *roads*. Often, lanes of the same road have different characteristics, e.g., different traffic density, so lanes must be captured separately [23]. We refer to segments that capture individual lanes, as *lane segments*. Lane segments may be further subdivided into subsegments to obtain more precise positioning (see "Content"). In the "LOCATION" cluster, each such lane segment level is a separate class. "LN_REPR" has only one level, "Link", that contain segments where the characteristics such as the speed limit remain constant. "POST_REPR" has three levels: 1) the "Road_Lane" class which captures particular lanes, e.g., a lane on an exit from a highway; 2) the "Post_Scope_Lane" class which captures segments between two kilometer posts, i.e, subdivisions of the road lanes above; 3) the "One_Meter_Interval_Lane" class which captures one-meter intervals (of the post scope segments above). The "GEO_REPR" also has three levels (but it could have more levels or less levels if needed). Here, a segment is a two-dimensional polyline representing (part of) a lane. Thus, a segment level is a geographical map. A sequence of segments from the "Lane_Poly_3" class (finest scale map), is approximated by (contained in) a segment from the "Lane_Poly_2" class (medium scale map), and similarly for "Lane_Poly_2" and "Lane_Poly_1" (coarsest scale map), see [10] for details. The levels define a hierarchy of *full containment (aggregation) relationships* between segments, which is denoted by empty rhombus-headed arrows in our model.

Finally, relationships between the representations must be captured, to allow content attached to one representation to be accessible from another. In the diagram, the relationships between the representations are modeled as "map" aggregations. Due to differences in how and from what data the representations are built, these mappings are *partial containment relationships*, i.e., segments from the class "Lane_Poly_3" *partially* contain (fully contain is a special case) segments from the "One_Meter_Interval_Lane" class. The "position" association captures attachments of user content to level-three segments of "GEO_REPR", making content mapped to "GEO_REPR" accessible from "POST_REPR". Further aspects such as road segments, traffic and traffic exchange directions, and lane change prohibitions are discussed in [26].

Time We now discuss the temporal characteristics of content. As mentioned above, content positions are captured at certain *time intervals*, which are organized in a containment hierarchy of temporal granularities, see the "TIME" cluster in



(a) (b) (c)

Figure 1. (a) Case Study, (b) dimension types $\mathcal{T}_u$ and $\mathcal{T}_t$, and (c) dimension type $\mathcal{T}_r$ and its instance

Figure 1(a). Our time hierarchy consists both of *full* and *partial* containment relationships between temporal granularities, e.g., the relationship between hours and days (weeks and years) is full (partial). User positions are linked to their time intervals by the "time" association.

Queries Analytical queries in LBS involve aggregations along multiple hierarchical dimensions, e.g., user content attachments will be aggregated along the USER, LOCATION, and TIME dimensions. As mentioned above, content positions may be given with some uncertainty, and we thus need to evaluate aggregate queries over uncertain information. Consider the four sample queries below, each concerning *point* content at a *current* or *future* time (the focus of this paper) and involving some kind of uncertainty. In this section, we give a prose description of the query, while in Example 6.8 we give a solution using the operators of our probabilistic model. The queries are: 1) as the *minimum expectation*, how many users of "age less than 21", a, are *possibly* in "the eastbound lane of Main Street", l_{ms}, at the current time, t?; 2) as the *average expectation*, what is an average age of the "male users", m, that will *possibly* be in "the second eastbound lane of I-90 highway between Moses Lake and Spokane", l_{90}, at the time "five minutes from now", $t?$; 3) as the *maximum expectation*, how many users whose locations will be known with a high degree of confidence, will pass through "Stadium Way's lane towards the campus", l, during the hour between 10AM and 11AM, $t?$; 4) (supposing some segments in "GEO_REPR" only partially contain one-meter interval segments in "POST_REPR"): as the *highest possible* value, what is the *maximum* age of the users that are *definitely* "between kilometer posts 45 and 46 of the eastbound lane of (Danish road) E45", l, at the current time, $t?$

All these queries aggregate probabilistic data with varying degrees of uncertainty at the current or a future time. They can all be formulated and evaluated in our framework, which improves the state-of-the-art by handling queries on DWs with probabilistic data.

3. The [OLAP$_{LBS}$] Model

We now briefly describe the data model from [26], which is the foundation for the probabilistic extension proposed in this paper. The model has constructs for defining both the *schema* (types) and the *data instances*. The schema of a cube is defined by a *fact schema* $\mathcal{S}$ that consists of a *fact type* $\mathcal{F}$ (cube name) and a set $\mathcal{D}$ of the *dimension types* $\mathcal{T}_i$ for each dimension.

A dimension type consists of a set $\mathcal{C}_\mathcal{T}$ of the *category types* $\mathcal{C}_j$ (dimension level types), a *relation* $\sqsubset_\mathcal{T}$ on $\mathcal{C}_\mathcal{T}$ specifying the hierarchical organization of the category types, and the special category types $\top_\mathcal{T}$ and $\bot_\mathcal{T}$ that denote the *top* and *bottom* category in the partial order, respectively. For example, a category type $\mathcal{C}$ may be used to model a level of *lane segments*. The transitive and irreflexive relation $\sqsubset_\mathcal{T}$, i.e., the partial order extended with equivalence, specifies the *partial* (including *full* as a special case) containment relationships among category types. The intuition is to specify whether members of a "child" category type have to be contained in a member of a "parent" category type *fully* or *partially*, e.g., segments from the same (different) representation(s). Next, a *subdimension type* of a dimension type is a set of its category types. Subdimension types of the same dimension type do not intersect except at $\top_\mathcal{T}$ category type. For example, a subdimension type is used to model a transportation infrastructure representation. The category types from the *same* (*different*) *subdimension type(s)* are related by full (partial) containment relationships.

Example 3.1. Figure 1(b) depicts dimension types $\mathcal{T}_u$ and $\mathcal{T}_t$. In addition, Figure 1(c) depicts a dimension type $\mathcal{T}_r$. The types capture the "USER", "TIME", and "LOCATION" clusters from Figure 1(a), respectively. Next, the type $\mathcal{T}_r$ has three subdimension types $\mathcal{T}_l$, $\mathcal{T}_g$, and $\mathcal{T}_p$, which capture "LN_REPR", "GEO_REPR", and "POST_REPR" , respectively. In the figure, the "boundary" of each type is a parralelogram and the types are labeled by (I), (II), and (III), respectively. In the figure, full (partial) containment category type relationships are given by empty (filled) rhombus-headed arrows. From these *direct* relationships we can deduce the *transitive* relationships between the category types. ∎

In the model *instances*, a *dimension* D consists of a set of *categories*. The *Type* function gives the corresponding type for dimensions and categories. A category C_j consists of a set of *dimension values* l_i. The transitive and irreflexive relation $\sqsubset$

on the union of all values, $\widehat{D}$, i.e., the partial order extended with equivalence, specifies the full or partial containment relationships of the values. For example, two values that model segments from the *same* (*different*) representation(s) are usually related by a full (partial) containment relationship. A special value $\top$ in each dimension *fully* contains every other value in the dimension.

Each relationship $l_1 \sqsubset l_2$ has an attached *degree of containment*, $d \in [0;1]$, written $l_1 \sqsubset_d l_2$. In a given dimension, the degrees have a unique interpretation, but different interpretations are possible. In the following definition, we present one such, *conservative*, interpretation (first introduced in [12]).

Definition 3.2. [**Safe degree of containment**] Given two dimension values l_1 and l_2 and a number $d \in [0;1]$, the notation $l_1 \sqsubset l_2 \wedge Deg_{saf}(l_1, l_2) = d$ (or $l_1 \sqsubset_d l_2$, for short) means that "l_2 contains at least $d \cdot 100\%$ of l_1". The special case of $d = 0$ means that "l_2 *may* contain l_1, and the size of contained part is unknown". We term d *safe degree of containment*. ∎

We abbreviate $l_1 \sqsubset_1 l_2 \wedge l_2 \sqsubset_1 l_1$ (equivalent values) by $l_1 \equiv l_2$. Transitive safe degrees are inferred according to the following rules. Given three dimension values, l_1, l_2, and l_3, and numbers $d_1 \in [0;1]$ and $d_2 \in [0;1)$:

1. *p-to-f transitivity:*
   $(l_1 \sqsubset_{d_1} l_2) \wedge (l_2 \sqsubset_1 l_3) \Rightarrow (l_1 \sqsubset_{d_1} l_3)$

   l_3 may contain the part of l_1 that is not in l_2, but the conditions of the rule do not give us information on this. We infer only what we can guarantee: what is contained in l_2 is *also* contained in l_3.
2. *p-to-p transitivity:*
   $(l_1 \sqsubset_{d_1} l_2) \wedge (l_2 \sqsubset_{d_2} l_3) \Rightarrow (l_1 \sqsubset_0 l_3)$

   If l_1 is fully or partially contained in l_2 and l_2 is partially contained in l_3 then we can only infer that at least "nothing" of l_1 is contained in l_3. In other words, we infer that l_1 *may* be contained in l_3.

Finally, *subdimension* is an instance of a subdimension type.

Example 3.3. Suppose we are given subdimensions D_l, D_g, and D_p of the subdimension types $\mathcal{T}_l$, $\mathcal{T}_g$, and $\mathcal{T}_p$, respectively. Parts of the subdimensions are depicted in Figure 1(c). In the following, we show how to infer transitive partial containment relationships with *safe* degrees. In the subdimension D_g, we have values $l_{121} \in C_{L\_P\_3}$ and $l_{12} \in C_{L\_P\_2}$ such that $l_{121} \sqsubset_1 l_{12}$. Then, in the subdimension D_p, we have a value $p_{1111} \in C_{L\_O\_M\_I}$ such that $p_{1111} \sqsubset_{0.3} l_{121}$. Consequently, we infer that $p_{1111} \sqsubset_{0.3} l_{12}$. Again, we have $l_{121} \sqsubset_1 l_{12}$. Then, in the subdimension D_l, we have value $a_1 \in C_{L\_L}$ such that $l_{12} \sqsubset_{0.8} a_1$. Consequently, we infer that $l_{121} \sqsubset_0 a_1$, which means that l_{121} *may* be contained in a_1. Next, in the subdimension D_p, we have a value p_{1133} such that $p_{1133} \equiv l_{121}$. Then we have already inferred that $l_{121} \sqsubset_0 a_1$. Consequently, we infer that $p_{1133} \sqsubset_0 a_1$. ∎

A *multidimensional object* (cube) consists of a set of *facts* F that are mapped to each dimension, D_j, with a *fact-dimension relation*, $R_j \subseteq F \times D_j$. For a fact $f \in F$ and a dimension value $l \in D_j$, we define (1) a *covering fact-dimension relationship* $(f, l) \in R_j^c \subseteq R_j$, which is read as "$f$ *covers* l", and (2) an *inside fact-dimension relationship* $(f, l) \in R_j^i \subseteq R_j$, which is read as "$f$ *is inside* l". Thus, the full set of fact-dimension relationships is $R_j = R_j^c \cup R_j^i$. Next, we define three kinds of *fact characterizations*, or inferred fact-dimension relationships, written $f \rightsquigarrow^c l$, $f \rightsquigarrow^i l$, and $f \rightsquigarrow_m^i l$. The semantics of the first two characterizations coincides with that of the corresponding fact-dimension relationships. The third characterization means that f *may be inside* l.

4. Expected Degrees of Containment

In this section, we introduce a new interpretation for degrees of containment. The motivation for the new interpretation is as follows. Assume that we are given a dimension D with its set of categories and the relation on its dimension values $\sqsubset$. As mentioned in Section 3, with the *safe* degrees of containment, the notation $l_1 \sqsubset_d l_2$, where $l_1 \in \widehat{D}$, $l_2 \in \widehat{D}$, and $d \in [0;1]$ means that the value l_2 contains *at least* $d \cdot 100\%$ of the value l_1. The disadvantage of this approach is that inferred, transitive relationships between dimension values are very likely to receive a degree equal to 0, because we infer only those degrees that we can guarantee, see Example 3.3. This makes the data too uncertain for practical use.

In order to make the transitive relationships more useful, we introduce the *expected* degrees of containment. Our approach is based on probability theory [3]. We consider each dimension value as an infinite set of points. We deal with the probabilistic events of the form "a value l_1 is contained in a value l_2", which is equivalent to "any point in l_1 is contained in l_2".

Definition 4.1. [**Expected degree of containment**] Given two dimension values l_1 and l_2 and a number $d \in [0;1]$, the notation $l_1 \sqsubset l_2 \wedge Deg_{exp}(l_1, l_2) = d$ (or $l_1 \sqsubset_d l_2$, for short) means that "l_2 is expected to contain $d \cdot 100\%$ of l_1", or, more formally, "l_1 is contained in l_2 with a probability of d". We term d the *expected degree of containment*. ∎

The formal definition is particularly useful for reasoning about transitivity of partial containment and fact characterizations. The rule of *transitivity of partial containment with expected degrees* is as follows. Given the values $l_1, l_2, \ldots, l_n$ from the category C and given the values l and l', the following holds:

$$(\bigwedge_{i=1}^{n} l \sqsubset_{d_i} l_i \wedge l_i \sqsubset_{d_i'} l' \Rightarrow l \sqsubset l' \wedge Deg_{exp}(l, l') = \sum_{i=1}^{n} d_i \cdot d_i')$$

The idea behind the rule is explained next. We will use notation $P(e)$ for the probability of the event e. Let us first consider a special case of the rule, when $i = 1$, i.e., when there is only one, unique path between values l and l'. Then, the rule takes the following form:

$$\forall (l_1, l_2, l_3) \in \widehat{D} \times \widehat{D} \times \widehat{D} (l_1 \sqsubset_{d_1} l_2 \wedge l_2 \sqsubset_{d_2} l_3 \Rightarrow l_1 \sqsubset_{d_1 \cdot d_2} l_3)$$

First, $l_1 \sqsubset_{d_1} l_2$ means that $P(e_1) = d_1$, where e_1 is "l_1 is contained in l_2". Second, $l_2 \sqsubset_{d_2} l_3$ means that $P(e_2) = d_2$, where e_2 is "l_2 is contained in l_3". The conjunction of these

two events, $e_1 \wedge e_2$, i.e., "l_1 is contained in l_3" is equivalent to $l_1 \sqsubset l_3$. Next, having assumed that the events e_1 and e_2 are independent, $P(e_1 \wedge e_2) = d_1 \cdot d_2$. This means that we have inferred the relationship $l_1 \sqsubset_{d_1 \cdot d_2} l_3$.

The general case of the rule allows n paths between l and l'. The ith path goes through a value l_i. Then, the event e, i.e., "l is contained in l'" is a disjunction of n disjoint events, $\bigvee_{i=1}^{n} e_i$, where e_i is "l is contained in l', given the ith path". The events $e_1, e_2, \ldots e_n$ are disjoint, because (1) we assume that values from the same category, in particular, the values $l_1, l_2, \ldots l_n$ do not overlap, and (2) consequently the n events "l is contained in l_i" are disjoint. Thus, the general case of the rule is n applications of the rule's special case. The ith application concerns an ith path and infers the probability $d_i \cdot d'_i$ of the event e_i. This means that the event e has the probability of $d = \sum_{i=1}^{n} d_i \cdot d'_i$, i.e., that there is a relationship $l \sqsubset_d l'$.

In order to produce the correct aggregates, i.e., to perform *correct aggregation*, a warehouse must consider all relevant aggregation paths between the source and destination category. Since no aggregation path is ignored during inferences of transitive partial containment relationships with expected degrees, the rule offers support for *correct aggregation*, which is missing from the analogous rule with safe degrees. A further support is offered by the rules for inferring fact characterizations (see Section 5).

Example 4.2. Continuing Example 3.3, we show how to infer transitive partial containment relationships with *expected* degrees. First, we demonstrate the support for correct aggregation. In the subdimension D_g, we have values $l_{121} \in C_{L\_P\_3}$, $l_{122} \in C_{L\_P\_3}$ and $l_{12} \in C_{L\_P\_2}$ such that $l_{121} \sqsubset_1 l_{12}$ and $l_{122} \sqsubset_1 l_{12}$. Then, in the subdimension D_p, we have a value $p_{1111} \in C_{L\_O\_M\_I}$ such that $p_{1111} \sqsubset_{0.3} l_{121}$ and $p_{1111} \sqsubset_{0.7} l_{122}$. In other words, we have two aggregation paths between values p_{1111} and l_{12}. Consequently, we "sum up" the paths, i.e., infer that $p_{1111} \sqsubset_{0.3 \cdot 1 + 0.7 \cdot 1 = 1} l_{12}$. Second, we demonstrate the improvement in certainty of transitive relationships, compared to those obtained by the rule with *safe* degrees. Then, in the subdimension D_l, we have value $a_1 \in C_{L\_L}$ such that $l_{12} \sqsubset_{0.8} a_1$. Consequently, we infer that $l_{121} \sqsubset_{1 \cdot 0.8 = 0.8} a_1$. Note that the last relationship would have received a (much lower) *safe* degree of 0. ∎

5. Probabilistic Fact Characterizations

In this section, we introduce a new kind of fact characterizations. The motivation for this is as follows. Assume that we are given a dimension, D, with its set of categories and the relation on its dimension values, $\sqsubset$. Recall content attachments from the case study in Section 2. Such attachments record that a *user* is in a specific *location* at a given *time*. The fact characterizations described in Section 3 allow us to express positions of *static* content, which are usually certain. However, positions of *dynamic* content are usually uncertain. For example, a user location may be given by a wireless phone cell, which only approximately locates the user. Furthermore, a practical prediction algorithm would predict future user locations with some degree of uncertainty. In addition to this location uncertainty, we may also have user and time uncertainty. For example, we may be certain about a location, but uncertain about what user is at that position or we may not know the time. In order to capture these possibilities, we generalize the notion of fact characterization by defining a *probabilistic fact characterization*.

Our approach is based on probability theory [3]. We consider the probabilistic events of the form "a fact f covers (is inside) a value l". We extend the definitions from Section 3 as follows.

Definition 5.1. [**Probabilistic fact-dimension relationships**] For a fact $f \in F$ and a dimension value $l \in \widehat{D}$, we define:

1. a *probabilistic covering fact-dimension relationship*, $(f, l, p_{min}, p_{max}) \in R^{c,p} \subseteq R$, which is read as "$f$ *covers* l *with probability of at least* p_{min} and *of at most* p_{max}", and
2. a *probabilistic inside fact-dimension relationship*, $(f, l, p_{min}, p_{max}) \in R^{i,p} \subseteq R$, which is read as "$f$ *is inside* l *with a probability of at least* p_{min} and *of at most* p_{max}".

