Query, Analysis, and Visualization of Hierarchically Structured Data using Polaris

Chris Stolte
Gates Building, Room 354 3A
Stanford, CA 94305
cstolte@graphics.stanford.edu

Diane Tang
Gates Building, Room 354 3A
Stanford, CA 94305
dtang@graphics.stanford.edu

Pat Hanrahan
Gates Building, Room 370 3B
Stanford, CA 94305
hanrahan@graphics.stanford.edu

ABSTRACT

In the last several years, large OLAP databases have become common in a variety of applications such as corporate data warehouses and scientific computing. To support interactive analysis, many of these databases are augmented with hierarchical structures that provide meaningful levels of abstraction that can be leveraged by both the computer and analyst. This hierarchical structure generates many challenges and opportunities in the design of systems for the query, analysis, and visualization of these databases.

In this paper, we present an interactive visual exploration tool that facilitates exploratory analysis of data warehouses with rich hierarchical structure, such as might be stored in data cubes. We base this tool on Polaris, a system for rapidly constructing table-based graphical displays of multidimensional databases. Polaris builds visualizations using an algebraic formalism derived from the interface and interpreted as a set of queries to a database. We extend the user interface, algebraic formalism, and generation of data queries in Polaris to expose and take advantage of hierarchical structure. In the resulting system, analysts can navigate through the hierarchical projections of a database, rapidly and incrementally generating visualizations for each projection.

1. INTRODUCTION

In the last several years, large OLAP databases have become common in a variety of applications. Corporations are creating large data warehouses of historical data on key aspects of their operations. International research projects such as the Human Genome Project [11] and the Sloan Digital Sky Survey [19] are generating massive scientific databases.

A major challenge with these data warehouses is to extract meaning from the data they contain: to discover structure, find patterns, and derive causal relationships. The sheer size of these data sets complicates this task: Interactive calculations that require visiting each record are not plausible, nor is it feasible for an analyst to reason about or view the entire data set at its finest level of detail. Imposing meaningful hierarchical structure on the data warehouse provides levels of abstraction that can be leveraged by both the computer and the analyst.

These hierarchies can come from several different sources. Some hierarchies are known *a priori* and provide semantic meaning for the data. Examples of these hierarchies are Time (day, month, quarter, year) or Location (city, state, country). However, hierarchies can also be automatically derived via data mining algorithms that classify the data, such as decision trees or clustering techniques. Part of the analysis task when dealing with automatically generated hierarchies is in understanding and trusting the results [23].

Visualization is a powerful tool for exploring these large data warehouses, both by itself and coupled with data mining algorithms. However, the task of effectively visualizing large databases imposes significant demands on the human-computer interface to the visualization system. The exploratory process is one of hypothesis, experiment, and discovery. The path of exploration is unpredictable, and analysts need to be able to easily change both the data being displayed and its visual representation. Furthermore, the analyst must be able to first reason about the data at a high level of abstraction, and then rapidly drill down to explore data of interest at a greater level of detail. Thus, the interface must expose the underlying hierarchical structure of the data and support rapid refinement of the visualization.

This paper presents an interactive visual exploration tool that facilitates exploratory analysis of data warehouses with rich hierarchical structure, such as would be stored in data cubes. We base this tool on Polaris [21], a system for the exploration of multidimensional relational databases. Polaris is built upon an algebraic formalism for constructing table-based visualizations. The state of the user interface is a visual specification. This specification is interpreted according to the formalism to determine both the series of queries necessary to retrieve the requested data, as well as the mapping and layout of the resulting tuples into graphical marks. Because every intermediate specification is valid and can be interpreted to create a visualization, analysts can rapidly and incrementally construct complex queries, receiving visual feedback as they assemble and alter the specifications.

The original version of Polaris did not directly support or expose hierarchically structured dimensions, instead presenting each level of the hierarchy as a separate, independent dimension. In this paper, we extend the algebraic formalism (Section 4), user interface (Section 5), and generation of data queries (Section 6) to take advantage of hierarchically structured data cubes. We then illustrate the ease and effectiveness of using Polaris to explore hierarchically structured data via three case studies (Section 7).

2. RELATED WORK

We consider two areas of related work: the visual exploration of databases and the use of data visualization in conjunction with data mining algorithms.

SIGKDD '02 Edmonton, Alberta, Canada

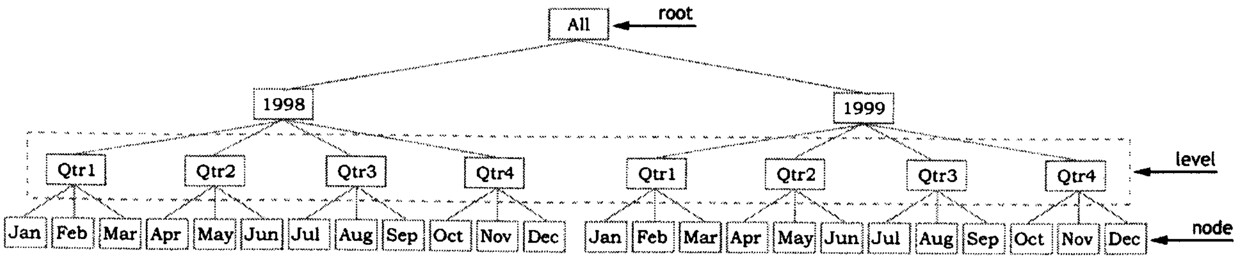


Figure 1: A hierarchical *Time* dimension. A hierarchical dimension is structured as a tree with multiple levels. In this case, there are four levels: *All, Year, Quarter,* and *Month*. Each level corresponds to a different semantic level of detail. The parent-child relationships in the tree are the basis for aggregation within the dimension.

2.1 Visual Exploration of Databases

One area of related work is the field of visual query tools. Projects such as VQE [5], Visage [17], DEVise [15], and Tioga-2 [27] have focused on building visualization tools that directly support interactive database exploration through visual queries. Users can construct queries and visualizations directly through their interactions with the interface. These systems have flexible mechanisms for mapping query results to graphs and support mapping database tuples to retinal properties of the marks in the graphs. Of these systems, only Tioga-2 provides built-in support for interactively navigating through and exploring data at different levels of detail. However, the underlying hierarchical structure must be created by the analyst during the visualization process; Polaris leverages the hierarchical structure that is already encoded in the data warehouse.

XmdvTool [25], Spotfire [20], and Xgobi [4] provide the analyst with a set of predefined visualizations such as scatterplots and parallel coordinates. These systems are augmented with extensive interaction techniques (e.g., brushing and zooming) that can be used to refine the queries. In contrast, we provide the analyst with a set of building blocks that can be used to interactively construct and refine a wide range of displays to suit the analysis process. Of these systems, only XmdvTool supports the exploration of hierarchically structured data. XmdvTool has been augmented with structure-based brushes [7] that allow the user to control the display's global level of detail (based on a hierarchical clustering of the data) and to brush records based on their proximity within the hierarchical structure. Again, this approach limits the user, in this case to viewing a single hierarchical structuring of the data and a single ordering of that hierarchy to make proximity meaningful. Polaris supports both the simultaneous exploration of multiple hierarchies (derived from semantic meaning or algorithmic analysis) and the ability to reorder the hierarchy as needed.

Another relevant visualization system, VisDB [13], focuses on displaying as many tuples as possible to provide feedback as users refine their queries. This system even displays tuples that do not satisfy the query, indicating their "distance" from the query criteria using spatial encodings and color. This approach helps the user avoid missing important data points that fall just outside of the selected query parameters. In contrast, Polaris, by taking advantage of the hierarchical structure of the warehouse, provides extensive ability to drill down and roll up data, allowing the analyst to get a complete overview of the data set before focusing on detailed portions of the database.

2.2 Visualization and Data Mining

Many research and commercial systems use visualization in conjunction with automated data mining algorithms. One common application of visualization together with data mining is in helping analysts understand models generated by the data mining process. For example, several researchers have developed techniques specifically for displaying decision trees, Bayesian classifiers, and decision table classifiers [1], and these visualization techniques have been incorporated into products such as SGI's MineSet [3].

Other approaches to coupling visualization and data mining have traditionally been employed within focused domains. One approach is to use visualization to gain an initial understanding of a warehouse and then apply algorithmic analysis to the identified areas of interest [14][23]. The other major approach is to use data mining to compress the size and dimensionality of the data and then use focused visualization tools to explore the results [10][26].

Unlike these examples, Polaris is not focused on a particular algorithm, a single phase of the discovery process, or a narrow application domain. Instead, Polaris is a general tool that can be used to gain an initial understanding of a warehouse, to visually mine the warehouse, to understand algorithm output, and to interactively explore a mining model. The ability to encode a large number of dimensions in a table layout in Polaris helps an analyst gain an initial understanding of how different dimensions relate as a precursor to automated discovery. Similarly, Polaris can be used directly as a visual mining tool. Finally, by integrating the decision trees and classification networks into the data warehouse as dimension hierarchies, Polaris can be used by analysts to gain an understanding of how these models classify the data.

3. BACKGROUND

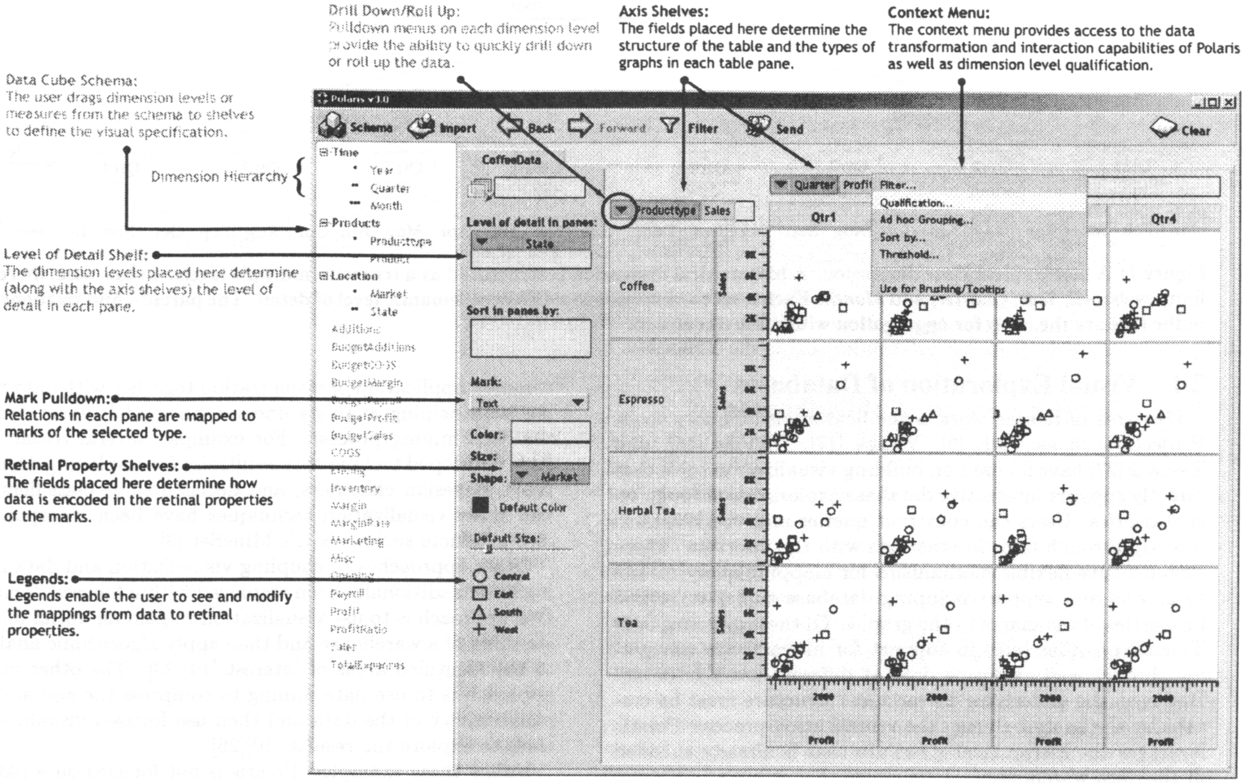
Polaris [21] was originally designed to support the interactive exploration of multidimensional relational data warehouses rather than data sets with rich hierarchical structure. In this section, we explain the difference between the two types of data sources as well as give a brief overview of Polaris before discussing our extensions to Polaris in the rest of the paper.

3.1 Relational Databases vs. Data Cubes

Relational databases organize data into *relations*, where each row in a relation corresponds to a basic entity or fact and each column represents a property of that entity [24]. For example, a relation may represent transactions in a bank, where each row corresponds to a single transaction, and each transaction has multiple properties, such as the transaction amount, the account balance, the bank branch, and the customer.

We refer to a row in a relation as a *tuple* or *record*, and a column in the relation as a *field*. A single relational database will contain many heterogeneous but interrelated relations.