$R = R^{c,p} \cup R^{i,p}$ is the full set of fact-dimension relationships. ∎

Consider an inside fact-dimension relationship, $(f, l, p_{min}, p_{max}) \in R^{i,p}$, p_{min} is a *lower* bound on the "true" probability of the relationship. Moreover, for a fact, f, and a category, C, any two events "f is inside l_1" and "f is inside l_2", where $l_1 \in C$ and $l_2 \in C$, are disjoint. For this reason, in an MO, we impose the following restriction on minimum probabilities: for any category, C, and any fact, f and given the restriction of $R^{i,p}$ on C and f, $R^{i,p}_{|C,f} = \{(f', l, p^l_{min}, p^l_{max}) | l \in C \wedge f' = f\}$, we require that $\sum_{l \in C} p^l_{min} \leq 1$. However, p_{max} is a *higher* bound on the "true" probability of the relationship (f, l, p_{min}, p_{max}). For this reason, we do not impose an analogous restriction on maximum probabilities.

The exact probabilities of fact-dimension relationships may also be expressed. For example, the statement "f covers l with probability p" is expressed as $(f, l, p, p) \in R^p_c$. The deterministic fact-dimension relationships are a special case of the probabilistic fact-dimension relationships, i.e., $(f, l) \in R_c$ is expressed as $(f, l, 1, 1) \in R^p_c$ and $(f, l) \in R_i$ is expressed as $(f, l, 1, 1) \in R^p_i$.

Next, we define two new kinds of fact characterizations, written $f \rightsquigarrow^c_{[p_{min}; p_{max}]} l$ and $f \rightsquigarrow^i_{[p_{min}; p_{max}]} l$. The non-probabilistic fact characterizations are a special case of the probabilistic characterizations, i.e., (1) $f \rightsquigarrow^c l$ is expressed as $f \rightsquigarrow^c_{[1;1]} l$, which is also read as "f *covers* l *for sure*", (2) $f \rightsquigarrow^i l$ is expressed as $f \rightsquigarrow^i_{[1;1]} l$, which is also read as "f *is inside* l *for sure*", and (3) $f \rightsquigarrow^i_m l$ is expressed as $f \rightsquigarrow^i_{[0;1]} l$, which is also read as f *is inside* l *with unknown probability*. In addition, $f \rightsquigarrow^c_{[0;1]} l$ is also read as f *covers* l *with unknown probability*.

The set R is stored in the data warehouse and the probabilistic fact characterizations are inferred when needed. For the inference, the warehouse uses the rules described in Sections 5.1 and 5.2. In essence, the rules provide a recursive definition of the notion of probabilistic fact characterization. Since

the rules are valid both with the *safe* and *expected* degrees of containment, the notation used in the rules does not reflect the kind of degrees.

5.1. Basic Rules

In the following, we present the basic rules for inferring fact characterizations.

$\forall(f, l) \in F \times \widehat{D}$

1. $((f, l, p_{min}, p_{max}) \in R_c^p \Rightarrow f \rightsquigarrow^c_{[p_{min};p_{max}]} l)$
2. $((f, l, p_{min}, p_{max}) \in R_i^p \Rightarrow f \rightsquigarrow^i_{[p_{min};p_{max}]} l)$

   If a fact f is attached to and covers (is inside) a segment l with the probability of at least p_{min} and of at most p_{max} , then we can infer that f covers (is inside) l with the probability of at least p_{min} and of at most p_{max}.
3. $(f \rightsquigarrow^c_{[p_{min};p_{max}]} l \Rightarrow f \rightsquigarrow^i_{[p_{min};p_{max}]} l)$

   If a fact f *covers* a segment l with the probability of at least p_{min} and of at most p_{max}, then f is also *inside* l with the same probability. The idea behind the rule is as follows. If a piece of content *covers* a segment with some probability, i.e., between p_{min} and p_{max}, then it is possible to state that the piece is also *inside* that segment with the same or even greater probabilities. However, the data at hand, i.e., the probabilities that we can use for arguing, only allows us to record the lowest possible probabilities, i.e., those between p_{min} and p_{max}.

$\forall(f, l_1, l_2, \ldots, l_n, l) \in F \times D \times \ldots \times D$

4. $(l_1 \equiv l_2 \wedge f \rightsquigarrow^i_{[p_{min};p_{max}]} l_1 \Rightarrow f \rightsquigarrow^i_{[p_{min};p_{max}]} l_2)$
5. $(l_1 \equiv l_2 \wedge f \rightsquigarrow^c_{[p_{min};p_{max}]} l_1 \Rightarrow f \rightsquigarrow^c_{[p_{min};p_{max}]} l_2)$

   If two values are equivalent, then they characterize the same facts in the same way.

5.2. The Characterization Sum Rule

In the following, we present the most important rule for inferring fact characterizations, called the *characterization sum rule*. Among other things, the rule provides support for *correct aggregation*.

Definition 5.2. [**Characterization sum rule**] For any fact, f, and dimension values, $l_1, l_2, \ldots, l_n$, and l, the following holds:

$$(\bigwedge_{i=1}^{n} (l_i \sqsubset_{d_i} l \wedge f \rightsquigarrow^i_{[p^i_{min};p^i_{max}]} l_i) \Rightarrow f \rightsquigarrow^i_{[p_{min};p_{max}]} l)),$$

where $p_{min} = \sum_{i=1}^{n} d_i \cdot p^i_{min}$ and
$p_{max} = min(\sum_{i=1}^{n} d_i \cdot p^i_{max}, 1)$ ∎

The basic idea behind the rule is that we obtain the probability for a fact characterization by summing up probabilities for that fact characterization obtained through n different aggregation paths. A more formal explanation is as follows. We use the notation $P(e)$ for the probability of the event e and $P(e \wedge e')$ for the conjunction of the events e and e'. First, let the event e_1 be "a piece of content is inside a segment l", i.e., is "$f \rightsquigarrow^i l$". We need to compute p_{min}, which is a lower bound on $P(e_1)$. The event e_1 is a disjunction of n disjoint events $e^i_2 \wedge e^i_3$, where e^i_2 is "$f \rightsquigarrow^i l_i$" and e^i_3 is "$l_i \sqsubset l$". The events $e^1_2 \wedge e^1_3, e^2_2 \wedge e^2_3, \ldots, e^n_2 \wedge e^n_3$ are disjoint, because (as with transitivity of containment) we assume that values from the same category, in particular, the values $l_1, l_2, \ldots l_n$ do not overlap. For this reason, $P(e_1) = \sum_{i=1}^{n} P(e^i_2 \wedge e^i_3)$. Since events e^i_2 and e^i_3 are independent, $P(e^i_2 \wedge e^i_3) = P(e^i_2) \cdot P(e^i_3)$. Next, $P(e^i_2) \geq p^i_{min}$ and $P(e^i_3) \geq d_i$ or $P(e^i_3) = d_i$, if d_i is an expected or safe degree, respectively. This means that $P(e_1) \geq \sum_{i=1}^{n} d_i \cdot p^i_{min}$. So, $p_{min} = \sum_{i=1}^{n} d_i \cdot p^i_{min}$. The case of p_{max}, i.e., the maximum probability that a piece of content is inside a segment l, is analogous. However, since in this case we sum *upper* bounds on probabilities, the resulting upper bound, p_{max}, may be higher than 1. Since according to probability theory the maximum probability of any event is 1, we "cut" p_{max} down to 1.

Since no aggregation path is ignored in the process of inferring fact characterizations, the characterization sum rule offers significant support for *correct aggregation*. Furthermore, if expected degrees are used for constructing an MO, the rule for inferring transitive relationships between dimension values (see Section 4) provides additional support. In particular, combined effect of these two rules is that a query engine may perform inferences on an MO in any order without losing any information, i.e., transitive relationships between values first, then fact characterizations, or in the reverse order.

Example 5.3. Given a dimension hierarchy from Figure 1(c), we exemplify the use of the characterization sum rule. Suppose our data warehouse has data on (uncertain) positions of a user in the kilometer-post representation, which are stored as $(f_1, p_{1111}, 0, 0.1) \in R^p_i$ and $(f_1, p_{1133}, 0.9, 1) \in R^p_i$. Then, the positions of the user in the link-node representation are deduced as follows. First, assuming that the degrees from Figure 1(c) are *expected degrees*, we infer the relationships $p_{1111} \sqsubset_{0.8} a_1$, $p_{1111} \sqsubset_{0.2} a_2$, $p_{1133} \sqsubset_{0.8} a_1$, and $p_{1133} \sqsubset_{0.2} a_2$. Second, by basic rule 2, we obtain the fact characterizations $f_1 \rightsquigarrow^i_{[0;0.1]} p_{1111}$ and $f_1 \rightsquigarrow^i_{[0.9;1]} p_{1133}$. Finally, by the characterization sum rule, we infer $f_1 \rightsquigarrow^i_{[p^1_{min};p^1_{max}]} a_1$ and $f_1 \rightsquigarrow^i_{[p^2_{min};p^2_{max}]} a_2$, where $p^1_{min} = 0.8 \cdot 0 + 0.8 \cdot 0.9 = 0.72$, $p^1_{max} = 0.8 \cdot 0.1 + 0.8 \cdot 1 = 0.88$, $p^2_{min} = 0.2 \cdot 0 + 0.2 \cdot 0.9 = 0.18$, and $p^2_{max} = 0.2 \cdot 0.1 + 0.2 \cdot 1 = 0.22$. ∎

Note that Definition 5.1 allows two levels of uncertainty in fact-dimension relationships. At the first level, we express *uncertainty about content attachments* by relating the same fact, f, to several dimension values. At the second level, we express *uncertainty about probabilities of content attachments* by specifiying the lower and upper bounds of the probabilities. The second level provides the model with additional flexibility: (1) it makes more sense to map the user's intuitive understanding of uncertainty to intervals, rather than precise numbers, e.g., "very low probability" means a whole range of uncertainty and is more accurately represented by $[0.1; 0.3]$ than by 0.2; (2) using probability bounds when inferring fact characterizations allows having simple inference rules.

6. The Algebra

In this section, we present a set of algebraic operators, i.e., algebra, that is a formal foundation for querying the data captured by our model. The operators allow us to formulate queries for probabilistic fact-dimension characterizations. Due to space constraints, only *selection* and *aggregate formation* operator are discussed in detail, while others, i.e., *union*, *projection*, *join*, and *difference* are discussed briefly.

We base our algebra on the *deterministic* algebra from [20], which is proven to be at least as powerful as the relational algebra with aggregation functions [14]. The operators from [20] need to be extended to handle probabilistic aspects of our model. Intuitively, after the extension, our algebra will be at least as powerful as a probabilistic relational algebra (e.g., from [1]).

Let i range from 1 to n. For unary operators, we assume a single n-dimensional MO $M = \{\mathcal{S}, F, D_M, R_M\}$, where $D_M = \{D_i\}$ and $R_M = \{R_i\}$. For binary operators, we assume two n-dimensional MO's $M_j = (\mathcal{S}_j, F_j, D_{M_j}, R_{M_j})$, $j = 1, 2$, where $D_{M_j} = \{D_i^j\}$ and $R_{M_j} = \{R_i^j\}$.

6.1. Selection Operator

The selection operator is used to select a subset of the facts in an MO based on a predicate. Let $K = \{i, c\}$, where the symbols i and c stand for "inside" and "covering", respectively, and let $I = [0; 1]$. The selection operator, σ, uses a predicate $q : \widehat{D}_1 \times \ldots \times \widehat{D}_n \times (I \times I)^n \times K^n \rightarrowtail \{true, false\}$. Thus, the parameters of the predicate q are n dimension values, each from a different dimension, n intervals of probability values, and n "inside" or "covering" symbols. The resulting set of facts is:

$$
\begin{aligned}
F' = \{f \in F \mid\ & \exists(l_1, \ldots, l_n) \in \widehat{D}_1 \times \ldots \times \widehat{D}_n \\
& (\exists([p^1_{min}; p^1_{max}], \ldots, [p^n_{min}; p^n_{max}]) \in (I \times I)^n \\
& \quad (\exists(k_1, \ldots, k_n) \in K^n \\
& \quad\quad (q(l_1, [p^1_{min}; p^1_{max}], k_1, \ldots, l_n, [p^n_{min}; p^n_{max}], k_n) \\
& \quad\quad \wedge \bigwedge_{j=1}^{n} f \rightsquigarrow^{k_j}_{[p^j_{min}; p^j_{max}]} l_j\}
\end{aligned}
$$

We thus restrict the set of facts to those that are characterized by dimension values where q evaluates to $true$. This operator supports probabilistic covering/inside fact characterizations. Specifically, the operator allows us to formulate queries that select facts that are characterized (1) with given intervals of uncertainty, i.e., $[p^i_{min}; p^i_{max}]$ for a characterization by the dimension D_i, and (2) kind of characterization, i.e., inside, covering, or both by means of k_i for a characterization by the dimension D_i. In addition, we restrict the fact-dimension relations accordingly, while the dimensions and the fact schema stay the same.

Example 6.1. [**Selection operator**] Continuing Example 5.3, suppose that we would like to select *reliable* data on male users, $m \in C_{Gender}$, on a link, $a_1 \in C_{L\_L}$, at a future time, $t \in C_{Second}$. For this, the predicate q could be defined as follows:

$$
\begin{aligned}
& q(l_1, [p^1_{min}; p^1_{max}], k_1, l_2, [p^2_{min}; p^2_{max}], k_2, \\
& \quad l_3, [p^3_{min}; p^3_{max}], k_3) = true \Leftrightarrow \\
& (l_1 = m \wedge p^1_{min} = p^1_{max} = 1 \wedge k_1 = i) \wedge \\
& (l_2 = a_1 \wedge [p^2_{min}; p^2_{max}] \subseteq [0.5; 1] \wedge k_2 = i \wedge \\
& (l_3 = t \wedge p^3_{min} = p^3_{max} = 1 \wedge k_3 = i)
\end{aligned}
$$

The predicate defines the *reliable* data as the fact characterizations such as: (1) in the USER and TIME dimension, the minimum and maximum probability equals to 1, (2) in the LOCATION dimension, the minimum (maximum) probability is at least 0.5 (any).

Suppose we have characterizations $f_1 \rightsquigarrow^i_{[1;1]} m$ and $f_1 \rightsquigarrow^i_{[1;1]} t$ in the USER and TIME dimension, respectively. Since we have inferred the characterization $f_1 \rightsquigarrow^i_{[0.72;0.88]} a_1$, the fact f_1 would contribute to the result, i.e. $f_1 \in F'$. However, if we replace a_1 with a_2 in the query, then the fact f_1 would be outside the result, because of the characterization $f_1 \rightsquigarrow^i_{[0.18;0.22]} a_2$. As another example, we could select all data that is *unreliable* with respect to positioning, for instance, to remove it from a subsequent computation, as follows:

$$q(\ldots) = true \Leftrightarrow [p^2_{min}; p^2_{max}] \subseteq [0; 0.5)$$

6.2. Aggregate Formation Operator

The unary aggregate formation operator is used when applying aggregate functions to an MO. We assume a set of traditional aggregation functions, $H = \bigcup_{i=1}^{n} \{SUM_i, AVG_i, MIN_i, MAX_i\} \cup \{COUNT\}$. The $COUNT$ works by considering fact-dimension relationships for all dimensions, while other functions "look up" the required data for the facts in the relevant fact-dimension relation. For example, SUM_1 finds its data in the fact-dimension relation R_1 and sums them.

In addition, the operator $Group : D_1 \times \ldots \times D_n \rightarrowtail 2^F$ is defined. In general, the operator groups the facts characterized by the same dimension values, i.e., $Group(l_1, \ldots, l_n) = \{f \in F \mid f \rightsquigarrow l_1 \wedge \ldots \wedge f \rightsquigarrow l_n\}$. Later in this section, we present more elaborate definitions of the grouping operator.

Aggregate Formation Operator Definition Next, we restate a generic definition of the aggregate formation operator from [12], which is also suitable in our context of uncertain data. In the definition, we denote $(l_1, \ldots, l_n)$ and $Group(l_1, \ldots, l_n)$ by $\vec{l}$ and G, respetively. Also, we assume that $\vec{l} \in C_1 \times \ldots \times C_n$.

Definition 6.2. [**Aggregate formation operator**] Given a new (result) dimension D_{n+1} of a new (result) type $\mathcal{T}_{n+1}$, an aggregation function $h : 2^F \rightarrowtail D_{n+1}$ from the set H, and a set of grouping categories $\{C_i \in D_i, i = 1, \ldots n\}$, the *aggregate formation operator*, α, is defined as follows: $M' = \alpha[D_{n+1}, h, C_1, \ldots, C_n](M) = (\mathcal{S}', F', D'_{M'}, R'_{M'})$, where $\mathcal{S}' = (\mathcal{F}', \mathcal{D}')$ and

$\mathcal{F}' = 2^{\mathcal{F}}, \mathcal{D}' = \{\mathcal{T}'_i, i = 1, \ldots, n\} \cup \{\mathcal{T}_{n+1}\},$
$\mathcal{T}'_i = (\mathcal{C}'_i, \sqsubset'_{\mathcal{T}_i}, \bot'_{\mathcal{T}_i}, \top'_{\mathcal{T}_i}),$
$\mathcal{C}'_i = \{\mathcal{C}_{ij} \in \mathcal{T}_i \mid \mathcal{C}_i \sqsubset_{\mathcal{T}_i} \mathcal{C}_{ij}\} \cup \{\mathcal{C}_i\},$
$\sqsubset'_{\mathcal{T}_i} = \sqsubset_{\mathcal{T}_i|_{\mathcal{C}'_i}}, \bot'_{\mathcal{T}_i} = \mathcal{C}_i, \top'_{\mathcal{T}_i} = \top_{\mathcal{T}_i}, F' = \{G \neq \emptyset\},$
$D' = \{D'_i, i = 1, \ldots, n\} \cup \{D_{n+1}\}, D'_i = (C'_{D'_i}, \sqsubset'_{D'_i}),$
$C'_{D'_i} = \{C'_{ij} \in D_i \mid \mathcal{C}'_{ij} \in \mathcal{C}'_i\}, \sqsubset'_{D'_i} = \sqsubset_{D_i|_{D'_i}}$
$R'_{M'} = \{R'_i, i = 1, \ldots, n\} \cup \{R'_{n+1}\},$
$R'_i = \{(f', l_i) \mid \exists \vec{l}(f' = G\}, R'_{n+1} = \bigcup_{\vec{l}} \{(G \neq \emptyset, h(G)\}$ ∎

Thus, for every combination of dimension values $\vec{l} = (l_1, \ldots, l_n)$ in the given grouping categories, the aggregation function h is applied to the set of facts characterized by $\vec{l}$, i.e., to the group $G = Group(\vec{l})$, and the result is placed in the new dimension D_{n+1}.