The fields within a relation can be partitioned into two



Drill Down/Roll Up:
Pulldown menus on each dimension level provide the ability to quickly drill down or roll up the data.

Axis Shelves:
The fields placed here determine the structure of the table and the types of graphs in each table pane.

Context Menu:
The context menu provides access to the data transformation and interaction capabilities of Polaris as well as dimension level qualification.

Data Cube Schema:
The user drags dimension levels or measures from the schema to shelves to define the visual specification.

Dimension Hierarchy

Level of Detail Shelf:
The dimension levels placed here determine (along with the axis shelves) the level of detail in each pane.

Mark Pulldown:
Relations in each pane are mapped to marks of the selected type.

Retinal Property Shelves:
The fields placed here determine how data is encoded in the retinal properties of the marks.

Legends:
Legends enable the user to see and modify the mappings from data to retinal properties.

Figure 2: The Polaris user interface with enhancements (shown in blue) to expose and support hierarchical dimensions. Analysts construct table-based displays of relational and cube data by dragging dimension levels and measures from the data cube schema onto shelves throughout the display. A given configuration of levels and measures on shelves is called a visual specification. The specification unambiguously defines the analysis and visualization operations to be performed by the system to generate the display.

types: *dimensions* and *measures.* Dimensions and measures are similar to independent and dependent variables in traditional analysis. For example, the bank branch and the customer would be dimensions, while the account balance would be a measure.

In many data warehouses, these multidimensional databases are structured as n-dimensional data cubes. Each dimension in the data cube corresponds to one dimension in the relational schema. Each cell in the data cube contains all the measures in the relational schema corresponding to a unique combination of values for each dimension.

The dimensions within a data cube are often augmented with a hierarchical structure. This hierarchical structure may be derived from the semantic levels of detail within the dimension or generated from classification algorithms. Using these hierarchies, the analyst can explore and analyze the data cube at multiple meaningful levels of aggregation calculated from a base *fact table* (i.e., a relation in the database with the raw data). Each cell in the data cube now corresponds to the measures of the base fact table aggregated to the proper level of detail.

The aggregation levels are determined from the hierarchical dimensions; each dimension is structured as a tree with multiple *levels.* Each level corresponds to a different semantic level of detail for that dimension. Within each level of the tree there are many *nodes,* with each node corresponding to a value within the domain of that level of detail of that dimension. The tree forms a set of parent-child relationships between the domain values at each level of detail. These relationships are the basis for aggregation, drill down, and roll up operations within the dimension hierarchy. Figure 1 illustrates the hierarchy for a *Time* dimension.

Simple hierarchies, like the one shown in Figure 1, are commonly modeled using a *star schema.* The entire dimension hierarchy is represented by a single dimension table (also stored as a relation) joined to the base fact table. In this type of hierarchy, there is only one path of aggregation. However, there are more complex dimension hierarchies where the aggregation path can branch. For example, a *Time* dimension might aggregate from *Day* to both *Week* and *Month.* These complex hierarchies are typically represented using a *snowflake schema* that uses multiple relations to represent the diverging hierarchies.

When referring to values within a dimension hierarchy, we will use a dotted notation to specify a specific path from the root level (*All*) of the hierarchy down to the specified value. Specifically, to refer to a value on level m of a hierarchy, we first optionally list the dimension name, then zero or more of the $(m-1)$ intermediate ancestor values, and then finally the value on the m^{th} level, all separated by periods. For example, the *Jan* node on the *Month* level in the *Time* hierarchy that corresponds to January, 1998, can be referred to as *1998.Qtr1.Jan.* When this notation is used, we will call the reference a *qualified value.* When a value is simply described by its node value (without any path to the root node) we call the reference an *unqualified value.*

The hierarchical data model we have outlined in this section is only one of many possible models. Other models, such as that proposed by Jagadish et al. [12], include advantages such as structural and schematic heterogeneity. We have chosen to focus on the model commonly found in commercial data warehouse and data cube products.

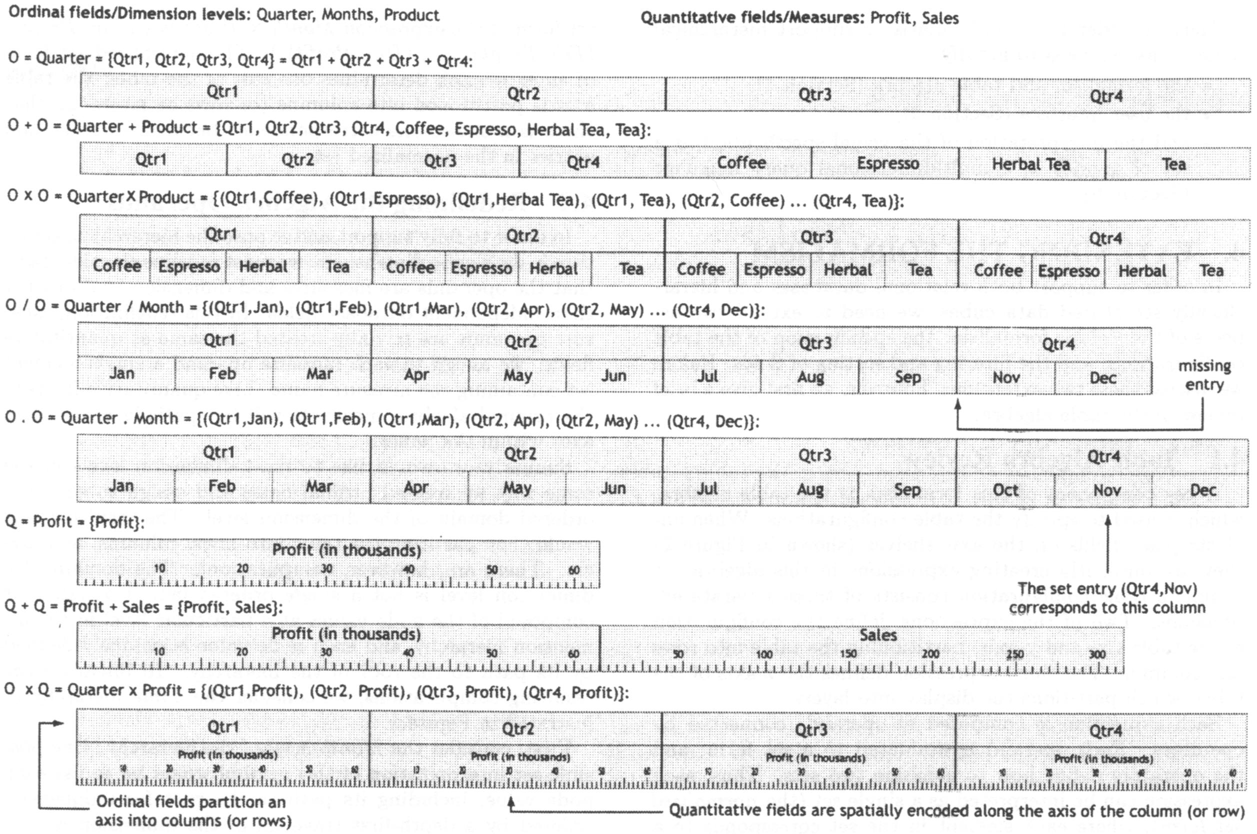

Ordinal fields/Dimension levels: Quarter, Months, Product
Quantitative fields/Measures: Profit, Sales
O = Quarter = {Qtr1, Qtr2, Qtr3, Qtr4} = Qtr1 + Qtr2 + Qtr3 + Qtr4:
Qtr1 Qtr2 Qtr3 Qtr4
O + O = Quarter + Product = {Qtr1, Qtr2, Qtr3, Qtr4, Coffee, Espresso, Herbal Tea, Tea}:
Qtr1 Qtr2 Qtr3 Qtr4 Coffee Espresso Herbal Tea Tea
O x O = Quarter X Product = {(Qtr1,Coffee), (Qtr1,Espresso), (Qtr1,Herbal Tea), (Qtr1, Tea), (Qtr2, Coffee) ... (Qtr4, Tea)}:
Qtr1 Qtr2 Qtr3 Qtr4
Coffee Espresso Herbal Tea
O / O = Quarter / Month = {(Qtr1,Jan), (Qtr1,Feb), (Qtr1,Mar), (Qtr2, Apr), (Qtr2, May) ... (Qtr4, Dec)}:
Jan Feb Mar Apr May Jun Jul Aug Sep Nov Dec
missing entry
O . O = Quarter . Month = [(Qtr1,Jan), (Qtr1,Feb), (Qtr1,Mar), (Qtr2, Apr), (Qtr2, May) ... (Qtr4, Dec)}:
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Q = Profit = {Profit}:
Profit (in thousands)
10 20 30 40 50 60
The set entry (Qtr4,Nov) corresponds to this column
Q + Q = Profit + Sales = {Profit, Sales}:
Profit (in thousands)
Sales
50 100 150 200 250 300
O x Q = Quarter x Profit = {(Qtr1,Profit), (Qtr2, Profit), (Qtr3, Profit), (Qtr4, Profit)}:
Profit (in thousands)
Ordinal fields partition an axis into columns (or rows)
Quantitative fields are spatially encoded along the axis of the column (or row)

Figure 3: Example of the set interpretations and table structures resulting from simple applications of the table algebra operators. Ordinal fields (e.g., dimension levels) partition the table into columns (or rows) and quantitative fields (e.g., measures) are spatially encoded as axes within the columns. Note the difference between the application of the nest and dot operator to the same operands when the fact table does not contain data for October.

3.2 Polaris Overview

Before explaining the extensions to Polaris needed for supporting interactive visual exploration of hierarchically structured data sets, we first give a brief overview of the original Polaris system.

The goal of Polaris was to provide an interface for rapidly and incrementally generating table-based displays (note that from here on out, unless otherwise specified as a fact table or dimension table, the term *table* refers to a table-based visualization and not a relation in a database). Users construct these table-based visualizations via a drag-and-drop interface, dragging field names from the Schema box to various blue shelves throughout the interface, as shown in Figure 2. Any configuration of field names on shelves is valid.

The Polaris interface is simple and expressive because it is built upon a formalism for precisely describing graphical table-based visualizations. The configuration of fields on shelves forms a *visual specification.* Each visual specification is an expression of the Polaris formalism that can be interpreted to determine the exact analysis, query, and drawing operations to be performed by the system.

The specification consists of two main portions. The first portion, built on top of an algebra, describes the structure of the table-based visualization (i.e., how the table is divided into panes). We can think of a table as having three axes: the x-axis divides the table into columns, the y-axis divides the table into rows, and the z-axis layers x-y tables that are composited on top of one another. Each intersection of an x-, y-, and z-axis results in a table pane. Thus, the first portion of the specification consists of *table algebra expressions*, with one expression per axis. Each pane contains a set of records (obtained by querying the data cube) that are visually encoded as a set of marks to create a graphic.

While the first portion of the specification determines the "outer table layout," the remaining portion determines the layout within a pane, such as how the data within a pane is transformed for analysis and how it is encoded visually. Specifically, it describes:

1. The sorting and filtering of fields.
2. The mapping of data sources to layers.
3. The grouping of data within a pane and the computation of statistical properties, aggregates, and other derived fields.
4. The type of graphic displayed in each pane of the table. Each graphic consists of a set of marks (e.g., circles, bars, glyphs, etc.) with one mark per record in that pane.
5. The mapping of data fields to retinal properties of the marks in the graphics (for example, mapping *Profit* to the size of a mark or *Quarter* to the color).

We only need to extend the table algebra and the specification of the filtering and sorting. The rest of the formalism, including how we determine the type of graphic and the visual encodings, has not changed and so we do not discuss them further here. See Stolte et al. [21] for a detailed discussion.

Thus, in order to extend Polaris to support hierarchical dimensions, we need to modify:

- the formalism and table algebra (Section 4),
- the user interface (Section 5),
- and the interpretation of the visual specification as a set of queries in a multidimensional query language (Section 6).

4. EXTENDING THE FORMALISM

In order to support both relational databases and hierarchically structured data cubes, we need to extend two aspects of the Polaris formalism: the specification of the table configurations, and the filtering and sorting of fields. Before we discuss these two extensions, however, we first give a brief review of the table algebra.

4.1 Table Algebra Review

A key component of this formalism is the table algebra, which is used to specify the table configurations. When analysts place fields on the axis shelves (shown in Figure 2) they are implicitly creating expressions in this algebra. A complete table configuration consists of three separate expressions. Two of the expressions define the configuration of the table's x- and y-axis, partitioning the table into rows and columns. The third expression defines the z-axis of the table, which partitions the display into layers.

Each expression is composed of operands connected by operators. Each operand is evaluated to a set form, and the operators define how to combine two sets. Thus, each expression can be interpreted as a single set (the *normalized set form*), where each element in the set corresponds to a single row, column, or layer.