The new facts from the set F' are of type $\mathcal{F}'$, which denotes *sets of the argument fact type*, and the resulting dimensions types from $\mathcal{D}'$ are obtained by restricting argument dimension types to the category types that are greater than or equal to the types of the grouping categories. The new dimension type $\mathcal{T}_{n+1}$ for the result is added to the set of dimension types.

The new set of facts F' consists of *sets* of the original facts, where original facts in a set share a combination of characterizing dimension values. The argument dimensions are restricted to the remaining category types, and the result dimension D_{n+1} is added. The fact-dimension relations for the argument dimensions now link sets of facts directly to their corresponding combination of dimension values, and the fact-dimension relation R'_{n+1} for the result dimension links sets of facts to the function results for these sets.

Grouping In Section 5, we introduced probabilistic fact characterizations, which allows us to group facts with an arbitrary *degree of confidence*, i.e., with arbitrary requirements to the probabilities of the characterizations of the grouped facts. Next, we define different kinds of grouping, considering *inside* fact characterizations only. The cases of *covering* characterizations are analogous.

Definition 6.3. [**Grouping operators**] We define the following *grouping operators*.

1. *Degree-of-confidence grouping operator*, $Group_d$:
$Group_d(l_1, \ldots, l_n, [p^1_{min'}, p^1_{max'}] \ldots, [p^n_{min'}, p^n_{max'}]) =$
$\{f \in F \mid \bigwedge_{k=1}^{n} f \rightsquigarrow^i_{[p^k_{min}; p^k_{max}]} l_k \wedge$
$[p^k_{min}; p^k_{max}] \subseteq [p^k_{min'}; p^k_{max'}]\}$
2. *Conservative grouping operator*, $Group_c$:
$Group_c(l_1, \ldots, l_n) = Group_d(l_1, \ldots l_n, [1; 1], \ldots, [1; 1])$
3. *Liberal grouping operator*, $Group_l$:
$Group_l(l_1, \ldots, l_n) = Group_d(l_1, \ldots l_n, [0; 1], \ldots, [0; 1])$ ∎

In the *degree-of-confidence grouping*, a group is formed from the facts that belong to the group with a probability given by the parameters of $Group_d$ operator.

We define the following special cases of the operator. First, in the *conservative grouping*, a group is formed from the facts that *definitely* belong to the group. Since only precise data will be used in calculations and the remaining data discarded, this kind of grouping is useful for computing a "lower bound" for a query result, i.e., the result contains as little data as possible.

Second, in *liberal grouping*, a group is formed from the facts that *possibly* belong to the group. Liberal grouping can be used for computing an "upper bound" for a query result, i.e., the result contains as much data as possible, because all the data, both precise and imprecise, are taken into consideration. This means that our definition of conservative and liberal grouping corresponds to the general understanding of the terms introduced in [20].

Example 6.4. Continuing Example 5.3, suppose we also have a fact f_2 characterized as follows: $f_2 \rightsquigarrow^i_{[1;1]} m$, $f_2 \rightsquigarrow^i_{[1;1]} a_1$, and $f_2 \rightsquigarrow^i_{[1;1]} t$. Then, suppose we wish to aggregate the *certain* data to the level of C_{Gender}, C_{L_L}, and C_{Second}, and discard everything else, e.g., in order to decrease the chance of *overcounting*. Then, we will use the *conservative grouping* operator and obtain the groups $Group_c(m, a_1, t) = \{f_2\}$ and $Group_c(m, a_2, t) = \emptyset$. Next, if we wish to aggregate *all* data, e.g., in order to decrease the chance of *undercounting*, then we will use the *liberal grouping* operator and obtain the groups $Group_l(m, a_1, t) = Group_l(m, a_2, t) = \{f_1, f_2\}$. Finally, if we wish to aggregate the data given with a *reliable degree of confidence*, e.g., in order to balance the chances of *undercounting* and *overcounting*, then we will use a *degree-of-confidence grouping* operator, e.g., $Group_d(l, [0.5; 1])$. In this case, we obtain the groups $Group_d(m, a_1, t, [1; 1], [0.5; 1], [1; 1]) = \{f_1, f_2\}$ and $Group_d(m, a_2, t, [1; 1], [0.5; 1], [1; 1]) = \emptyset$. ∎

Aggregation Functions In the following, we discuss aggregation functions. We assume a group

$$G = \{f_j \in F \mid \bigwedge_{k=1}^{n} f_j \rightsquigarrow^i_{[p^{k,j}_{min}; p^{k,j}_{max}]} l_k\}.$$

We start with the COUNT function, which counts *minimum expected*, *maximum expected*, *average expected*, *definite*, and *possible* number of facts that belong to the group G.

Definition 6.5. [**COUNT function**] Given that N is the number of facts in the group G, we define different kinds of counts.

1. The *minimum expected count* is:
$$COUNT_{min}(G) = \sum_{j=1}^{N} (p^{1,j}_{min} \cdot \ldots \cdot p^{n,j}_{min})$$
2. The *maximum expected count* is:
$$COUNT_{max}(G) = \sum_{j=1}^{N} (p^{1,j}_{max} \cdot \ldots \cdot p^{n,j}_{max})$$
3. The *average expected count* is:
$$COUNT_{avg}(G) = \sum_{j=1}^{N} (\frac{p^{1,j}_{min} + p^{1,j}_{max}}{2} \cdot \ldots \cdot \frac{p^{n,j}_{min} + p^{n,j}_{max}}{2})$$

4. If G is a conservative group, then the *definite count* is:

$$COUNT_{def}(G) = N$$

5. If G is a liberal group, then the *possible count* is:

$$COUNT_{pos}(G) = N$$ ∎

Since the procedure of computing the expected counts assigns degrees of group membership to facts, any grouping including the liberal and conservative groupings may be considered weighted groupings.

Example 6.6. [**COUNT function**] Continuing Example 6.4, we consider the following three groups: $G_c = Group_c(m, a_1, t)$, $G_l = Group_l(m, a_1, t)$, and $G_d = Group_d(m, a_1, t, [1;1], [0.5;1], [1;1])$.

Then, $COUNT_{min}(G_c) = 1 \cdot 1 \cdot 1 = 1$, $COUNT_{min}(G_l) = 1 \cdot 0.72 \cdot 1 + 1 \cdot 1 \cdot 1 = 1.72$, and $COUNT_{min}(G_d) = 0.72 + 1 = 1.72$. Also, $COUNT_{max}(G_c) = 1$, $COUNT_{max}(G_l) = 0.88 + 1 = 1.88$, and $COUNT_{max}(G_d) = 0.88 + 1 = 1.88$. ∎

As may be seen from Example 6.6, different $COUNT$ functions, in combination with different kinds of grouping, produce different values. For example, the difference between $COUNT_{min}(G_c)$, and $COUNT_{max}(G_l)$ is 88%. The former (latter) value is useful when the user wishes to maximally not overcount (undercount). In case the user wishes to obtain less extreme values, he/she may use an "intermediate" combination of a $COUNT$ function and grouping, such as $COUNT_{min}(G_l)$, $COUNT_{avg}(G_c)$, etc., that produce values between $COUNT_{min}(G_c)$ and $COUNT_{max}(G_l)$. Thus, the introduced means of querying flexibly adapt to concrete situation.

Due to space constraints, we only briefly discuss other aggregation functions. First, we consider the SUM function, which is, in essense, a generalization of the $COUNT$ function. Intuitively, the former sums arbitrary values of a measure, while the latter sums values of 1. Suppose, in an MO, the n^{th} dimension supplies data for the function. We assume that this dimension is *regular*, i.e. (1) there are only full containment relationships in the dimension hierarchy and (2) facts are only mapped to this dimension deterministically. Then, given the group G, we define the *minimum expected sum* by modifying the definition of the minimum expected count as follows:

$$SUM_{min}(G) = \sum_{j=1}^{N} (p_{min}^{1,j} \cdot \ldots \cdot p_{min}^{n-1,j} \cdot v(l_n, f_j))$$

where $v(l_n, f_j)$ is a numerical value assigned to a dimension value l such that $l \sqsubset l_n$ and $(f, l, 1, 1) \in R_n$. This way we sum the most precise data. Definitions of *maximum* or *average expected* and *possible* or *definite* sums can be obtained by modifying the definitions of the corresponding counts analogously.

Example 6.7. [**SUM function**] For example, suppose in our case study, we added a fourth dimension that captured weights of user cars. In addition, suppose (1) values $w_5 \in \widehat{D}_4$, $w_{2.5} \in \widehat{D}_4$, and $w_{1.75} \in \widehat{D}_4$, stand for 5, 2.5, and 1.75 tons, respectively, (2) $w_{2.5} \sqsubset w_5$ and $w_{1.75} \sqsubset w_5$, and (3) $(f_1, w_{1.75}, 1, 1) \in R_4$ and $(f_2, w_{2.5}, 1, 1) \in R_4$. Thus, $v(w_5, f_1) = 1.75$ and $v(w_{10}, f_2) = 2.5$. Then, we could find the *minimum expected sum* of weights of cars of users from the group $G'_l = Group(m, a_1, t, w_5)$ (see Example 6.6), as follows: $SUM_{min}(G'_l) = 1 \cdot 0.72 \cdot 1 \cdot 1 \cdot 1.75 + 1 \cdot 1 \cdot 1 \cdot 1 \cdot 2.5 = 1.26 + 2.5 = 3.76$. ∎

Second, we consider the AVG function. Given the group G, we define (different kinds of) the function as follows:

$$AVG_{mod}(G) = \frac{SUM_{mod}(G)}{COUNT_{mod}(G)}$$

where mod is one of the following: min, max, avg, def, and pos.

Finally, we consider the MIN function. Given the group G, we define the *possible* and *definite minimum* as follows:

$$MIN_{mod}(G) = min(\{v(l_n, f_j), j = 1, \ldots, N\})$$

where mod is either pos or def and min is a function that returns the minimum number from a set of numbers. Analogously with the $COUNT$ function, MIN_{pos} (MIN_{def}) is defined, if G is a *liberal* (*conservative*) group.

Example 6.8. [**Queries**] In this example, we express queries from Section 2 with the operators of our probabilistic algebra. The expressions are:
(1) $COUNT_{min}(GROUP_l(a, l_{ms}, t))$,
(2) $AVG_{avg}(GROUP_l(m, l_{90}, t))$,
(3) $COUNT_{max}(GROUP_d(\top, l, t, [1;1], [0.75;1], [1;1]))$,
(4) $MAX_{pos}(GROUP_c(\top, l, t))$. ∎

6.3. Other Operators

Union Operator The union operator is used to take the union of two MOs. Prior to defining the operator itself, we define two helper union operators, *union on dimensions* and *union on fact-dimension relations*. Due to space constraints, all the definitions in this section are informal.

Given two dimensions of the same type, D_1 and D_2, the *union operator on dimensions*, $\bigcup^D$, builds a new dimension, denoted $D' = D_1 \bigcup^D D_2$ by performing set union on corresponding categories and by building a new relation $\sqsubset^{D'}$ on dimension values: there exists a *direct* relationship between two values l_1 and l_2 if there exist a *direct* relationship between the values in either dimension. The degree of containment d for a resulting relationship depends on the degrees for given relationships d_1 and d_2 and may be determined in different ways. For example, if a user wants to decrease the chance of *overcounting (undercounting)* facts in the resulting MO, then d should be equal to the *minimum (maximum)* of d_1 and d_2. The *indirect* relationships between values in the resulting dimension are inferred using our transitivity rules from Section 5.

Next, given two fact-dimension relations, R_1 and R_2, relating facts and dimensions of the same type, the *union operator on the relations*, $\bigcup^R$, builds a new fact-dimension relation $R' = R_1 \bigcup^R R_2$: the new relation relates a fact and a dimension value, if either relation relates the fact and the value. The probabilities for a resulting relationship, p'_{min} and p'_{max}, depend on the probabilities for given relationships, p^1_{min}, p^1_{max}, p^2_{min}, and p^2_{max}, and may be determined in different ways. For example, if the facts-dimension relationships from the first MO are *less precise* than those from the second MO, then p'_{min} and p'_{max} should be equal to p^2_{min} and p^2_{max}, respectively. The *fact characterizations* are inferred using the rules from Section 5.

Finally, given two MO's with common fact schemas, M_1 and M_2, the *union operator*, $\bigcup$, builds a new MO, $M' = M_1 \bigcup M_2$, by combining dimensions and fact-dimension relations with the help of the $\bigcup^D$ and $\bigcup^R$ operator, respectively.

Given argument MOs, *difference*, *projection*, and *identity-based join* operators are not meant to transform the probabilities of the fact characterizations or the degrees of containment in the dimensions. The only requirement is to preserve the probabilities. Therefore, we define the probabilistic version of these operators as their deterministic counterparts in [20] except the probabilistic operators take probabilistic MOs as arguments and produce probabilistic MOs in the result. However, more general versions of difference and join operator that transform or combine probabilities of input MOs may also be useful.

7. Conclusions and Future Work

Motivated by the increasing use of location-based data warehouses (LBDWs) in industry, and the need to handle complex, dynamic, uncertain multidimensional data in such LBDWs, we propose a probabilistic data model that is able to capture the complexity of such data. The model provides a foundation for handling LBDW data, e.g., LBS data such as transportation infrastructures and the attached static/dynamic content (e.g., speed limits and vehicle positions). The paper also formally defines a set of algebraic operators that support querying of the afore-mentioned data. Finally, the paper outlines a real-world case study, based on our collaboration with a leading Danish vendor of location-based services. To our knowledge, this paper is the first to address the management of LBDW data.

In future work, it is interesting to generalize the expected degree of containment approach to intervals. Also, one interesting area of future work is to extend the fact characterizations to include a density function, in addition to a minium and maximum probability. The density function, e.g., uniform or normal, could be used to provide more information in the computation of some aggregates. However, more research is needed to ensure that the cost of adding and using this information is feasible. Furthermore, this paper considers only identity-based joins, but it would be useful to generalize our approach to consider other kinds of joins, in particualr joins on uncertain attributes [1]. On the implementation side, the interesting directions concern pre-aggregation issues, e.g., (1) using pre-aggregation to compute a wide range of aggregate functions, (2) probabilistic pre-aggregation techniques, and (3) (which would enable LBS queries for future time) pre-aggregation techniques for dynamic content such as user positions, including an embedded probabilistic position prediction method. Also, it is important to define a user-friendly, possibly SQL-like, query language.

Acknowledgements

This work was supported in part by the Danish Centre for IT Research, grant no. 216.

References

[1] D. Barbara et al. The Management of Probabilistic Data. *TKDE* 4(5):487–502, 1992.

[2] R. Cavallo and M. Pittarelli. The Theory of Probabilistic Databases. In *Proc. VLDB*, pp. 71–81, 1987.

[3] M. H. DeGroot and M. J. Schervish. Probability and Statistics. *Addison-Wesley*, 816 pp., 2002.

[4] R. Cheng et al. Querying Imprecise Data in Moving Object Environments. *TKDE* 16(9):1112–1127, 2004.

[5] R. Cheng et al. Evaluating Probabilistic Queries over Imprecise Data. In *Proc. SIGMOD*, pp. 551-562, 2003.

[6] N. Dalvi and D. Suciu. Efficient Query Evaluation on Probabilistic Databases. In *Proc. VLDB*, pp. 864–875, 2004.

[7] Euman. *www.euman.com* (in Danish). Current as of 2/1/05.

[8] E. Gelenbe and G. Hebrail. A Probability Model of Uncertainty in Data Bases. In *Proc. ICDE*, pp. 328-333, 1986.

[9] R. H. Güting et al. A Foundation for Representing and Querying Moving Objects. *ACM TODS* 25(1):1–42, 2000.

[10] C. Hage et al. Integrated Data Management for Mobile Services in the Real World. In *Proc. VLDB*, pp. 1019-1031, 2003.

[11] R. Jacobson. Microsoft SQL Server 2000 Analysis Services Step By Step. *Microsoft Press*, 400 pages, 2000.

[12] C. S. Jensen et al. Multidimensional Data Modeling for Location-Based Services. *The VLDB Journal* 13(1):1–21, 2004.

[13] R. Kimball et al. The Data Warehouse Lifecycle Toolkit. *Wiley Computer Publishing*, 800 pages, 1998.

[14] A. Klug. Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *J. ACM* 29(3):699-717, 1982.

[15] B. R. Moole. A Probabilistic Multidimensional Data Model and Algebra for OLAP in Decision Support Systems. In *Proc. IEEE SoutheastCon*, pp. 18–30, 2003.

[16] C. Murray. Oracle spatial user guide and reference, Release 9.2. *Oracle Corporation*, 2002.

[17] NCHRP. A Generic Data Model for Linear Referencing Systems. *Transportation Research Board, Washington, DC*, 28 pp., 1997.

[18] Oracle Corporation. Oracle9i Business Intelligence. In: *otn.oracle.com/products/bi/content.html*. Current as of 2/1/05.

[19] D. Papadias et al. Query Processing in Spatial Network Databases. In *Proc. VLDB*, pp. 802–813, 2003.

[20] T. B. Pedersen et al. A Foundation for Capturing and Querying Complex Multidimensional Data. *Information Systems* 26(5):383–423, 2001.

[21] T. B. Pedersen and N. Tryfona. Pre-aggregation in Spatial Data Warehouses. In *Proc. SSTD*, pp. 460–478, 2001.

[22] S. Šaltenis et al. Indexing the Positions of Continuously Moving Objects. In *Proc. ACM SIGMOD*, pp. 331–342, 2000.

[23] L. Speičys et al. Computational Data Modeling for Network-Constrained Moving Objects. In *Proc. ACM GIS*, pp. 118–125, 2003.

[24] J. Sun et al. Querying about the Past, the Present and the Future in Spatio-Temporal Databases. In *Proc. ICDE*, pp. 202–213, 2004.

[25] Y. Tao et al. Spatio-Temporal Aggregation Using Sketches. In *Proc. ICDE*, pp. 214–226, 2004.

[26] I. Timko and T. B. Pedersen. Capturing Complex Multidimensional Data in Location-Based Data Warehouses. In *Proc. ACM GIS*, pp. 147-156, 2004.