To be more specific, each operand of the table algebra is the name of a field. There are two types of operands: ordinal and quantitative. Whether an operand is ordinal or quantitative depends on the type of the corresponding field in the database.

The set interpretation of an ordinal operand consists of the members of the ordered domain of the field. For example, the set interpretation of the *Month* operand would be {*Jan*, *Feb*, ..., *Dec*}. The set interpretation of a quantitative operand is a single-element set containing the field name. For example, the set interpretation of the Profit operand would be {*Profit*}.

The assignment of sets to the different types of operands reflects the difference in how the two types of fields are encoded into the structure of the table. Ordinal fields partition the table into rows and columns, whereas quantitative fields are spatially encoded as axes within the table panes. Examples of the set interpretations and resulting table structures for both ordinal and quantitative operands are shown in Figure 3.

As stated above, a valid expression in the algebra is an ordered sequence of one or more operands with operators between each pair of adjacent operands. The operators in this algebra, in order of precedence, are cross (×), nest (/), and concatenation (+); parentheses can be used to alter the precedence. Because each operand is interpreted as an ordered set, the precise semantics of each operator are defined in terms of how they combine two sets (one each from the left and right operands) into a single set. Some examples are shown in Figure 3.

Thus, every expression in the algebra can be reduced to a single set, with each entry in the set being an ordered concatenation of zero or more ordinal values followed by zero or more quantitative field names. For example, the normalized set form of the expression *Month* × *Profit* is { (*Jan*, *Profit*), (*Feb*, *Profit*), ..., (*Dec*, *Profit*) }. The normalized set form of an expression determines one axis of the table: the table axis is partitioned into columns (or rows or layers) so that there is a one-to-one correspondence between columns and entries in the normalized set.

4.2 Redefining the Algebra Operands

In order to fully support and expose the hierarchical structure in data cube dimensions, we must redefine the algebra so that the operands are measures and dimension levels rather than independent database fields. In this redefinition, measure operands are trivially treated the same as quantitative fields: we assign to each measure operand a single element set containing the measure name. Like quantitative fields in the original algebra, measures will be spatially encoded as axes within the panes.

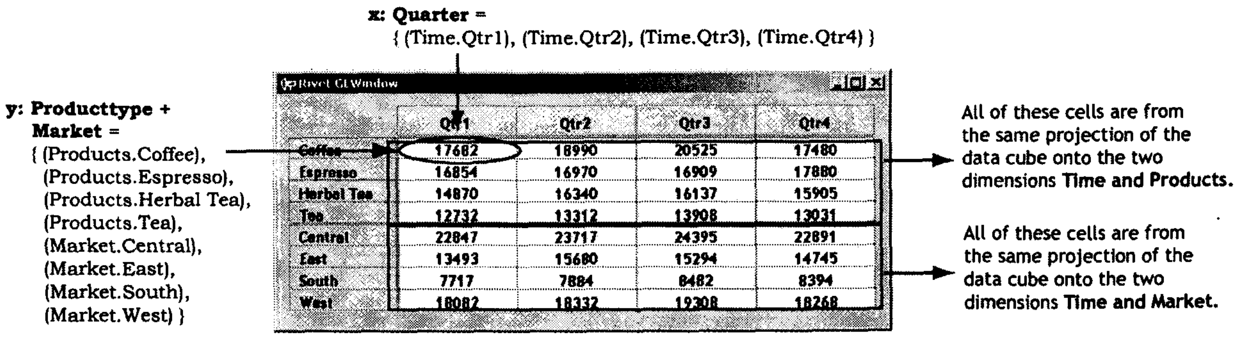
Similarly, we would like to treat dimension levels in the same way we treated ordinal fields and assign to each the ordered domain of the dimension level. The resulting sets would then partition the table into rows, columns, and layers. There are, however, complications. The domain of a dimension level is not a single ordered list. Instead, it is composed of the node values at a particular level in the dimension hierarchy, and each node value is uniquely defined by its path to the root of the hierarchy. To illustrate the resulting complications, we consider the *Time* hierarchy illustrated in Figure 1.

First, consider the *Month* level of the hierarchy. One possible set interpretation of this symbol would be to list each node value, including its path to the root for uniqueness, ordered by a depth-first traversal of the dimension hierarchy; e.g., {*1998.Qtr1.Jan*, ..., *1999.Qtr4.Dec*}. Although this approach provides a unique set interpretation for each dimension level, it limits the expressiveness of the algebra. Any table constructed to include *Month* must also include *Year*; it is not possible to create displays that summarize monthly values across years, a useful view that we would like to support. Interestingly, however, summarizing monthly values across years is not a standard projection of a data cube, as it requires aggregating across a hierarchical level. We discuss how we compute this type of aggregation in Section 6.

A second approach would be to list only the node values, ignoring the path to the root of the hierarchy and excluding repeated values. Again, we order the node values by a depth-first traversal of the dimension hierarchy. For *Month*, this approach would yield {*Jan*, *Feb*, ..., *Nov*, *Dec*}. Clearly, using this set interpretation we can generate displays that summarize monthly values across years. Furthermore, we can generate displays that drill down into a hierarchy by using our nest (/) operator, e.g., *Year / Month*.

The use of the nest for drilling down into a hierarchy, however, would be flawed. The nest operator is unaware of the defined hierarchical relationship between the dimension levels but instead works by deriving a relationship based on the tuples in the fact table. Not only is this approach inefficient, as fact tables are often quite large, but it can also yield incorrect results. For example, consider the situation where no data was logged for October. Application of the nest operator would result in an incorrectly derived *Time* hierarchy that did not include October as a child of Qtr4 or either year (see Figure 3).

Our solution is to introduce another operator, the dot (.) operator, that is similar to the nest operator but "hierarchy-aware." We review the definition of nest and then define dot. If we define *FT* to be the fact table being analyzed, *r* to be a



x: Quarter =
{ (Time.Qtr1), (Time.Qtr2), (Time.Qtr3), (Time.Qtr4) }

y: Producttype +
Market =
{ (Products.Coffee),
(Products.Espresso),
(Products.Herbal Tea),
(Products.Tea),
(Market.Central),
(Market.East),
(Market.South),
(Market.West) }

	Qtr1	Qtr2	Qtr3	Qtr4
Coffee	17682	18990	20525	17480
Espresso	16854	16970	16909	17880
Herbal Tea	14870	16340	16137	15905
Tea	12732	13312	13908	13031
Central	22847	23717	24395	22891
East	13493	15680	15294	14745
South	7717	7884	8482	8394
West	18082	18332	19308	18268

All of these cells are from the same projection of the data cube onto the two dimensions **Time and Products.**

All of these cells are from the same projection of the data cube onto the two dimensions **Time and Market.**

Figure 4: Each pane in a Polaris visualization corresponds to a slice of a projection of a data cube. The projection in each pane is determined by the contents of the "Level of Detail" shelf and by the normalized set form of the table expressions. The table is partitioned into rows, columns, and layers corresponding to the entries in these sets. The underlying data cube must be projected to include only the dimensions that occur in these entries. This figure shows a simple text-based table that requires two separate projections of the data cube because of the concatenation in the y-axis expression.

record, and $A(r)$ to be the value of the field A for the record r, then the definition of nest, as presented in [21], is:

$$A/B = \{(a,b) \mid \exists r \in FT \text{ st } A(r) = a \;\&\; B(r) = b\}$$

The dot operator is defined similarly. If we define DT to be the relational dimension table defining the hierarchy that contains the levels A and B, and A precedes B in the schema of DT, then:

$$A.B = \{(a.b) \mid \exists r \in DT \text{ st } A(r) = a \;\&\; B(r) = b\}$$

Note that whereas nest produces a set of two-valued tuples, dot produces a set of single-valued tuples, each containing a qualified value. If the two operands are not levels of the same dimension hierarchy (or set interpretations of operations on levels of the same hierarchy), or A does not precede B in the schema of DT (e.g., A must be an ancestor level in the tree defined by DT), then the dot operator evaluates to the empty set. With this definition, the two expressions *Month* and *Year.Month* are not equivalent: *Month* is interpreted as {*Jan*, *Feb*, ..., *Dec*} whereas *Year.Month* would be interpreted as {*1998.Jan*, *1998.Feb*, ..., *1999.Dec*}. With a fully populated fact table, *Year.Month* is equivalent to *Year / Month*.

Given these set interpretations for dimension and measure operands, we can apply the set semantics for each operator to reduce expressions in this new algebra to their normalized set form, with each entry in the normalized set being an ordered concatenation of zero or more domain values followed by zero or more measure names. As before, the normalized set form determines one axis of the table.

4.3 Filtering and Sorting within the Algebra

In our original formalism, a table configuration was specified by three expressions in the table algebra, and then filtering and sorting was specified separately by listing the sorted and filtered domain for each database field that was to be filtered or sorted. When the set interpretation was generated for field operands in the algebra, these specified domains would be used. It is possible, however, to generate a more succinct and general formalism if we incorporate the filtering and sorting directly into the table algebra.

In our revised formalism, if a dimension or measure is to be filtered (or sorted), then the filtered and sorted domain is listed directly after the instance of the level or measure operand in the expression, in effect directly specifying a set interpretation for the operand. For example, if we wished to filter the expression *Month* + *ProductType* to include only the first three months of the year, sorted in reverse order, we would specify the filtered domain by including it in the expression as follows: *Month*{*Mar*, *Feb*, *Jan*} + *ProductType*. The advantage provided by this revision of the table algebra is the ability to specify separate filters and orderings for different instances of the same operand in an expression. Similarly, we can filter a measure by specifying a range of values, e.g., *Profit*{0, 500}.

We also need to allow the use of qualified values in the specification of filtering or sorting of dimension levels. As we discussed in Section 3.1, a value in a dimension hierarchy can either be described by simply stating the value in the node (an unqualified value) or by describing a path from that node to the root node in the hierarchy (a qualified value). When filtering or sorting a dimension level, it is necessary to be able to use both types of values in the specification, as the unqualified node values are often not unique. For example, if the user wishes to exclude *1998.Jan* but not *1999.Jan*, then qualified values must be used.

5. REDEFINING THE USER INTERFACE

Having redefined the formalism underlying the Polaris interface, we must now alter the interface to support hierarchically structured data. Five major changes need to be made:

1. the Schema list must display dimension hierarchies and measures, not simply database fields;
2. the analyst must be able to distinguish between *Month* and *Year.Month* when including *Month* in a specification;
3. the analyst must be able to filter a dimension level using qualified values;
4. the analyst must be able to quickly drill down and roll up a dimension hierarchy using the interface;
5. the analyst needs to be able to change the number of marks within each pane to reflect different levels of detail.

Figure 2 illustrates the revised interface. We now discuss each interface extension in detail.

5.1 The Schema

In the original interface, the analyst was presented with a list box containing the ordinal and quantitative fields in the database. The analysts included these fields in a specification simply by dragging and dropping the field's name onto

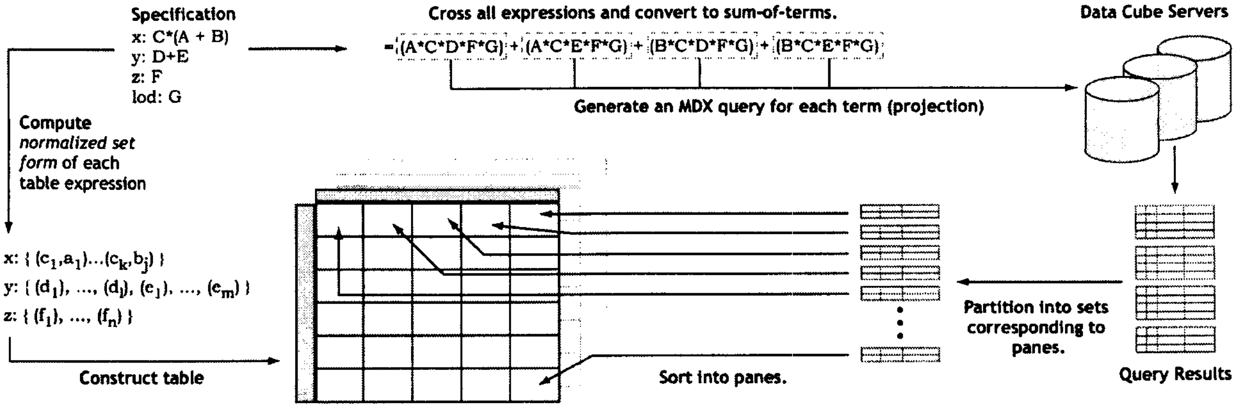

Specification
x: C*(A + B)
y: D+E
z: F
lod: G
Cross all expressions and convert to sum-of-terms.
=(A*C*D*F*G) + (A*C*E*F*G) + (B*C*D*F*G) + (B*C*E*F*G)
Data Cube Servers
Generate an MDX query for each term (projection)
Compute *normalized set form* of each table expression
x: { (c1,a1)...(ck,bj) }
y: { (d1), ..., (dj), (e1), ..., (em) }
z: { (f1), ..., (fn) }
Construct table
Sort into panes.
Partition into sets corresponding to panes.
Query Results

Figure 5: The overall data flow in Polaris when generating a visual representation of a data cube.

the appropriate shelf in the interface. To support hierarchical data cubes, we have extended this list box to display the dimensions of the data cube with an ordered list of the dimension's levels beneath each dimension. The analysts can drag and drop any dimension level to the interface as they did with the ordinal fields of the database. The dimension's name, however, cannot be dragged to the interface; the analyst can only manipulate the individual levels within a dimension.