[27] I. Timko et al. Probabilistic Data Modeling and Querying for Location-Based Data Warehouses. *www.cs.auc.dk/DBTR*, 2005.

[28] G. Trajcevski. Probabilistic Range Queries in Moving Object Databases with Uncertainty. In *Proc. MobiDE*, pp. 39-45, 2003.

[29] G. Trajcevski et al. The Geometry of Uncertainty in Moving Objects Databases. In *Proc. EDBT*, pp. 233–250, 2002.

[30] D. Zhang et al. Efficient Aggregation over Objects with Extent. In *Proc. PODS*, pp. 121–132, 2002.

[31] D. Zhang et al. Temporal and Spatio-Temporal Aggregations over Data Streams Using Multiple Time Granularities. *Information Systems* 28(1–2): 61–84, 2003.

Querying the Content of Images in Material Science: An Integration of SQL and Map Algebra

Yan Huang, Brian Harrington
Computer Science & Engineering
University of North Texas
$[huangyan, brh]$@$cs.unt.edu$

Nandika Dsouza
Material Science
University of North Texas
$ndsouza@unt.edu$

Robert Brazile
Computer Science & Engineering
University of North Texas
$brazile@cs.unt.edu$

Abstract

Material scientists regularly acquire and analyze infrared images of deforming objects in their material tensile deformation, crack propagation, and fracture toughness tests. Although there are many image processing packages, none of them are available in a database integrated fashion. Material scientists typically select a set of image files satisfying given constraints by browsing a file directory/catalog, and then perform simple but labor-intensive content-querying on the images. These simple queries take days to answer. A more serious problem is that material scientists are not equipped with the flexibility to query images across different time snapshots or materials to validate their research hypotheses.

In this paper, we report about our work helping material scientists accelerate their work. Specifically, we propose a database approach to solve the problem of storing images and querying the content of images. In particular, we (1) proposed to use map algebra operations to compose image content querying needed by material scientists; (2) developed an SQL integrated image data cartridge which implements a core set of map algebra operations needed by material scientists; (3) analyzed the query processing and evaluation challenges; and (4) empirically evaluated the performance of three approaches, namely a multi-dimensional array based approach, a relational table based approach, and a binary large object (BLOB) based approach on bulk loading and typical material science queries using both real and synthetic data.

1 Introduction

Material scientists regularly acquire and analyze infrared images of deforming objects [22, 23, 3, 4] in their material tensile, crack propagation, and fracture toughness tests. The technique of thermal wave imaging utilizes the theories of radiation heat transfer which occur in the wavelengths of 3-13 m band. The overlap of a substantial wavelength with that of infrared (IR) wavelength leads to the utility of IR thermometry to interpret the temperatures generated in the emitting material. Figure 1 shows the basic setup material scientists use to study the materials.

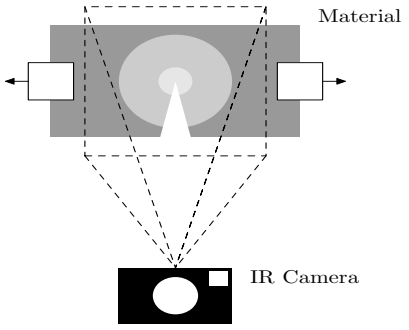
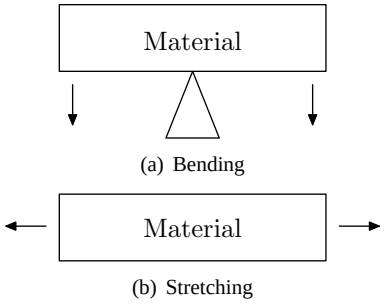


Figure 1. Setup used for collecting images.

Test machines such as Instron Materials are used to apply bending or stretching to material samples as shown in Figure 2, which allows material scientists to study how the material reacts under stress and strain. During the test the IR emission from the sample is monitored as a function of time using infrared cameras such as an Inframetrics IR740 Camera (8 m-12 m spectral band) at a 30 Hz frame rate. When a crack forms, energy is dissipated as heat [22, 23]. In general, the area around the crack-tip should have the highest temperature. Figure 3 (gray scaled to allow proper print) shows two images from the same test at different times. Notice that the area between the two crack-tips is brighter in the latter image indicating it has a higher temperature.

Each sample image set has associated meta-data which may correspond to the type of material, the concentration of filler, timestamps for the images, and the experimental setup. Material scientists ask various spatio-temporal questions [22, 23] about the images they capture, for example (1) Where is(are) the crack-tip(s) for a sample? (2) What is the average temperature 5 mm from the crack-tip for a given sample? (3) What is the temperature change for a point (x, y) while a sample is pulled? (4) What is the change in distance between two marked points after a given material is pulled? (5) What is the area of deformation for a sam-

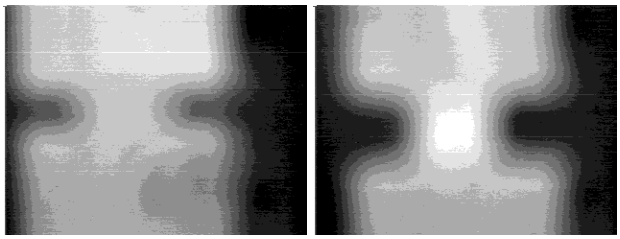


(a) Bending

(b) Stretching

Figure 2. Deformations applied to the material.

ple? (6) What is the difference in the area of deformation for different samples at time=10 seconds? (7) How do the contours of the image grow over time? (8) When does the area of deformation become greater than A? (9) Which materials have similar areas of deformation to a given sample? These questions involve both meta-data (e.g. material and time) and content of the sample images (e.g. crack tip and formation area).



(a) Time=0 sec (b) Time=13 sec

Figure 3. Snapshots of a material as it is being stretched.

Like scientists in many other domains, material scientists are facing both data processing and data management challenges. Currently, all questions are typically answered by isolated software packages, e.g. Image J and Matlab, on isolated image files. It takes material scientists days to get answers to simple queries on these data using either inflexible predefined functionalities of software packages or writing customized codes. A more serious problem is that material scientists are not equipped with the flexibility to issue ad hoc queries across different time snapshots or materials to validate their research hypotheses. Their research has been greatly hindered by the limited expressive power of image querying languages/tools.

In this paper, we report our work on helping material scientists accelerating their answers to their research questions. Specifically, we propose a database approach to solve the problem of storing and querying images that has plagued material scientists for years. In particular, we first proposed to use map algebra operations to compose image content querying needed by material scientists. Then we developed a SQL integrated image data cartridge which implements a core set of map algebra operations needed by material scientists. Third, we analyzed the query processing and evaluation challenges. We proposed possible solutions although the actual implementation and evaluation of these solutions are out of the scope of this paper. Finally, we empirically evaluated the performance of three basic approaches, namely a multi-dimensional array based approach, a relational table based approach, and a binary large object based (BLOB) approach on bulk loading and typical material science queries using both real data and synthetic data.

The paper is organized as follows. Section 2 starts off by surveying related work. The proposed SQL integrated map algebra approach is described in section 3. Section 4 discusses the various implementation options of the proposed approach and their query processing challenges. We present the experiment results on the performance of data load and typical queries on both real and synthetic datasets in section 5.The paper concludes by discussing some of the future work in section 6.

2 Related Work

Researchers in scientific database management have been addressing the problem of organizing and querying images for many years [2, 19, 7]. Images typically have both meta-data and content. Approaches for managing images have been centered around meta-data based, content based, or a hybrid approach. A meta-data based approach could easily adopt traditional database operations. A content based approach, noticeably Content Based Image Retrieval (CBIR) [21], retrieves images similar to a given image using features derived from image content, e.g. color, textures, and shapes. A hybrid approach allows queries involving both meta-data and image contents. However the "content" in research literature is generally referring to features describing the images in a very general sense, e.g. color histogram and textures. In a broad sense, our work is related to the hybrid approach. The uniqueness of our approach is the ability to allow material scientists to ask ad hoc queries to the content of their images in a SQL integrated way.

Material scientists are interested in quantifying qualitative assessment of failure in materials. Temperature changes on the surface represented by thermal images are related to the change in stresses via thermoelastic equations. Quantification of this has been limited since tools that allow queries across data that are continuous across the surface are not generally available to material scientists. As a result, one of the authors had to resort to using map algebra [18, 5, 13], supported by many GIS software packages, to query the images in a semi automatic but still laborious fashion.

Spatial information is generally represented by two models: *object* and *field* [16]. The object model represents conceptual entities as objects in an intuitive and direct way. The *vector* data structure implements the *object* model using polygons, lines, and points to represent shapes of objects, e.g. lakes, buildings, and rivers. The *field* model is often used to represent continuous phenomena over a space, e.g. temperature. The raster data structure obtained by imposing a uniform grid on the underlying space implements the *field* model on a computer. Map algebra [18, 5, 13] is an informal and the *de facto* geographic and cartographic modelling language for manipulating raster data. As a high-level computational language to describe geographic data processing, map algebra creates new map layers using existing map layers and operations in a sequence. It is a powerful raster manipulation language by allowing calculation of values for locations based on the location itself, its neighborhood, a related zone, or the entire raster. Although supported by many GIS software products, e.g. ArcInfo [6], AutoCAD, and PCRaster [11], map algebra is not supported in an SQL integrated fashion. Users first need to select the raster files satisfying given constraints, e.g. all images of material X with $y\%$ filler concentration, by either browsing a file directory/catolog or writing SQL statements. The names of raster files are then embedded into map algebra operations manually.

A logical way to support image content querying in a database system is to extend current database data types and query languages. Many object-oriented or object-relational database management systems allow abstract data types (ADT) together with stand alone/member functions defined by end users. Such research prototypes/commercial/open source software products include Postgres [8], Monet [1], Jasmine [9], Starburst [15], Oracle, DB2, and MySQL. Many of them support images as binary large objects (BLOB). A cartridge supporting map algebra in an SQL integrated fashion is not known.

In a similar vein as our proposed approach in this paper, Nes proposed to add core image processing functionality to the database management system, making it a better tool for image analysis research in his thesis [10]. Image algebra [14] is a formal set of image-processing operations many of which may not have practical usage in material science image content querying, which is a very light form of image analysis. Geo-algebra [17] attempts to borrow techniques from image algebra to formalize and extend the functionalities of map algebra.

RasDaMan [20] is an extension of an ODBMS to support multidimensional arrays. In addition, it also supports an extended version of SQL called RasQL that allows the user to select and manipulate multidimensional array data. However, for the queries posed by material scientists the expressive power of RasQL is limited by the difficulty of performing focal and zonal operations.

3 SQL Integrated Map Algebra Approach

Fracture mechanics relies on balancing the energy before and after failure in terms of incremental crack growth. Since temperature has a dominating effect on polymeric materials, analysis of the temperature fields around the crack tip over the time of propagation of the crack to failure yields information on the state of the material. Being able to issue ad hoc queries in the form of :"What is the temperature change for a point (x, y) while a given material X with filler concentration $s\%$ is pulled?" or "What is the average temperature l mm from the crack-tip for material X with filler concentration $s\%$ at time t?" helps material scientists to study the state of a material at a given time. The latter question could also be easily extended to queries like "What is the rate of temperature change l mm from the crack-tip for material X with filler concentration $s\%$ at time t? This section describes how map algebra operations can be used in conjunction with SQL to answer these queries.

We classify the material science queries into two categories, namely image content queries and multi-criteria queries. The first category has simple meta-data conditions to specify which images you want content information for, while the second category has mixed meta-data and image content conditions.

3.1 Map Algebra

Map algebra [18] is a cell by cell combination of raster layers using some mathematical operation. A raster layer is a grid imposed over an image with a value for each cell. Map algebra can be extended to support most mathematical operations found in a spreadsheet tool. The result of a map algebra operation is another raster layer. Operators can be grouped into four basic categories (1) local operations where the value of a cell in the output is a function of the corresponding cells of the input; (2) focal operations that determine the output value of a cell based on a small specified neighborhood; (3) zonal operations that determine the output based on all of the values in a zone; (4) global operations determine the output based on all of the values in the raster.

1	2	4		2	3	5		3	5	9
3	6	7	+	4	7	8	=	7	13	15
5	9	8		6	1	9		11	10	17

Figure 4. Local addition operator.

Figure 4 shows a local addition operator on two rasters. Local operations are useful for general manipulation such as for looking at the difference in temperatures between two images of a material over time.

Figure 5 shows the general focal max with a 3×3 neighborhood. This operation chooses the new cell value for the center pixel of the output by looking at all of the cells in

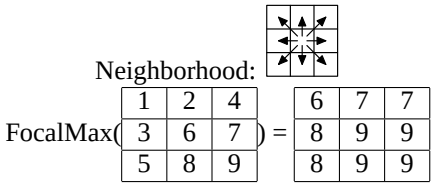


Figure 5. Focal maximum operator.

the specified neighborhood. Focal operations are useful in many material science image queries, e.g we can find the boundaries of the temperature zones by using the focal majority operator (will be described in detail later in section 3.2).

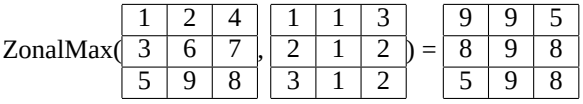


Figure 6. Zonal maximum operator. The first raster is the input and the second is the zonal map.

For some tasks zonal operations are also useful. For example, as stated earlier, the crack-tip in general should have the highest temperature. In order to find the location of the crack-tip, we need to know the location with the highest temperature or in terms of the raster the pixel with the maximum value in some zone. Figure 6 shows how the zonal maximum operator works. A zone is the set of cells that have the same value in the zonal map raster. The new cell value is determined by looking at all cells in the same zone. Global operations are a special case of zonal operations where the entire raster is one zone.

Local Operations	
Arithmetic	add, subtract, abs, sqrt, etc...
Boolean	equal, greater than, less than
Aggregate	min, max, minority, majority, median, sum
Statistic	standard deviation, mean
Focal Operations	
Aggregate	min, max, minority, majority, median, sum
Statistic	standard deviation, mean
Zonal and Global Operations	
Geometric	fill, area, perimeter, thickness, distance
Aggregate	min, max, minority, majority, median, sum
Statistic	standard deviation, mean
Miscellaneous Operations	
Miscellaneous	get value, trim

Table 1. Map algebra operations.

There are also two useful operations that do not fit well into the other categories. Both involve reducing the dimensions of an input raster to get a new raster. The first provides for getting the value of a specified cell or zone and returns

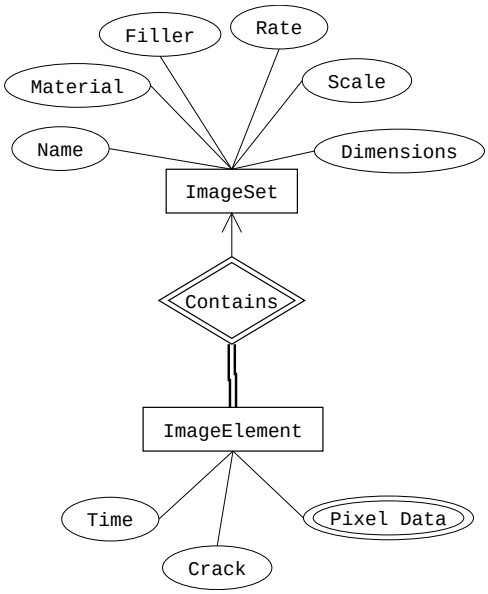


Figure 7. ER diagram of schema.

a scalar value[1]. The second trims a raster and returns some sub-raster.

Many GIS products support some form of map algebra, however currently there is no official map algebra standard. For this paper table 1 summarizes the representative operations we use for each category.

For all our queries we will assume a simple schema shown in the entity-relationship diagram in figure 7. The *ImageSet* entity set would contain the general description of a set of images. This would include the meta-data that is the same for every image in the set such as the set name, material, filler, the scale for the images, and the dimensions of the images, and strain rate applied to the material. For example, the material might be a polypropylene + ethylene propylene diene blend, with a filler that is a 1000 nanometer wide by 1 nanometer thick ceramic platelet added to the material, and a 4 mm/min strain rate. For simplicity, we will use filler concentration to represent questions related to various treatment applied to the material. The *ImageElement* entity set has the actual image data along with the time stamp for the image and the location of the crack tip.

3.2 Expressing Image Content Queries

The first class of queries involves querying the content of the images. The constrains of these queries are simple meta-data constraints, e.g. material type and filler concentration. The main challenge is how image content can be queried using the map algebra operations.

It is assumed for this paper that the materials have only one crack and thus one crack-tip. The crack propagation

[1]This can be thought of as a 1×1 raster so the operation is still closed.

ID	SQL Statement	Description
Q1	**SELECT** ct.crack_tip **FROM** ImageElement **as** ie, ImageSet **as** is **WHERE** Material=X **and** **Time**=t **and** Filler=s **and** is.SetID=ie.SetID;	Where is(are) the crack-tip(s) for material X with filler concentration $s\%$ at time t?
Q2	**SELECT** FindShape(ie.Data) **FROM** ImageElement **as** ie, ImageSet **as** is **WHERE** Material=X **and** Filler=s **and** is.SetID=ie.SetID;	What is the shape of the crack for material X with filler concentration $s\%$ at time t?
Q3	**SELECT** GetValue(ie.Data, x, y) **FROM** ImageElement **as** ie, ImageSet **as** is **WHERE** Material=X **and** Filler=s **and** is.SetID=ie.SetID;	What is the temperature change for a point (x, y) while a given material X and filler concentration $s\%$ is pulled?
Q4	ZM=MakeRaster("circle", CX, CY, ToPixels(5)) **SELECT** GetValue(ZAvg(ie.Data, ZM), ZM, 1) **FROM** ImageElement **as** ie, ImageSet **as** is **WHERE** Material=X **and** Filler=s **and** is.SetID=ie.SetID **and time** = t;	What is the average temperature 5 mm from the crack-tip for material X with filler concentration $s\%$ at time t?
Q5	**SELECT** **Time**, CreateMap(ie.Data) **FROM** ImageElement **as** ie, ImageSet **as** is **WHERE** Material=X **and** Filler=s **and** is.SetID=ie.SetID;	How do the contours of the image grow over time?

Table 2. Typical Material Science Image Content Queries(Q1-Q5) and Their SQL Statements

rate is important for determining how resilient a material is to stress. The propagation rate can easily be determined given the location of the crack-tip for each image. So the query can be formulated as "(Q1) Where is the crack-tip for material X with filler concentration $s\%$ at time t?". As the crack-tip is part of the meta-data this query can be answered using standard SQL. The resulting SQL statement is shown in Table 2.

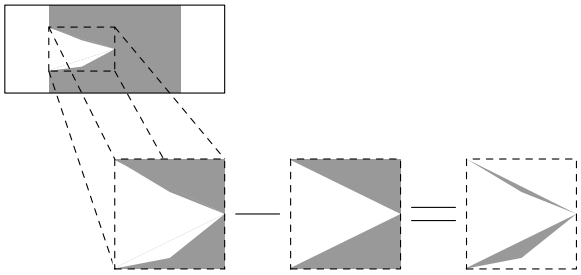


Figure 8. Finding the shape of a crack.

The shape of the crack that is formed in the material is important because it can indicate how fast the crack is likely to propagate through the material. A common question would be "(Q2) What is the shape of the crack for material X with filler concentration $s\%$ at time t?".

To determine the shape of the crack the image first needs to be cleaned up. For determining the shape of the crack only two zones are needed: one that represents the background with a value of zero in every cell, and one that represents the material with a value of one in every cell. This can be done using a simple local thresholding operation. Then the minimum bounding box around the crack is selected as shown in figure 8. Since the location of the crack-tip is part of the meta-data, a reference raster can be created for a particular shape such as a triangle. The reference raster is subtracted from the actual crack. The fewer non-zero values there are in the result the closer the actual crack is to the reference shape. The *FindShape* macro performs the steps outlined earlier for each of the basic crack shapes to see which shape matches with the highest probability. The returned result is a textual description of the basic shape, e.g. triangle or parabolic. The resulting SQL statement is shown in table 2.

As mentioned earlier temperature has a dominating effect on polymeric materials so it is important to be able to query the temperature in a number of ways including: a specific value "(Q3) What is the temperature for a point (x, y) while a material X with filler concentration $s\%$ is pulled?", the average for a set of values "(Q4) What is the average temperature p mm from the crack-tip for a material X with filler concentration $s\%$ at time t?", and for the growth of temperature zones "(Q5) How do the contours of the image for a material X with filler concentration $s\%$ grow over time?". Query Q5 is particularly useful as fracture toughness equations call for understanding the volume of the deformed material.

Query Q3 can be answered by simply using the get value operation to get a scalar value. As the image uses pixels a simple macro can be defined to convert millimeters to pixels as the dimensions of a pixel in millimeters is part of the meta-data for each image set. Query Q4 can be answered by creating a zone map raster that has a 1 in every cell that is p mm from the crack-tip and getting the value of that zone. Q3 and Q4 are shown in Figure 2

Query Q5 is more complicated. Since an image can have hundreds of distinct temperature values, the first step is to assign ranges of temperatures to a given value so that the result will have the desired number of zones. This is similar to adjusting the distance between contours on a topographic map to get the desired amount of detail. Let R be the original raster and M be the contour map.

$$T = GMax(R) - GMin(R) \quad (1)$$

$$D = LDivide(T, MakeRaster(n)) \quad (2)$$

$$M = FMajority(LRound(R, D, B)) \quad (3)$$

Equation 1 uses the global operations min and max to find the overall temperature range for the image. The output raster, T, is a raster where every cell has the temperature range as the value. In equation 2 the temperature range is divided by the desired number of zones, n. The function $MakeRaster(n)$ creates a raster where every cell has the value n. Equation 3 produces the final contour map raster, M, by rounding the values and using the focal majority operation with a 3×3 neighborhood to clean up the zones. Equation 4 shows how the i, j^{th} cell of the raster returned from the local round operation is determined from the corresponding cells of the three input rasters. A good value for B is $GMin(R)$.

$$R'_{i,j} = \left\lfloor \frac{R_{i,j} - B_{i,j}}{D_{i,j}} \right\rfloor D_{i,j} + \frac{D_{i,j}}{2} + B_{i,j} \quad (4)$$

As you will see later creating a contour map is useful for other operations. As such we will define the *CreateMap* macro which performs the steps outlined earlier to create a contour map from an image. Q3, Q4, and Q5 are shown in table 2.

Stress-strain curve could be used to divide materials into different classes that depend on their relative behavior under stress. Stress is a force applied over an area. When materials deform they are said to strain. A strain is a change in size, shape, or volume of a material. When a material is subjected to increasing stress it passes through three successive stages of deformation: reversible elastic deformation, irreversible ductile deformation, and fracture. Depending on the regions of elastic and ductile behaviors, materials can be said to be brittle or ductile. The extent of certain parts of the material have stretched could be used to measure the strain of the material. So material scientists want to know "(Q6) What is the change in distance between two marked points after the material is pulled?". This query can be answered by using the global distance function to find the distance from the cell marked $M1$ to every other cell. Then, using the original image as a zone map, get the value of the cell marked $M2$.

Many of the equations for determining the energy that is being dissipated as heat require the area of the deformed region. Given a contour map of the image this can be thought of as the area of the zone that has the highest temperature. In particular queries such as "(Q7) What is the area of deformation for material X with filler concentration $s\%$?" and "(Q8) What is the difference in the area of deformation for different samples at time t?" are very useful. As finding the deformation area is fairly common a *DfrmArea* macro can be defined that finds the area of the contour with the highest temperature as follows:

```
DfrmArea(R)
  MAP:=CreateMap(R, n)
  ZONE:=GetValue(GMax(MAP), 0, 0)
  return GetValue(ZArea(MAP), MAP, ZONE)
```

Essentially it creates a contour map and then uses the global max operation to find the correct zone. As global max creates a raster where every cell has the same value, we can get any value to pass to the operation for getting the value of a zone. The zonal area function calculates the area of each zone and then the value of the zone with the highest temperature is returned. Q6,Q7, and Q8 are shown in table 3.

3.3 Expressing Multi-Criteria Queries

In this section, we discuss queries whose constraints use information about image content as well as other meta-data to determine the result set.

Materials are selected by matching their properties to the service conditions. The growth rate of the area of deformation of a material is an important property of a material. The deformation area is defined as the area of the highest temperature zones. Useful questions include "(Q9) When does the area of deformation become greater than A?" and "(Q10) Which materials have similar areas of deformation to material X with filler concentration $s\%$ at time t?". Query Q9 requires a query with a image content condition "deformation area is greater than A" to be specified and can be expressed as:

```
SELECT Min(Time)
FROM ImageElement as ie, ImageSet as is
WHERE Material=X and Filler=s and
   is.SetID=ie.SetID
   and DfrmArea(ie.Data)>A;
```

Query Q10 is a complex query involving calculating intermediate results storing the deformation area of a given material with a given filler concentration at a given time and comparing every other images in the database with it. This query can be expressed as:

```
WITH Refer(A) as
  SELECT DfrmArea(ie.Data) as A
  FROM ImageElement as ie,
     ImageSet as is
  WHERE Material=X and Time=t and
     Filler=s and is.SetID=ie.SetID
```

ID	SQL Statement	Description
Q6	**SELECT** GetValue(GDist(ie.Data, M1), ie.Data, M2) **FROM** ImageElement **as** ie, ImageSet **as** is **WHERE** Material=X **and** Filler=s **and** is.SetID=ie.SetID;	What is the change in distance between two marked points after the material is pulled?
Q7	**SELECT** DfrmArea(ie.Data) **FROM** ImageElement **as** ie, ImageSet **as** is **WHERE** Material=X **and** Filler=s **and** is.SetID=ie.SetID;	What is the area of deformation for material X with filler concentration $s\%$?
Q8	**SELECT** DfrmArea(ie1.Data)−DfrmArea(ie2.Data) **FROM** ImageElement **as** ie1, ImageElement **as** ie2 **WHERE** ie1.SetID=1 **and** ie2.SetID=2 **and** **Time**=10;	What is the difference in the area of deformation for different samples at time=10 seconds?

Table 3. Typical Material Science Image Content Queries (Q6-Q8) and Their SQL Statements

```
SELECT Name, Material, Filler
FROM ImageElement as ie, ImageSet as is
WHERE is.SetID=ie.SetID
  and DfrmArea(ie.Data) − Refer.A <0.5;
```

These two queries represent a broad range of complex queries. Q10 may be easily extended to questions that require a spatial join of the raster images, e.g. "Find the pair of materials whose area of deformation correlated well".

4 Implementation and Query Processing

There are three basic methods which can be used to store images in an OO/OR database management systems. One way is to store them in a table that has an image id, (x,y)) coordinates, and the pixel value. This method is not very practical as it wastes space, since an image id and the coordinates must be explicitly stored for each pixel, and it leads to a huge table which usually has to be joined with another table to connect an image with its meta-data. One reason that this approach may still be useful is that it represents images and meta-data in a consistent way and standard database language such as SQL could be adopted in querying content of the image tables. One may want to implement the image tables in an efficient way while still providing end users with a table kind of view. A second method is to store images using the binary large object (BLOB) data type. The primary disadvantage with this is the lack of random access to cells. Finally, some OO/OR database management systems provide multidimensional array types which provide efficient storage with random access (provided the implementation is good).

For the first class of queries (Q1-Q8), the constraints are meta-data based. Traditional query processing and optimization techniques could be adopted well. When the images are implemented as tables, the image content queries require time consuming join operations and indexing on coordinates of cells may be useful. For the BLOB and multi-array based approaches the support for random cell access is critical in improving query performance.

The second class of queries have complex constraints involving both meta-data and derived values of image content. The major performance enhancement techniques include materializing frequently calculated values, e.g. deformation area, and building function based indexes for image content queries with variables, e.g. l mm from crack-tip.

5 Experiment Design and Results

We chose to test queries Q1, Q3, Q5, and Q9 to see how their performance varied with the number of images in a single image set. These four queries were chosen to get a representative sample of the performance of queries with no map algebra functions (Q1) compared to those using the get value operation (Q3), the create map macro (Q5), and the deformation area macro (Q9). Finally query Q10 was tested to see how it performed depending on the number of image sets where each set had 100 images.

The performance was tested on both synthetic and real data. The synthetic data consisted of 100 image sets that each had 100 images. Each image was 200 by 200 pixels. The deformation area of the images for each set was chosen randomly to be between 100 and 1600 pixels. The real data consisted of 26 images of a material as it was being deformed. The material was a polyethylene teraphthalate (PET) nanocomposite with 3% MLS concentration supplied by KOSA [12]. Like the synthetic data, each image was 200 by 200 pixels.

The test environment is PostgreSQL 8.0 running on Windows XP. The machine is a 2.8 GHz Pentium IV with 512 MB of ram. The map algebra operations and user macros were implemented in Java and connected to PostgreSQL using PLJava.

5.1 Results

Importing the data consists of reading the bitmap files for the real data or generating the appropriate synthetic data and storing this data in the database system. We used the Java ImageIO library with the Java Advanced Imaging plug-ins for working with the bitmap images.

For inserting the data the blob representation was the fastest. It took 512.116 seconds (8.54 minutes) to insert the

100 image sets of synthetic data. The array approach took 5994.515 seconds (99.91 minutes). The table approach was much slower than the other two taking 2594.297 seconds (41.57 minutes) to insert 1 data set.

For the synthetic data queries Q1, Q3, Q5, and Q9 were tested to see how the performance varied between the three storage techniques.

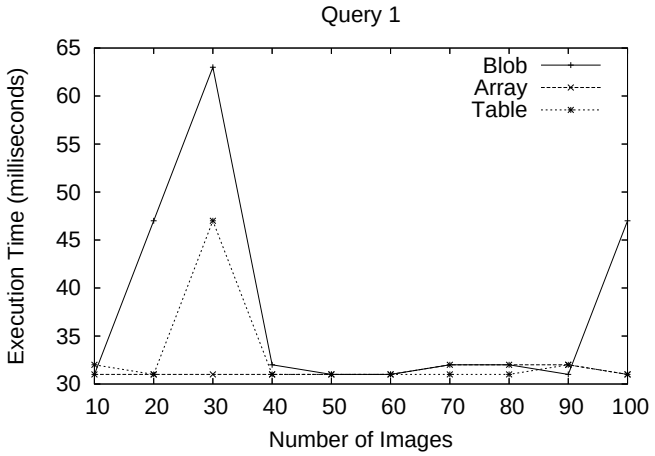


Figure 9. Results for tests on synthetic data with Q1.

Query Q1, shown in figure 9, does not use any map algebra functions and does not access the image data.

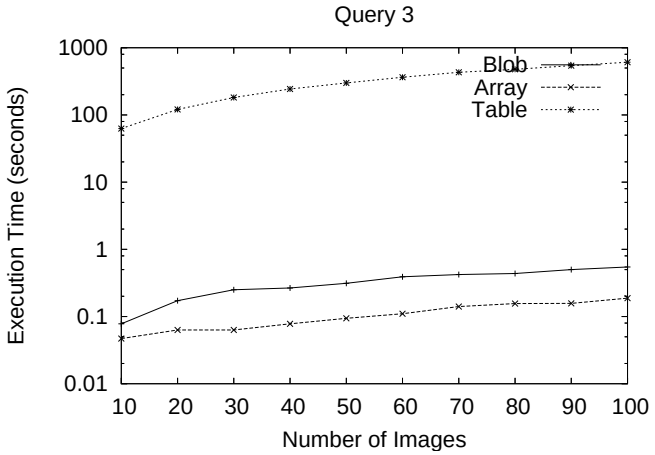


Figure 10. Results for tests on synthetic data with Q3.

It is shown as a reference for the general response time for simple queries under the test environment. Query Q1 could be answered in under 70 milliseconds regardless of the image representation. Note that the times for all of these queries are small so the spikes in the figure for the blob and table based representations were probably the result of other factors on the machine or in the database.

Query Q3, shown in figure 10, tests the time to access a single value from a particular image. The array approach did the best followed by the blob. The blob is approximately 2-3 times slower than the array based method. The table based approach was more than a thousand times slower than both. All methods had a linear growth rate.

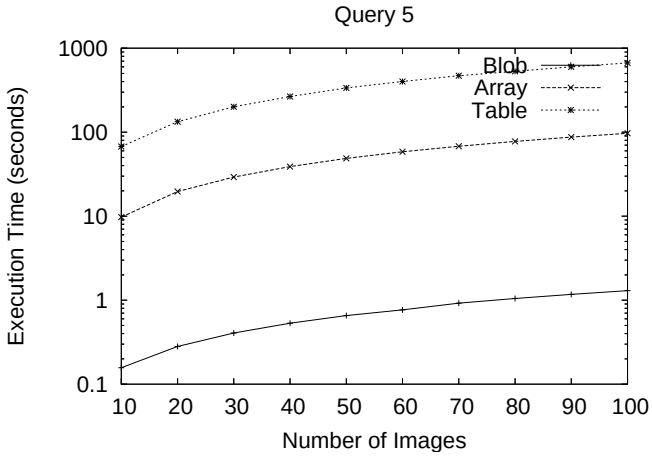


Figure 11. Results for tests on synthetic data with Q5.

Query Q5, shown in figure 11, tests the create map function. The blob approach was about 60 times faster than the array based approach. This was probably due to string processing that has to be done to pass the array from the system to the back-end functions. The table approach was several hundred times slower than the blob approach.

Query Q9, shown in figure 12, tests the deformation area function. The blob approach was about 16 times faster than the array based approach. The array approach was about 4 times faster for this query compared to query Q8 with the same number of images. This speed up is probably due to the deformation area function returning a simple floating point number and the create map returns a raster which for the array approach must be packaged into a string. The table approach was still much slower than the other two.

Query Q10 was also tested for synthetic data using the blob and array representations to see how they performed on a more complicated query comparing image sets. Figure 13 shows the results. The blob approach was considerably faster. With 100 image sets the blob approach took 364.938 seconds (6.08 minutes) where the array approach took 5108.891 seconds (85.14 minutes).

Real data was also tested for queries Q1, Q3, Q5, and Q9 using all of the images. It had roughly the same per-

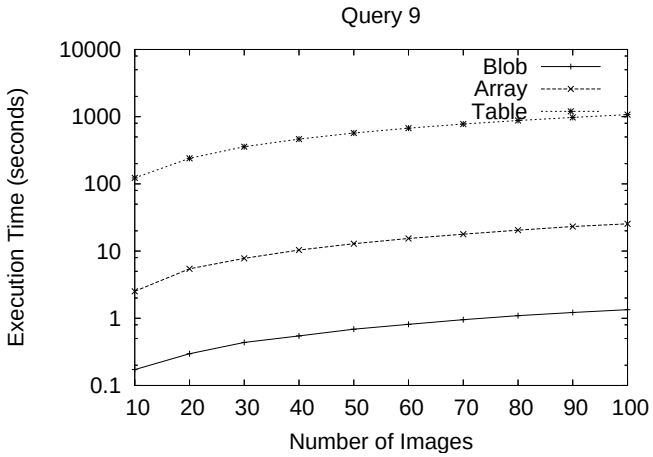


Figure 12. Results for tests on synthetic data with Q9.

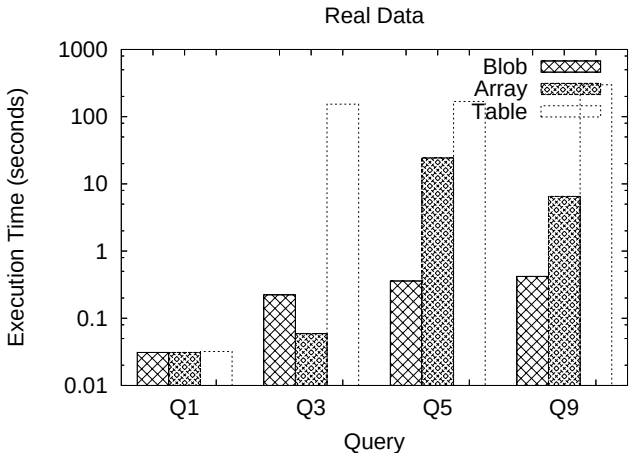


Figure 14. Results for tests on real data with different queries.

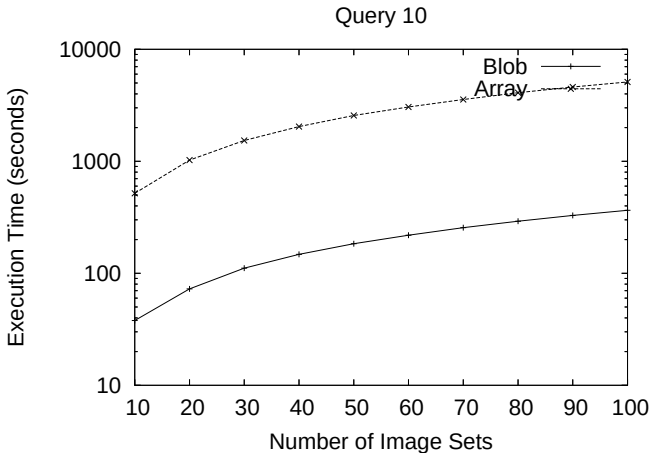


Figure 13. Results for Q10 on synthetic data.

formance as the synthetic data, which was expected since the image sizes and processing requirements were the same. Figure 14 shows the results. It can clearly be seen that the blob and array based representations have much better performance.