5.2 Qualifying Dimension Levels

When an analyst drops a dimension level, such as *Month*, on a shelf, there are several potential intentions. He may intend to include the operand *Month* in an expression, but he may also mean *Year.Month* or *Year.Quarter.Month*; the analyst needs to be able to specify the exact qualification desired. Our solution is the make full qualification (e.g., *Year.Quarter.Month*) the default. To generate a different qualification, the user can right-click the dimension level in the shelf and select the "Qualification..." menu item. He is then presented with a dialog box that allows him to explicitly specify which of the intermediate levels to include in the qualification of the operand, thus generating the applicable expression.

5.3 Qualifying Dimension Level Filters

When applying filters to a dimension level, an analyst may want to specify the filter using either qualified or unqualified values. We have extended the Polaris interface to allow both options. For example, if the user wishes to exclude *1998.Jan* but not *1999.Jan*, he can choose to filter using qualified values. Similarly, it is possible to specify a filtering using unqualified values: each qualified value that matches the unqualified value will be included in the filter. Currently, Polaris requires the filter be specified using either qualified or unqualified values, but not both. As a future extension, we intend to support heterogeneous filtering.

5.4 Drilling down and Rolling up

When analyzing and exploring large data cubes, a common operation is to drill down or roll up within a dimension hierarchy. Therefore, it is important to include a simple mechanism for performing these operations. One option is for the analyst to remove the current level from the appropriate shelf (by dragging it off the shelf) and then drag the new level to that same shelf. Although the desired effect is achieved, it is more complicated than we would like.

We provide an alternate mechanism for drilling down and rolling up a dimension. Within the box representing each dimension level on a shelf, there is an "∇" icon, as can be seen in Figure 2. When the user clicks on the "∇" icon, he is presented with a listing of all the levels of the dimension (including diverging levels in complex dimension hierarchies). Selecting a new level is interpreted as a drill down (or roll up) operation along that dimension and the current level is automatically replaced with the selected level (with the same qualification). Thus, the user can rapidly move between different levels of detail along a dimension, refining the visualization as he navigates.

5.5 Grouping within Panes

In the original version of Polaris, the analyst specified the grouping of tuples within each pane by placing fields on the shelf titled "Group By." Each field in this shelf was included in the GROUP BY clause in the SQL query that aggregated the data in each pane into tuples to be mapped to marks.

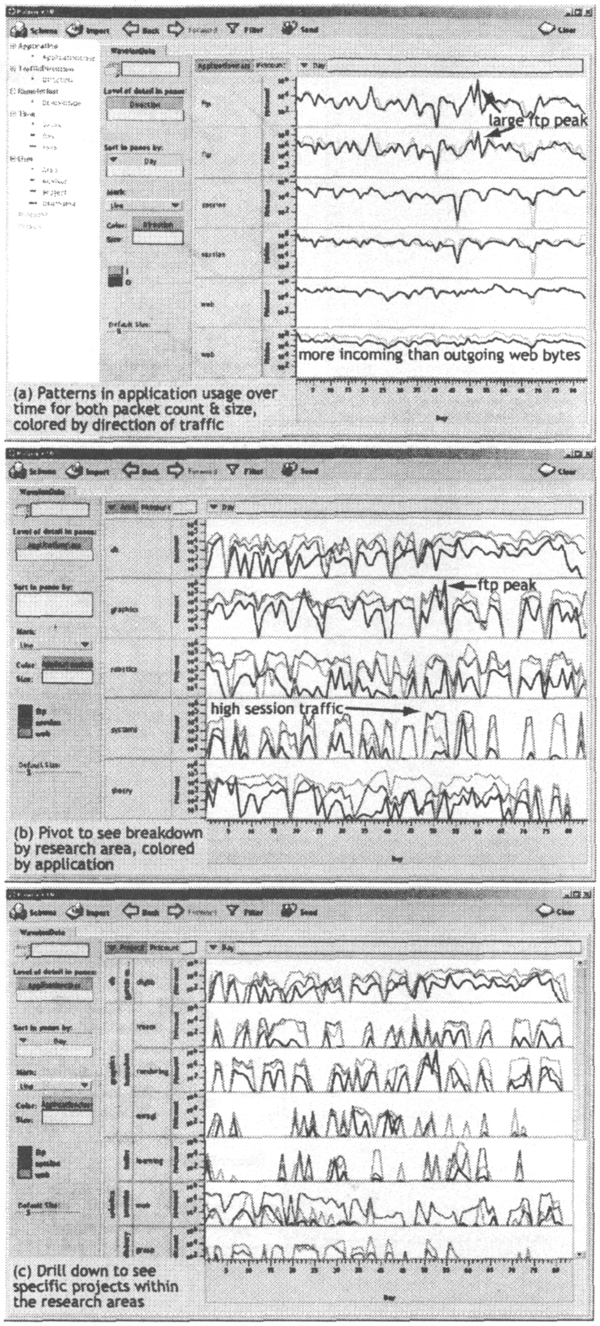
The situation when visualizing data cubes is slightly different. The query for each pane does not produce a relational data set that is then grouped and aggregated. Instead, each pane corresponds to a projection of the data cube, with the projection determined by the dimension levels included in the table expressions. To produce additional marks within a pane, the analyst must specify additional dimensions to be included in these projections, done by including the desired dimension levels in the "Level of Detail" shelf (shown in Figure 2), the hierarchical analog of the "Group by" shelf.

As was the case with the original version of Polaris, this "Level of Detail" shelf gives the analyst the ability to rapidly drill down into their data without changing the table configuration. Changing the level of detail without changing the table configuration only changes the data density within each pane.

6. GENERATING QUERIES

The final step in extending Polaris to fully support hierarchical data cubes is to show how to construct an efficient set of multidimensional queries from a specification in our formalism.