6 Conclusion

By integrating map algebra operations with SQL material scientists can now efficiently answer some ad-hoc queries based on meta-data and image content. These queries can be image content queries which return information about image content using conditions based solely on meta-data, or multi-criteria queries which allow for operations based on image content to be part of the conditions.

Of the three ways outlined for storing images in ORDBMS emperical results indicate that using the BLOB or multidimensional array currently offer the best performance with arrays being better for getting a specific value and BLOBs being better when the entire image must be loaded and manipulated as with the create map and deformation area macros. Using a separate relation does not perform well but has the advantage that the images can be manipulated using standard SQL.

Future work should focus on improving the shortcomings of the various ways of representing images, and increasing the integration with SQL to make image content queries less complex. Furthermore, work needs to be done to evaluate the benefit of storing or pre-computing values for commonly used functions or macros such as create map and deformation area. It is also possible that indexes could be built based on the results of these functions to speed up processing at the expense of storage space.

References

[1] P. A. Boncz and M. L. Kersten. Monet: An Impressionist Sketch of an Advanced Database System. In *Proceedings Basque International Workshop on Information Technology*, San Sebastian, Spain, July 1995.

[2] R. Chbeir, S. Atnafu, and L. Brunie. Image data model for an efficient multi-criteria query: A case in medical databases. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, pages 165–174, 2002.

[3] N. A. D'Souza, W. Brostow, L. D. Favro, A. C. Ramamurthy, R. L. Thomas, and Y. Wang. Mechanics of failure of plastics by thermal imaging: Examination of a thermoplas-

tic polypropylene + epdm blend. *Polymer Engineering and Science*, 36:194–202, 1996.

[4] N. A. D'Souza and A. Hernandez-Luna. Characterization of structural changes in polypropylene nanocomposites by infrared thermal wave imaging. In *Annual Technical Conference Society of Plastics Engineers*, page 1342, 2003.

[5] M. J. Egenhofer and T. Bruns. Visual map algebra: A direct-manipulation user interface for gis. In *VDB*, pages 235–253, 1995.

[6] ESRI. Arcinfo. http://www.esri.com/software/arcgis/arcinfo/.

[7] W. I. Grosky. Managing multimedia information in database systems. *Commun. ACM*, 40(12):72–80, 1997.

[8] T. P. G. D. Group. Postgresql documentation. http://www.postgresql.org/docs/7.4/static/index.html.

[9] H. Ishikawa, Y. Yamane, Y. Izumida, and N. Kawato. An object-oriented database system jasmine: Implementation, application, and extension. *IEEE Trans. Knowl. Data Eng.*, 8(2):285–304, 1996.

[10] N. Nes. Image database management system design considerations, algorithms and architecture. Ph.D. dissertation, University of Amsterdam, 2000.

[11] PCRaster. Pcraster environmental software. http://www.pcraster.nl/.

[12] S. Pendse, N. D'Souza, and J. A. Ratto. Deformation of pet nanocomposites.

[13] D. Pullar. Mapscript: A map algebra programming language incorporating neighborhood analysis. *GeoInformatica*, 5(2):145–163, 2001.

[14] G. X. Ritter, J. N. Wilson, and J. L. Davidson. Image algebra: An overview. *Computer Vision, Graphics, and Image Processing*, 49(3):297–331, 1990.

[15] P. M. Schwarz, W. Chang, J. C. Freytag, G. M. Lohman, J. McPherson, C. Mohan, and H. Pirahesh. Extensibility in the starburst database system. In *OODBS*, pages 85–92, 1986.

[16] S. Shekhar and S. Chawla. *Spatial Databases: A Tour*. Prentice Hall, ISBN: 0130174807, 2003.

[17] M. Takayama and H. Couclelis. Map dynamics integrating cellular automata and gis through geo-algebra. *International Journal of Geographical Information Science*, 11(1):73–91, 1997.

[18] D. Tomlin. *Geographic Information Systems and Cartographic Modeling* . Prentice Hall College Div, 1990.

[19] M. Ubell and M. Olson. Embedding image query operations in an object-relational database management system. In *Proc. Storage and Retrieval for Image and Video Databases III*, pages 197–203, 1995.

[20] N. Widmann and P. Baumann. Performance evaluation of multidimensional array storage techniques in databases. In *Proceedings of the 1999 International Database Engineering and Applications Symposium*, 1999.

[21] A. Yoshitaka and T. Ichikawa. A survey on content-based retrieval for multimedia databases. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):81–93, 1999.

[22] A. Zehnder. Temperature rise due to dynamic crack growth. http://people.ccmr.cornell.edu/˜atz/crack_temp.html.

[23] A. Zehnder, K. Bawa-Bhalla, R. Thomas, and L.D.Favro. Thermal analysis of crack tearing. http://people.ccmr.cornell.edu/ atz/crack_temp.html.

Fuzzy Decomposition of Spatially Extended Objects

Hans-Peter Kriegel, Martin Pfeifle
University of Munich, Germany
{*kriegel*, *pfeifle*}*@dbs.ifi.lmu.de*

Abstract

Modern database applications including computer-aided design, multimedia information systems, medical imaging, molecular biology, or geographical information systems impose new requirements on the effective and efficient management of spatial data. Particular problems arise from the need of high resolutions for large spatial objects. In this short paper, we sketch a new decompositioning approach based on clustering. We propose to describe a voxelized spatial object by a set of Gaussian distribution functions. Based on this decompositioning technique, we propose intersection queries which do not simply return a boolean value for each database object, but assign to each object a probability value indicating how likely an intersection is. The benefit of this approach compared to traditional approaches is that we do not any longer need an expensive refinement step for detecting whether objects intersect exactly on the fine-grained voxel sets.

1. Introduction

The efficient management of rasterized geographical objects has become an enabling technology for many novel database applications. As a common and successful approach, spatial objects can conservatively be approximated by a set of voxels, i.e. cells of a grid covering the complete data space (cf. Figure 1). By means of space filling curves, each voxel (often called pixel in 2D) can be encoded by a single integer and, thus, an extended object is represented by a set of enumerated voxels. As a principal design goal, space filling curves achieve good spatial clustering properties since cells in close spatial proximity are encoded by contiguous integers. Adjacent cell values can be grouped together to intervals, tiles or boxes which are basic datatypes for spatial applications.

By expressing spatial region queries as intersections of these spatial primitives, vital operations for GIS applications can be supported. For these applications suitable index structures, which guarantee efficient spatial query processing, are indispensable. An important new requirement for large spatial objects is a high approximation quality which is primarily influenced by the resolution of the grid covering the data space. A promising way to cope with high resolution spatial data may be found somewhere in between replicating and non-replicating spatial index structures. In the case of *replicating access methods*, e.g. the Relational Interval Tree [7], the number of the simple spatial primitives used to approximate the objects can become very high, resulting in a storage and query processing overhead. On the other hand, many of the *non-replicating access methods*, e.g. R-trees [5], use simple spatial primitives such as rectilinear hyper-rectangles for one-value approximations of extended objects. Although providing the minimal storage complexity, one-value approximations of spatially extended objects often are far too coarse. In many GIS applications, objects feature a very complex geometry. A non-replicating storage of such data causes region queries to produce too many false hits that have to be eliminated by subsequent filter steps. For such applications, the accuracy can be improved by decomposing the objects.

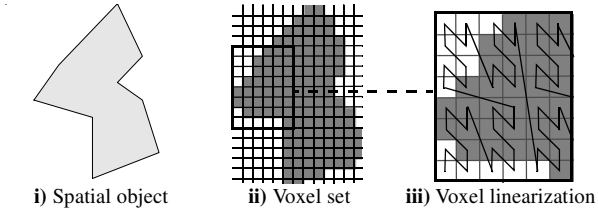


Figure 1. Voxelized Spatial Objects.

In this paper, we propose a new fuzzy decompositioning paradigm for high-resolution objects which is based on the well-known *k*-means clustering algorithm. The basic idea is to describe the voxel set by *k* clusters where the value of *k* depends on the characteristic of the voxel set. Each cluster is described by a few statistical values which are stored in a database. If we carry out collision or window queries, we determine for each object in the database a certain probability value that indicates the likelihood that the object belongs to the result set. This probability value can be computed by exploiting the statistical information describing the *k* clusters of the object and without accessing the fine-grained exact voxel representations. Note that the traditional approach also assigns probability values to the object, i.e. 0 if the object intersects the query object and 1 otherwise. As we omit the refinement step on the exact voxel representations, we can accelerate the complete query process.

The remainder of the paper is organized as follows. In Section 2, we present the related work in the area of spatial object decompositioning. In Section 3, we present our decompositioning approach based on clustering. In Section 4, we show how we can carry out fuzzy intersection queries based on decomposed spatial objects. Finally, we will close the paper in Section 5 with a short summary and a few remarks on future work.

2. Related Work

In this section, we will shortly present the related work in the area of decomposing high resolution voxelized objects.

Gaede pointed out that the number of voxels representing a spatially extended object exponentially depends on the granularity of the grid approximation [4]. Furthermore, the extensive analysis given in [2] shows that the number of voxels is proportional to the surface of the approximated object. Thus, in the case of large high resolution parts, the number of voxels can become unreasonably high.

A common approach to approximate the high resolution voxelized objects is to use their minimum bounding rectangles. Although providing the minimal storage complexity, one-value approximations of spatially extended objects often are far too coarse. In many applications, GIS or CAD objects feature a very complex and fine-grained geometry. The rectilinear bounding box of the brake line of a car, for example, would cover large parts of the data space. A non-replicating storage of such data would cause too many false hits in the *filter step* that have to be eliminated by the *refinement step*.

On top of the resolution of the data space and the clustering properties of the space-filling curve, a more fine-grained control of the trade-off between redundancy and accuracy is desired for many applications. Note that the granularity may have to differ for each individual object rather than to apply the same resolution to all objects. An approach to control this trade-off is the concept of size-bound and error-bound approximation [9] beyond the granularity-bound approximation [4]. A recursive subdivision procedure stops if the desired redundancy (size-bound) or the desired maximum approximation error (error-bound) is reached.

In [10], Kriegel and Schiwietz tackled the problem of "*complexity versus redundancy*" for 2D polygons. They investigated the natural trade-off between the complexity of the components and the redundancy, i.e. the number of components, with respect to its effect on efficient query processing. The presented empirically derived root-criterion suggests to decompose a polygon consisting of n vertices into $O(\sqrt{n})$ many simple approximations.

In [6] a high-resolution spatial object was decomposed based on its linearized voxel sequence (cf. Figure 1c) into gray intervals which cover both object voxel and non object voxel. The hull of the gray intervals was used in the filter step to generate a candidate set. In the refinement step the voxel set covered by a gray interval was evaluated to avoid false hits. The disadvantage of this approach is that the filter step has a rather bad selectivity because much dead space is covered by the gray intervals.

In this paper, we propose a totally new decompositioning approach of voxelized objects based on clustering. Thus, we decompose the spatial objects directly in the original 2D/3D space without linearizing the voxels before by means of space filling curves.

3. Clustering based Object Decompositioning

In the following, we assume that the geometry of a spatial object is described by a set of voxels which in 2D are also known as pixels. Throughout the remainder of this paper, we assume a 3D data space.

Definition 1 (Voxelized Objects)
Let O be the domain of all object identifiers and let $id \in O$ be an object identifier. Furthermore, let IN^3 be the domain of 3-dimensional points. Then, we call a pair $O_{voxel} = (id, \{v_1, ..., v_n\}) \in O \times 2^{N^3}$ a 3-dimensional voxelized object. We call each of the v_i an object voxel, where $i \in \{1, .., n\}$. By v_x, v_y, and v_z we describe the corresponding coordinates of a voxel v.

The idea of our decompositioning approach is to apply the rather simple and well-known *k-means* clustering algorithm [8] to our voxel set. The *k*-means algorithm can be regarded as a simplified version of the more general EM algorithm [1] which describes a dataset by multiple Gaussian distribution functions. In our approach, we regard each voxel as a 3-dimensional feature vector. The clustering algorithm *k*-means divides each voxelized object $o = (id, \{v_1, ..., v_n\})$ into a set of k clusters $C_1^o, .., C_k^o$. Each cluster C_i^o contains a voxel set V_i^o. For these k voxel sets the two following properties hold:

- $\forall i, j \in (1 \ldots k): (i \neq j \Rightarrow (v \in V_i^o \Rightarrow v \notin V_j^o))$
- $\bigcup_{i=1 \ldots k} V_i^o = \{v_1, \ldots, v_n\}$

Each cluster C_i^o is represented by a *centroid* $\overrightarrow{C_i^o}$.

$$\overrightarrow{C_i^o} = \frac{1}{|V_i^o|} \left(\sum_{v \in V_i^o} v_x, \sum_{v \in V_i^o} v_y, \sum_{v \in V_i^o} v_z \right)^t$$

Each voxel of the object is thereby assigned to the closest centroid. An iterative control strategy is used to minimize the squared distances of the voxels to the centroids:

$$sqerr^o = \sum_{i=1}^{k} \sum_{v \in V_i^o} L_2\left(v, \overrightarrow{C_i^o}\right)^2$$

The algorithm starts with a random partition and iteratively reassigns the voxels to the centroids based on the distances between the voxels and the centroids until a convergence criterion is met. For example, the iteration may stop when no voxels are reassigned from one centroid to another one any more, or when the squared error $sqerr^o$ of the clustering ceases to decrease significantly, or after a maximum number of iterations has been performed. Advantages of the *k*-means clustering algorithm are that it is easy to understand and to implement and that the runtime complexity is $O(n \cdot k \cdot l)$ for n voxels, k clusters and l iterations.

The accuracy of our decompositioning algorithm is influenced by the chosen parameters, i.e. the value of k and the initial centroids. In [3] a method based on sampling is introduced which helps to chose these parameters appropriately.

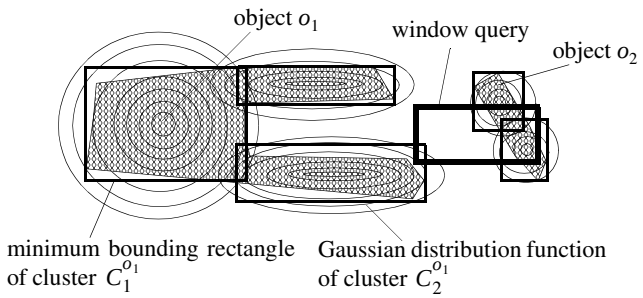


Figure 2. Gaussian distribution functions.

The distribution of the voxels within one cluster C_i^o can accurately be described by the centroid $\vec{c}_i^o = (c_{i,x}^o, c_{i,y}^o, c_{i,z}^o)$ and the standard deviations $\sigma_{i,x}^o$, $\sigma_{i,y}^o$, $\sigma_{i,z}^o$. These standard deviation values can be estimated by using the following formula[1].

$$\sigma_{i,x}^o = \sqrt[2]{\frac{1}{|V_i^o| - 1} \sum_{v \in V_i^o} (v_x - c_{i,x}^o)^2}$$

Obviously, we could also describe the distribution of the voxels V_i^o by means of their covariance matrix Cov_i^o. In order to reduce the storage cost per cluster, we refrain from this more complex description and store only the centroids and the standard deviation values. Thus, we approximate a voxel set V_i^o of a cluster C_i^o by an axis-parallel Gaussian distribution function.

Figure 2 shows an example for describing two 2-dimensional voxelized objects o_1 and o_2 by means of 3 and 2 axis-parallel Gaussian distribution functions, respectively. Note that the problem of finding an appropriate value of k is part of active research in the data mining community. We can benefit from these results for solving the problem of finding the right trade-off between accuracy and redundancy in the case of spatial object decompositioning.

Figure 2, furthermore, shows the minimum bounding rectangles (MBR) of the clusters. These MBRs can be used throughout the filter step to detect true misses. Nevertheless, in the example in Figure 2, for both objects o_1 and o_2 a rather expensive refinement step on the exact voxel representation is necessary to decide whether the objects belong to the result set or not.

We propose to store the MBRs of an object in an R-tree [5] or one of its variants. In order to increase the efficiency of the refinement step, we do not store the exact voxel sets V_i^o of a cluster C_i^o. We only store the centroid values $c_{i,x}^o$, $c_{i,y}^o$, $c_{i,z}^o$ and the standard deviations $\sigma_{i,x}^o$, $\sigma_{i,y}^o$, $\sigma_{i,z}^o$ along with the MBRs of the cluster. Based on this statistical information, we can compute how likely an intersection between an object and the query object is without accessing the detailed information provided by the voxel set. In Figure 2, for instance, we will detect that the probability that o_1 is intersected by the window query is below 50%. On the other hand, based on the Gaussian distribution functions we can compute that it is very likely that o_2 belongs to the result set. In the following section, we will formally introduce this approach.

1. Likewise, we compute the values $\sigma_{i,y}^o$ and $\sigma_{i,z}^o$

4. Fuzzy Intersection Query

The idea to avoid the expensive refinement step, is to assign to each database object represented by an object dependent number of k clusters a probability value indicating the likelihood that the object belongs to the result set. Traditional intersection queries assign a value 1 or 0 to each object in the database indicating whether the object belongs to the result or not. We propose fuzzy intersection queries which order all database objects according to their intersection probability.

Definition 2 Fuzzy Intersection Query

Let DB be a database of voxelized decomposed objects, and let q be a query object. Furthermore, let $p(q \cap o)$ denote the probability that q intersects the object $o \in DB$. Then, the fuzzy intersection query $fuzzy_\cap$: $1..|DB| \rightarrow DB \times [0,1]$ is a function which ranks all objects $o \in DB$ according to their probabilities $p(q \cap o)$, i.e.

$$\forall i, j \in 1..|DB|:$$
$$i<j \wedge fuzzy_\cap(i)=(o_i, p(q \cap o_i)) \wedge fuzzy_\cap(j)=(o_j, p(q \cap o_j)) \Rightarrow$$
$$o_i \neq o_j \quad \wedge \quad p(q \cap o_i) \geq p(q \cap o_j)$$

Note that the result of such a fuzzy intersection query does not necessarily contain less information than the traditional result set. For instance, in Figure 2 the traditional approach only detects that object o_1 does not intersect the window query. Our approach might assign a probability value $p(q \cap o_1) = 0.28$ indicating that the object is quite close to the query object. We suggest that this probability value can be regarded as a meaningful measure for describing the closeness between database objects and query objects. Thus, the fuzzy intersection queries might even provide more information than the binary result *intersection* or *non-intersection.*

The crucial question is now how to compute the probability that an object belongs to a result set. In this short paper, we will exemplarily demonstrate how to compute this value for box volume queries (cf. Figure 2) which are commonly used in many applications, e.g. GIS or CAD applications.

First, we must compute for each cluster C_i^o of an object o the probability that at least one voxel of the corresponding voxel set V_i^o intersects the query. Thereto, we first determine for each dimension $\Delta \in \{x, y, z\}$ individually the probability $p_{i,\Delta}^o$ that the query interval $[l_\Delta, u_\Delta]$ intersects the one-dimensional Gaussian distribution with center $c_{i,\Delta}^o$ and standard deviation $\sigma_{i,\Delta}^o$ (cf. Figure 3). By means of the standard Gaussian cumulative distribution function, conventionally denoted by Φ, we can compute the probability $p_{i,\Delta}^o$ straightforward by the following equation:

$$p_{i,\Delta}^o = p(c_{i,\Delta}^o, \sigma_{i,\Delta}^o, l_\Delta, u_\Delta) = \Phi\left(\frac{u_\Delta - c_{i,\Delta}^o}{\sigma_{i,\Delta}^o}\right) - \Phi\left(\frac{l_\Delta - c_{i,\Delta}^o}{\sigma_{i,\Delta}^o}\right)$$

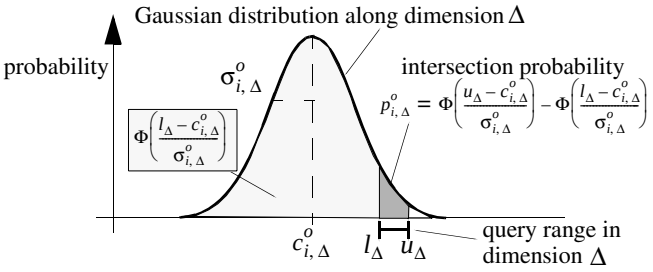


Figure 3. Intersection probability $p^o_{i,\Delta}$ of a query interval $[l_\Delta, u_\Delta]$ and a cluster C^o_i in dimension Δ.

For the voxel set belonging to the current cluster, we can assume that the three dimensions are independent from each other. Note that this assumption has to hold only for the object voxels of one cluster and not for all voxels of an object. Based on this legitimate assumption, we can easily compute the probability p that the box query intersects the axis-parallel 3-dimensional Gaussian distribution which describes the cluster C.

Lemma 1. Let $q = [l_x, u_x]\times[l_y, u_y]\times[l_z, u_z]$ be a 3-dimensional box query. Furthermore, let C^o_i be a cluster having $|V^o_i|$ many voxels which can be described by a 3-dimensional Gaussian distribution function with centroid $\vec{C}^o_i = (c_{i,x}, c_{i,y}, c_{i,z})$ and standard deviations $\sigma^o_{i,x}$, $\sigma^o_{i,y}$, $\sigma^o_{i,z}$. Then, we can compute the probability that at least one voxel of the cluster C^o_i is within the box query by:

$$p(q, C^o_i) = 1 - \left(1 - \prod_{\Delta \in \{x,y,z\}} p(c^o_{i,\Delta}, \sigma^o_{i,\Delta}, l_\Delta, u_\Delta)\right)^{|V^o_i|}$$

Finally, we can state the following lemma.

Lemma 2. Let $q = [l_x, u_x]\times[l_y, u_y]\times[l_z, u_z]$ be a 3-dimensional box query. Furthermore, let o be a voxelized object which is decomposed into k clusters $C^o_1, ..., C^o_k$. Then, we can compute the probability $p(q, o)$ that q intersects o by:

$$p(q, o) = 1 - \prod_{i=1}^{k} (1 - p(q, C^o_i))$$

Obviously, based on this probability value, we can answer the fuzzy intersection query introduced in Definition 2. We traverse the R-tree as usual. On the leaf level of the R-tree, we compute the probability values for each cluster (cf. Lemma 1) and combine these probability values to object probability values according to Lemma 2. Note that if we find one cluster which intersects our window query with high probability, the corresponding object probability value will be close to 1. On the other hand, to those objects for which we have not detected any clusters throughout the tree traversal, we assign a probability value of 0.

5. Conclusion

In this short paper, we sketched a new approach which helps to find an optimal trade-off between complexity and redundancy of object approximations. The proposed decompositioning algorithm is based on the well-known clustering algorithm *k*-means. Thus a voxelized object is described by *k* clusters. We describe each of these clusters by an axis-parallel 3-dimensional Gaussian distribution function and a minimum bounding rectangle of the cluster voxels. We store these minimum bounding rectangles along with statistical information in standard index structures. During query processing, we propose to omit the expensive refinement step on the exact voxel representations. Instead, we assign a probability value to each database object indicating how likely it belongs to the result set. The corresponding probability values are computed by exploiting statistical information describing the multivariate Gaussian distribution functions. In a give-me-more manner the user receives the objects which most likely belong to the result set.

Our first experiments showed that we can accelerate intersection queries considerably while still achieving high quality results. In our future work, we plan a detailed experimental evaluation demonstrating the characteristics and benefits of our fuzzy decompositioning approach.

References

[1] Dempster A.P., Laird N.M., and Rubin D.B.: *Maximum Likelihood from Incomplete Data via the EM algorithm.* Journal of the Royal Statistical Society, Series B, 39(1):1-31, 1977.

[2] Faloutsos C., Jagadish H. V., Manolopoulos Y.: *Analysis of the n-Dimensional Quadtree Decomposition for Arbitrary Hyperrectangles.* IEEE TKDE 9(3), 1997, 373-383.

[3] Fayyad U., Reina C., Bradley P.: *Initialization of Iterative Refinement Clustering Algorithms.* KDD 1998.

[4] Gaede V.: *Optimal Redundancy in Spatial Database Systems.* SSD 1995, pp. 96-116.

[5] Guttman A.: *R-trees: A Dynamic Index Structure for Spatial Searching*. SIGMOD1984, pp. 47-57.

[6] Kriegel H.-P., Pfeifle M., Pötke M., Seidl T.: *Spatial Query Processing for High Resolutions*. DASFAA 2003.

[7] Kriegel H.-P., Pötke M., Seidl T.: *Interval Sequences: An Object-Relational Approach to Manage Spatial and Temporal Data.* SSTD 2001, pp. 481-501.

[8] McQueen J.: *Some Methods for Classification and Analysis of Multivariate Observation.* Proc. 5th Berkeley Symp. on Math. Statist. and Prob., Vol. 1, 1965.

[9] Orenstein J. A.: *Spatial Query Processing in an Object-Oriented Database System*. SIGMOD 1986, pp. 326-336.

[10] Schiwietz M., Kriegel H.-P.: *Query Processing of Spatial Objects*: *Complexity versus Redundancy.* SSD 1993, pp. 377-396.

Spatial Search of Orbital Swath Data

Ross S. Swick and Kenneth W. Knowles
National Snow and Ice Data Center
Cooperative Institute for Research in Environmental Sciences
University of Colorado, Boulder, CO
swick@nsidc.org, knowlesk@nsidc.org

Abstract

The high volume of today's remotely sensed Earth Science data creates a strong motivation to minimize the amount of data delivered to the end user. The goal is to get users everything they need but nothing they don't need. One way to decrease the amount of unneeded data delivered is to increase spatial search accuracy. Unfortunately, the most voluminous data, orbital swath data, is also the most difficult to run spatial searches on. Three reasonably accurate means for spatially searching orbital swath data are: methods employing lookup tables, methods employing orbit propagators, and backtrack orbit searching. This paper outlines these three types of methods and discusses some of the advantages and disadvantages of each.

1. Introduction

Many satellites circle the globe, continuously imaging and collecting Earth Science data. These ribbons of remotely sensed data, which wrap around and around the Earth, are called swaths. For convenience, the continuous swaths are split into individual orbits that begin and end where the satellite crosses the equator from south to north. Typical polar orbiting satellites (meaning their orbits go nearly over each pole) will make 14 to 15 orbits per day, with each orbit comprising many megabytes of data. In general, only a few of these orbits will cover a researcher's study area. Earth scientists requesting data need some means to find these matching orbits. To aid the researcher, most data producers provide spatial search of their inventory.

The first problem is to define the area covered by the swath. As provided, most swath data does not contain an explicit description of the covered area. This information is usually computed upon ingest into the database. The simplest and most popular spatial search method is to define both search areas and data coverage areas as latitude, longitude (lat/lon) bounding boxes on a flat Earth. Latitude and longitude is a common spatial reference system for the Earth and all swath data users should be familiar with its use. (All methods discussed here are projection neutral.) A lat/lon bounding box simplifies spatial search because each area can be completely defined with only four numbers and area intersections can be determined by appropriate boolean comparisons of the minima and maxima. The popularity of this method is understandable because it's easy to implement and sufficient for most purposes. Orbital swath data is an extraordinarily bad fit to a lat/lon bounding box. A single orbit swath from a sensor with a sufficiently wide field of view can easily cover all latitudes and all longitudes while covering only a small portion of the Earth. Spatial search of orbital swath data requires special methods to accurately and efficiently compare search areas to data coverage areas.

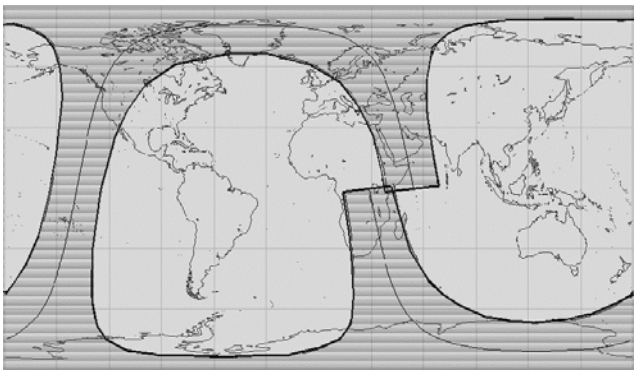


Figure 1. Typical swath coverage of a polar orbiting satellite shown on a flat Earth.

2. Lookup table methods

The idea behind lookup table methods is that while spatial search of swath data is difficult, spatial search of other areas, of a different spatial type, isn't. So, if we can

convert the swath into smaller areas that are easier to work with the problem is more manageable. The actual spatial search becomes a preliminary search on the smaller areas in the lookup table, which returns area IDs that match the area of interest. Spatial search of the data is then turned into a search on the set of IDs.

The Worldwide Reference System (WRS-1 & WRS-2) used by Landsat is one such method [5]. WRS-1 and WRS-2 are custom coordinate systems based on Landsat orbits. For WRS-1 252 "paths" are defined that correlate with the 252 orbits in the 18-day repeat cycle. Then, each path is broken into 248 "rows" that each represent 25 seconds, or about 1.45 degrees, of coverage in the path. This scheme yields 62,496 areas to search on in the lookup table and each Landsat scene in the inventory is tagged with the path and row coordinates it covers. The path and row coordinate system is similarly defined for WRS-2 based on a shorter 16-day repeat cycle.

The Nominal Orbit Spatial Extent (NOSE) scheme used by NASA's Earth Observing System (EOS) is an attempt to extend the WRS concept to other satellites and other sensors [2]. For satellites without a well defined and enforced repeat cycle, a typical implementation of NOSE will simply create 360 nominal orbits, one per degree, called "tracks". Each track will then be broken into 36 "blocks" that each represents about 2.7 minutes, or 10 degrees, of coverage in the track. This scheme yields 12,960 smaller areas to search on in the lookup table and each granule in the inventory is tagged with the track and block coordinates it covers.

Because both WRS and NOSE rely on custom coordinate systems derived from the orbital characteristics of the satellite and the field of view of the sensor, they end up being sensor specific schemes. This means the work involved in creating the lookup table in the first place has to be repeated for each new sensor, and sometimes even for the same sensor on a different satellite.

A more general approach is to break the Earth into areas that are independent of the data being collected. A simple approach is to use 1x1 degree lat/lon bins, which yields 64,800 smaller areas to search on in the lookup table. Because those bins are generic the lookup table can be reused for swath data from other sensors, and even for other kinds of data. Indeed, one advantage of a binning scheme is that it can be used for all types of data, which means the data provider can have a single spatial search mechanism for all its data. Both accuracy and performance will suffer as a result, but the trade-off may be worth it in some cases.

Lookup table schemes all require some preprocessing of the data to determine which IDs each granule in the inventory should be tagged with. But, the main disadvantage is both accuracy and performance suffer. Landsat has an advantage in that the orbit is tightly controlled and any drift in the satellite is periodically corrected. So, even with only 252 paths defined, the accuracy is much better than the (360°/252=) 1.43 degrees one would expect. Moreover, the performance hit associated with using the lookup table in a preliminary search can often be avoided because Landsat users have become so accustomed to the WRS scheme they often search directly on the path and row coordinates they already know cover their area of interest.

Satellites in less tightly controlled orbits must rely on the more general NOSE scheme or a simple binning scheme. With 360 nominal orbits defined, or bins defined as 1x1 degree boxes, accuracy is limited to 1 degree, or 111 km at the equator. Greater accuracy requires a larger lookup table, which further degrades performance.

There are schemes that use bins of variable size, hierarchical nesting, and more efficient search methods, for example, Peano keys, Morton codes, R-trees, spherical quad-trees, etc., which may partially mitigate the performance issues. However these schemes require the swath coverage to be computed, either by a lat/lon bounding box, nominal orbit, or orbit propagator. We have already discussed the problems with the lat/lon bounding box. Once you have found a nominal orbit or defined the coverage with an orbit propagator, as we shall see, it's no longer an area comparison problem.

3. Orbit Propagator Methods

Orbit propagators have been around for quite a while because they are useful for more than just spatial search. They are primarily used to simply track and predict satellite movements and using them for search purposes evolved naturally from that. Systems that use propagators to run spatial search include: NOAA's Comprehensive Large Array-data Stewardship System (CLASS), The University of Wisconsin-Madison's Man computer Interactive Data Access System (McIDAS), and The University of Alabama-Hunstville's Space Time Toolkit (STT).

The idea behind using orbit propagator methods for spatial search is that while spatial search of swath data is difficult, temporal search isn't. Orbit propagators work by using an orbit model to determine when the satellite was over the area of interest [1]. Given a time range for the search the propagator will initialize with an ephemeris file that defines exact position, speed, and heading of the satellite prior to the start of the time range. The propagator then spins the model forward to predict when the sensor saw the area of interest. Any cumulative error is corrected by periodically re-initializing the system with known ephemeris data. Consequently the spatial search is turned into a temporal search and since the temporal coverage of each granule is already in the inventory table no database changes are required.

Once an orbit propagator is integrated into a system the addition of new satellites, and new sensors, is relatively easy. Because they use actual ephemeris data and reinitialize periodically, orbit propagators can be quite accurate. Because they turn spatial search into temporal search, orbit propagators are ideally suited for use in coincident search. And, both the work of the propagator and the subsequent search are fast enough for short time ranges that performance is not an issue.

The main disadvantage of orbit propagator methods is that performance is inversely related to the time range searched. Earth scientists studying climate change are often interested in data over a time range of several years or even decades and at these time scales performance drops dramatically. No matter how fast the propagator may be it still has to spin through the entire time range, which means the time it takes the propagator to complete its task is directly proportional to the time range searched. More importantly, the end result is a large set of disjoint time ranges to search on, which can degrade database performance considerably.

For example, Thule, Greenland is located fairly far north and sensors on board polar orbiting satellites see Thule frequently. Many sensors see Thule ten times a day, which results in a query for a single day's worth of data like the one below. A request for a year's worth of data over Thule would result in 3,650 disjoint time ranges to search on. Obviously, the problem is less extreme at lower latitudes where coverage is less frequent. But, even at a point near the Equator, where coverage may be only twice daily, a request for only a year's worth of data would result in a search on 730 disjoint time ranges.

```
where (
('2003/02/18 01:23:03' between startDateTime and endDateTime) or
('2003/02/18 03:03:57' between startDateTime and endDateTime) or
('2003/02/18 04:44:34' between startDateTime and endDateTime) or
('2003/02/18 06:05:21' between startDateTime and endDateTime) or
('2003/02/18 07:46:32' between startDateTime and endDateTime) or
('2003/02/18 12:47:46' between startDateTime and endDateTime) or
('2003/02/18 14:10:32' between startDateTime and endDateTime) or
('2003/02/18 15:51:02' between startDateTime and endDateTime) or
('2003/02/18 17:11:31' between startDateTime and endDateTime) or
('2003/02/18 18:52:07' between startDateTime and endDateTime)  )
```

Figure 2. The temporal clause created by an orbit propagator to search for data over Thule on February 18, 2003.

4. Backtrack Orbit Search

The Backtrack Orbit Search Algorithm (BOSA) is currently in use at the National Snow and Ice Data Center (NSIDC) and NASA's EOS ClearingHOuse (ECHO). The idea behind backtrack orbit search is that while spatial search of swath data is difficult in general, Earth Science swath data has a number of characteristics that make the task a lot easier. Remotely sensed data is valuable to Earth scientists because it is frequent, regular, and global. For the purposes of doing Earth Science, scientists have an interest in keeping the data as consistent as possible. Among other things, that means they want the sensor to have a constant field of view. An easy way to accomplish that is to put the satellite in a circular orbit. For this reason (and others), all Earth Science satellites are in a circular orbit.

The Backtrack Orbit Search Algorithm [4] exploits this fact to greatly simplify the orbit model by just modeling an orbit as a great circle under which the Earth rotates. The simplicity of the model allows backtrack to be more efficient than orbit propagator methods, which are designed to work with any satellite. The simplified orbit model relies on only three parameters: inclination, period, and swath width. The accuracy of the method depends on the stability of those three parameters over the life of the sensor, but there is also a scientific interest in keeping those parameters stable, so they generally stay within reason, or the data aren't useful.

As the name implies, backtrack works by tracing the orbit backwards. Backtrack starts with the area of interest and answers the question "In order for the sensor to have seen this area, where must the satellite have crossed the equator?" There is no time dependence, so the speed of the algorithm is independent of the time range searched. There is no cumulative error because backtrack backs up at most one orbit. There is no performance hit from using a lookup table because backtrack calculates the actual equatorial crossing range. And, the subsequent search is a simple, fast, boolean search on that crossing range rather than a text search on IDs. Shown below is one such search statement for 20 year's worth of AVHRR data.

```
select distinct file_ID, start_date, data_set, ascending_crossing
from inventory where data_set like "AVHRR_LAC" and
start_date >= 19800101 and end_date <= 20000218 and
(ascending_crossing between -80.0 and -64.0)  or
(ascending_crossing between 88.0 and 106.0)  )
```

Figure 3. A query for 20 years worth of data created using Backtrack.

The main disadvantage of backtrack is that the crossings do have to be in the database to be searched on. Those values are generally known but because they are not generally used they don't often get stored in the database. So, a change to the inventory table will usually be required.

5. Summary

Lookup table methods provide reasonably fast search once the table parameters are defined, calculated, and stored in the database. New satellites or sensors require

new parameter schemes. Both search speed and accuracy are determined by the size of the individual bins in the table.

Orbit propagator methods are the most accurate means of searching. However, the search speed is proportional to the length of the time interval of interest. This penalty is paid twice, once in building the query, and once in executing the search. The ancillary ephemeris data is unique to each satellite and requires frequent updating.

Backtrack orbit searching is fast and accurate. However, it is restricted to sensors with a constant field of view on satellites in circular orbits. The ancillary orbit parameters are unique to each satellite and sensor, but only need be obtained one time. Accuracy depends on the stability of those parameters. The crossing times must be added to the database.

Throughout this paper we have assumed single orbit granules. Each of the methods described requires some adjustment for partial or multiple orbit granules—some more than others.

The choice of which method to use largely depends on the purpose of the system. For the purposes of searching inventories of orbital Earth Science data we highly recommend backtrack as faster, cheaper, and more efficient than any other method. Moreover, the accuracy of backtrack orbit search is competitive with orbit propagator methods without the attendant performance issues. We realize this is a qualitative analysis. Further quantitative tests need to be done.

6. References.

[1] Bates, Roger R., Donald D. Mueller, and Jerry E. White, *Fundamentals of Astrodynamics*, Dover Publications, New York, 1971.

[2] Heroux, David. "Leveraging Nominal Orbit Spatial Extent to Provide a Solution for the Archival of Geoscience Laser Altimeter System (GLAS) Products in ECS", http://edhs1.gsfc.nasa.gov/waisdata/sdp/pdf/tp1601401.pdf, December, 2001.

[3] Swick R. and K. Knowles, “Spatial Tools for a Round Planet”, Poster #OS51B-166 at the American Geophysical Union Fall Meeting, San Francisco, December 6-10, 2002.

[4] Swick, R., and K. Knowles. "The Backtrack Orbit Search Algorithm", http://www.geospatialmethods.org/bosa, November, 2004.

[5] Williams, Daryl. "The Worldwide Reference System (WRS)", http://landsat.gsfc.nasa.gov/documentation/wrs.html, May, 2004.

Panel Discussion

NSF Long Term Ecological Research Sites
Praxis et Theoria – LTER Information Management & CS Research

Panel Organizers

Judith Cushing
The Evergreen State College
judyc@evergreen.edu

Kristin Vanderbilt
Sevilleta LTER - University of New Mexico
vanderbi@sevilleta.unm.edu

Panelists

James Brunt
LTER Network Office - University of New Mexico
jbrunt@lternet.edu

Amarnath Gupta
San Diego Supercomputer Center
gupta@sdsc.edu

Matt Jones
National Center for Ecological Analysis and Synthesis
University of California at Santa Barbara
jones@nceas.ucsb.edu

Peter McCartney
Central Arizona – Phoenix LTER
Arizona State University
Peter.mccartney@asu.edu

Abstract

Information managers at each of the 26 sites in the Long Term Ecological Research (LTER) Network face many challenges managing structurally and thematically diverse ecological data sets collected by LTER scientists. Since the LTER's inception in 1980, LTER information managers have partnered with computer scientists to seek innovative information management solutions. This panel of three computer scientists and three LTER informatics scientists will present current collaborative research projects and offer perspectives on engendering mutually beneficial partnerships among research ecologists, the eco-informatics community, supercomputer centers, and computer science researchers. A discussion with SSDBM attendees about DB and CS research areas that might benefit the LTER information management mission will follow panelist remarks.

1. Introduction

In 1980 the National Science Foundation (NSF) set aside $1.5 million in research funds to support six Long Term Ecological Research (LTER) sites [3], a budget which has now grown to $20 million per year to support 26 sites. All LTER sites accumulate long term data on common themes including plant and animal population dynamics, biogeochemistry and primary productivity. Some sites are operated in conjunction with government agencies (e.g., US Forest Service) and have data extending back many decades.

Since the program's inception, NSF has charged LTER scientists with preserving and making data from ecological studies publicly available so that retrospective research and network-level data syntheses could be more easily performed. Most LTER sites presently employ an information manager (IM) to facilitate site and inter-site science through informatics, and to support the databases and data repositories necessary to fulfill the NSF data mandate. LTER information managers (IM's) have cooperatively developed metadata standards, inter-site databases, information management evaluation criteria, *Databits* (a publication on LTER informatics issues http://lternet.edu/documents/Newsletters/DataBits), and informatics training modules. LTER IM's also assist each other with information management issues at LTER sites. With 20+ years in managing information for ecology researchers, the LTER IM's have significant insight into the problems and issues that arise from acquiring, documenting, validating, publishing, and analyzing long-term ecological data.

Information managers and ecologists are faced with many challenges when developing tools to manage and analyze the ever-increasing volume and diversity of data being collected at LTER sites. Eco-informatics research [9] has also been identified by NSF as critical (http://www.evergreen.edu/bdei). This panel of three LTER information managers and three computer scientists will address the computer science research that might increase the effectiveness of LTER information management and ecological research. LTER panelists will 1) provide relevant background and history of the LTER; 2) address institutional collaborations between the LTER Network Office and supercomputer centers and national labs; and 3) describe the evolving nature of informatics research as a partnership between ecologists and computer scientists. Computer scientists on the panel will discuss projects they have conducted with LTER information managers and ecology researchers, and the research problem was defined so that CS partners could help provide IT solutions while contributing to the real-world problems faced by LTER IM's. Research projects include software component development, analysis and management of real-time sensor web data, and cyber-

infrastructure for ecological research. Perspectives on how to make successful collaborations among information managers, domain scientists, and computers scientists will be shared. After panelist remarks, panel organizers (one LTER Information Manager and one CS researcher) will moderate a discussion with SSDBM attendees on CS and DB research areas that might reap particular benefit to the LTER mission, how attendees might approach collaboration with LTER, how to balance *theoria* and *praxis* [5] in a successful collaboration, and what benefit to the computer science research community might follow.

2. Statements from Panelists

2.1 LTER Information Managers

2.1.1 Kristin Vanderbilt: Introduction to the LTER.

The Long Term Ecological Research (LTER) Network was established in 1980 with six sites, and has since grown to include 26 sites and 1800 scientists and students studying environmental phenomena that span multiple temporal and spatial scales. LTER sites encompass diverse ecosystems such as tropical rainforest, temperate deciduous forest, arctic tundra, lakes, coral reefs, estuaries, salt marsh and deserts. The five core areas of research that all LTER sites address are: 1) patterns of net primary production, 2) distribution and dynamics of populations, 3) organic matter accumulation and decomposition, 4) inorganic nutrient cycling through water, soils, and sediments, and 5) frequency and pattern of disturbance. The mission of the LTER Network is to promote synthesis and comparative research across sites and ecosystems and among other related national and international research programs.

LTER information managers have a diverse collection of physical, chemical, biological, and human-related data, often spanning many years, and must develop solutions for managing and creating linkages among these disparate data. LTER information managers have long worked cooperatively to solve IM problems. The LTER IM Committee, comprised of information managers for each site, has evolved from a handful of individuals who met irregularly during the 1980's to a highly organized group who have become leaders in the field of ecological information management. This group has actively standardized information management practices to facilitate data exchange and network-wide data integration. Many information managers have also sought partnerships with computer scientists to further enable LTER information management capabilities at the site level. The Sevilleta LTER information manager, for example, partnered with computer scientists at the University of New Mexico in the early 1990's to develop workflow software for image analysis.

More recently, collaborations between computer scientists and the LTER information management community have begun to occur at the LTER network level. In 1996, the proposal for the LTER Network Office (LNO), the office that coordinates LTER site activities, included the San Diego Supercomputer Center (SDSC) as a partner. The Spatial Data Workbench [8] resulted from this collaboration (http://sdw.sdsc.edu/index.html). A collaboration between SDSC, National Center for Ecological Analysis and Synthesis (NCEAS), and LNO yielded the Knowledge Network for Biocomplexity (http://knb.ecoinformatics.org/), which aimed to integrate data from distributed, autonomous data repositories [4]. The current SEEK (Science Environment for Ecological Knowledge [7]) collaboration includes computer scientists, LTER information managers, and domain scientists from several institutions. LNO is developing a Network Information Management system [1], which has as one of its components the EcoGrid, an internet architecture for data storage, sharing, access being developed by SEEK.

At the SSDBM panel, Kristin Vanderbilt will present an overview of LTER, and introduce collaborative LTER-CS projects that other panelists will discuss more fully.

2.1.2 James Brunt: Building institutional bridges between ecology and supercomputer centers and industry.

The US Long Term Ecological Network has been successful at attracting collaborators and funding for informatics efforts supporting ecological science. Interest has been high in part because of the high profile of the LTER program at the National Science Foundation. Successful partnerships have been forged with national labs and supercomputer centers, and proposals in high profile NSF programs for Knowledge and Distributed Intelligence and Information Technology Research have been funded.

The above efforts although successful have not always directly addressed the research and development needs of the LTER Network. Partnerships on specific efforts with national labs and supercomputer centers have taken on a variety of symbiotic relationships not all of which were mutual. Successful research proposals required impacts that extend beyond the boundaries of LTER science and that were primarily new "research". LTER has taken new approaches that hinge on direct collaboration with LTER information scientists. Clarity of purpose and persistence are required to convince computing partners steeped in "service" architecture of the mutual benefits of this type of relationship. While LTER has been successful at collaborating on informatics proposals, the real challenge is (once research projects are complete) how to fund deployment and maintenance of cyber-infrastructure that meets immediate requirements for the advancement of

ecological synthesis but does not interest funding agencies as research.

James Brunt will describe prior research collaboration (KDI and ITR grants), and suggest how deployment of research results within the LTER could be effected to bring about benefit to both the LTER and the research community.

2.1.3 Peter McCartney: Academic sciences such as ecology face considerable challenges in developing and maintaining cyber-infrastructure to support research, education and decision making. Many users in the business, medical, or other high-profile research fields can rely on the software industry or labs such as Los Alamos or JPL to take on the challenge of bringing advances in computer science into everyday software tools. Ecology, on the other hand, relies on a mixed strategy of adapting commercial off-the-shelf tools originally designed for other market bases, developing partnerships with our computer science colleagues, and building computing expertise of practitioners within its discipline. We typically refer to this boundary specialization as *informatics* (often with an appropriate prefix such as eco-, geo-, bio-, etc) to recognize the specific challenge of applying information technology to a specific research domain. These efforts are often frustrated by the fact that the domain problems as presented by the practitioner may pose insufficient research challenges to many computing science (CS) students or faculty and, in turn, the advanced solutions produced by CS research cannot sometimes be implemented with the expertise and resources found within the domain science community.

To build more effective partnerships, we need solutions for promoting greater student collaboration in informatics as a boundary specialty where CS students can be encouraged and rewarded for taking on topics that are less about carrying out cutting edge research and more about embedding applications of that research within research domains, and where domain students can be encouraged and offered opportunities to become more proficient in the advanced information technologies upon which the future of their domain's information legacy rests.

Peter McCartney will discuss the above issues along with Arizona State University experiences engaging CS students in ecological and archaeological infrastructure projects and promoting interdisciplinary thesis projects between CS and other sciences.

2.2 Computer Science Researchers

2.2.1 Judy Cushing: Tools that increase researcher productivity and provide metadata as a by-product.

Discussions with LTER Information Managers at the H. J. Andrews LTER Site articulated metadata acquisition as a major barrier to long term archiving of ecological data. An individual researcher, however, has little incentive to spend time documenting data if that activity (typically completed at the end of an active research project) does not also improve his or her own work. If database technology were integrated earlier into the research process, however, metadata might be collected more "naturally" if rendering one's data and metadata into that technology also provided benefits such as data transformation or data visualization.

The Canopy Database Project [2] is a joint computer science and ecology project that facilitates end-user database development and programming through domain-specific database components. The database technology and companion software components aim to facilitate not only data documentation of, but also data validation, analysis and visualization that will provide immediate value.

Longer term computer science research goals are physical and semantic integration of disparate data sets, component-based and domain-specific software engineering, and end-user programming strategies, but these have been subordinate to building a corpus of data sets and tools that illustrate the breadth of domain-specific components that might promote researcher productivity.

Though originally intended for end users who are ecology researchers, a proof of concept experiment with LTER information managers led to insights about the project's corpus of components and tools, and suggests that a database generator and companion tools based on domain-specific components could be an efficiency tool for information managers.

Judy Cushing will address how ecological information management problems were translated into a computer science research project and the benefit of collaboration with the LTER information managers. She will talk about how that partnership might have been improved and completed, and present ideas for sharing software components between research projects and the LTER IM network, and for future research.

2.2.2 Matt Jones: The *Knowledge Network for Biocomplexity* (KNB) and *Science Environment for Ecological Knowledge* (SEEK). Both the KNB and SEEK projects involve innovative research in CS that is risky in terms of deployment in particular application areas over short time scales. One example of computer scientist-ecologist interactions from SEEK is where the Ecological Niche Modeling group meets with computer scientists in workshops to evaluate approaches and set requirements. These interactions, and others like them in KNB and SEEK, have engendered helpful lessons during the project about successful collaboration and suggest

future activities that would aid deployment of technologies to specific application areas and networks like the LTER.

Matt Jones will describe the real world eco-informatics problems and ecology research needs that led to KNB and SEEK, along with lessons learned during those projects.

2.2.3 Amarnath Gupta: Collaboration strategies for sensor observing system research.

Recently, a number of large-scale scientific projects have been undertaken under the general umbrella of "observing systems". In all these projects, a variety of sensors are placed in the environment and information from them is combined to gain a deeper understanding of natural physical and biological phenomena, and perhaps more importantly, to predict significant natural events, and the impact of anthropogenic disturbances.

In the Lakes project [6], conducted at North Temperate Lakes LTER site in Wisconsin and Yuan Yang Lake in Taiwan, buoys containing heterogeneous sensors are placed in different lakes. These sensors collect information on variables like water temperature at different depths, dissolved oxygen, light penetration, water velocity vectors in a water column, barometric pressure and wind speed, humidity and so forth. The information is often collected locally in a low-capacity on-buoy data store and transmitted to a central server in bursts. In addition, near-lake data are collected from other sensors for barometric pressure, air temperature and so on. Further, water samples are collected at different intervals, to measure inorganic content like total phosphorous, nitrogen and carbon, and organic content like chlorophyll concentration, phytoplankton and bacteria counts.

In this context, information integration plays a number of roles. First, for exploratory data analysis, some observed sensor data, some transformed sensor data and some interpolated non-sensor data need to be window-joined over multiple streams. In this process, the non-sensor data may have to be transformed into a stream, by passing it through a simulation model that has been developed by scientists. Since there is a need to combine real-time observed streams with somewhat delayed computed streams, there is potential for interesting CS research in the development of join algorithms in a data stream management system (DSMS). Secondly, integrated information is often used for the purpose of discovering patterns of model invalidity. In this problem the goal is to first determine when the observed values of a variable are different from predicted values in a statistically significant manner, and then to determine whether such discrepancies occur for some specific ranges of the non-predicted variables. In yet a third use case, complex aggregates are computed from different combinations of sensor and non-sensor data coming from different lakes in a system of nearby lakes and are statistically compared against one another to estimate the influence of a common external cause such as a thunderstorm passing over the lake system.

Amarnath Gupta will describe some of the data management challenges that he and colleagues have encountered in working with the Lakes project. He will also highlight how often he uses a three-party-collaboration across the domain scientists, their data managers and the computer scientists to gain a deeper sense of the scientific tasks and how these tasks translate to implementation needs, as well as new research problems. While this presentation will confine itself to our experience within the limnological domain, this interdisciplinary collaboration strategy has been equally effective in other domains of natural science.

3. References

[1] Baker, K.S., B.J. Benson, D.L. Henshaw, D. Blodgett, J.H. Porter, and S.G. Stafford, "Evolution of a multisite Network Information System: The LTER information management paradigm", *Bioscience*, 2000, Vol. 50, pp. 963-978.

[2] Cushing J.B., N. Nadkarni, B. Bond, and R. Dial, "How trees and forests inform biodiversity and ecosystem informatics", *Computing in Science and Engineering*, 2003, Vol. 5, pp. 32-43.

[3] Hobbie, J.E., S.R. Carpenter, N.B. Grimm, J.R. Gosz, and T.R. Seastedt, "The US Long Term Ecological Research Program", *Bioscience*, 2003, Vol. 53, pp. 21-32.

[4] Jones, M.B., C. Berkley, J. Bojilova, and M. Schildhauer, "Managing scientific metadata", *IEEE Internet Computing*, 2001, Vol. 5, pp. 59-68.

[5] Knuth, D., "Theoria and Praxis", Acceptance Speech for ACM-SIGCSE Award for Outstanding Contribution to Computer Science Education, SIGCSE Conference, 1986.

[6] *LTER Network Newsletter*, "New wireless sensor network for studying lake metabolism," Vol 17, pp 16-17. 2004.

[7] Michener, W.K., J.H. Beach, M.B. Jones, B. Ludaescher, D.D. Pennington, R.S. Pereira, A. Rajasekar, and M. Schildhauer, "A Knowledge Environment for the Biodiversity and Ecological Sciences", *Journal of Intelligent Information Systems*, In press.

[8] Vande Castle, J., D.D. Pennington, T. Fountain, C. Pancake, "A Spatial Data Workbench for Data Mining, Analyses, and Synthesis", Pages 420-424 in: N. Callaos, J. Porter, N. Rishe, editor(s). *SCI2002: The Proceedings of the Sixth World Multiconference on Systematics, Cybernetics, and Informatics*, July 14-18, 2002, Orlando, Florida, USA (Vol. VII), Volume: VII. Orlando: IIIS.

[9] Schnase, J. and J. Cushing (eds), Issue on Eco-Informatics, *Journal of Intelligent Information Systems.* In press.

Author Index