Each pane in a Polaris visualization corresponds to either a slice of a projection of the data cube or an aggregation of such a projection. The specific projection corresponding to each pane is determined by the contents of the "Level of Detail" shelf (discussed in Section 5.5) and by the normalized set form of the table axis expressions (discussed in Section 4). The table is partitioned into rows, columns, and layers corresponding to the entries in these sets. Therefore,



(a) Patterns in application usage over time for both packet count & size, colored by direction of traffic

(b) Pivot to see breakdown by research area, colored by application

(c) Drill down to see specific projects within the research areas

Figure 6: Analysis of network usage data using Polaris.

each pane in the table is associated with three set entries corresponding to its row, column, and layer, respectively.

The underlying data cube must be projected to include only the dimension levels that occur in the three set entries (and in the Level of Detail shelf) and it must be sliced to include only the specific dimension members that occur in these entries. This process is illustrated in Figure 4. When multiple set entries defining a pane refer to different levels of the same dimension, the correct projection to retrieve is the one corresponding to the most detailed level of that dimension. Before determining how the projections are efficiently retrieved from the server, we must carefully consider the situation where set entries contain values whose qualification skips levels in the hierarchy.

Set entries containing values whose qualification skips levels (e.g., *Time.Jan*) are interpreted to imply that nodes in the hierarchy whose values are not unique (when we consider only the included levels) should be aggregated in that projection. In the *Time.Jan* example, the aggregation for the pane must be computed by aggregating across years, and thus, across a hierarchy level rather than up the hierarchy. This type of aggregation is not natively supported in most hierarchical query languages. Thus, we request the cube projection from the remote server and compute the aggregation within Polaris before sorting tuples into panes, as shown in Figure 5. If all node values are unique across the entire level, then no aggregation needs to be performed.

Although it is possible for each pane to correspond to a different projection of the cube, the common situation is for a large number of panes to correspond to the same projection and differ only by how that projection is sliced. For efficiency, we would like to consider these panes as a group and send a single query to the OLAP server requesting the appropriate projection (and then, if necessary, perform a single aggregation of the projection). The projection can then be sorted into panes locally.

One key to efficiently utilizing the OLAP server is this grouping of queries. By algebraically manipulating our table expressions, we can quickly determine all projections corresponding to a given table configuration. The key observation is that of our four algebraic operators (nest, cross, concatenate, and dot), the only operator that can produce adjacent panes with differing projections is the concatenate operator. Nest, cross, and dot include all input dimension levels in each output set entry; concatenate does not. Thus, if we compute a single expression as the cross of the three table expressions and then reduce to a sum-of-terms form, the resulting terms will correspond to the set of projections that need to be generated. This process is illustrated in Figure 5.

Most typical multidimensional query languages provide a mechanism for generating projections of the data cube. Our current implementation generates a single MDX query to a remote Microsoft Analysis Server for each projection. The resulting cells are then sorted into panes using transformation capabilities built into Polaris. In addition, any explicitly specified filtering of dimension members is included in the MDX queries sent to the remote server. The overall data flow in Polaris is depicted in Figure 5.

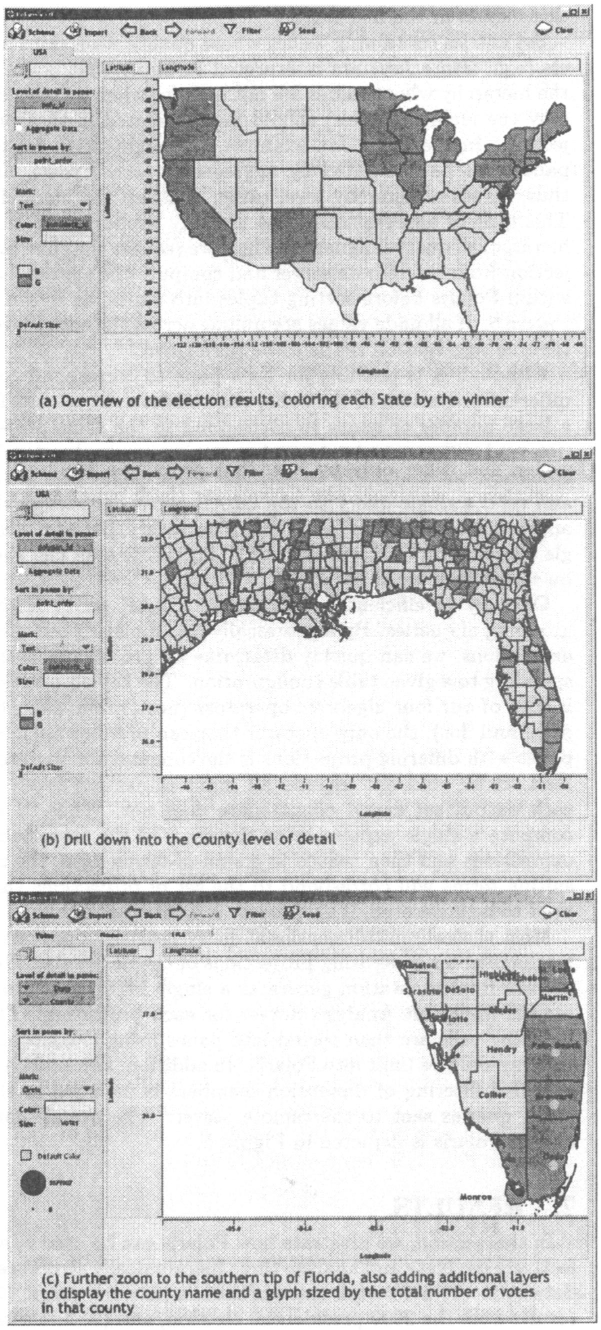
7. RESULTS

In this section, we illustrate how Polaris can be used to effectively navigate and analyze three hierarchically structured data sets: (1) a 12-week trace of mobile network usage, (2) results from the 2000 presidential election, and (3) historical business metrics for a hypothetical coffee chain.

7.1 Mobile Network Usage Data

Figure 6 shows an analysis of a 12-week trace of every packet that entered or exited the mobile network in the Gates building at Stanford University [22]. Over the 12 weeks, 78 million packet headers were collected. The analysis goal is to understand usage patterns of the mobile network. This data is stored in a data cube with many different dimensions (User, Time, Remote host, Traffic direction, and Application), each with multiple levels of detail. In this analysis, the queries generated when the user dropped a field on a shelf took one to two seconds to execute and returned several hundred to tens of thousands of tuples.

To start the analysis, the analyst first sees if she can spot

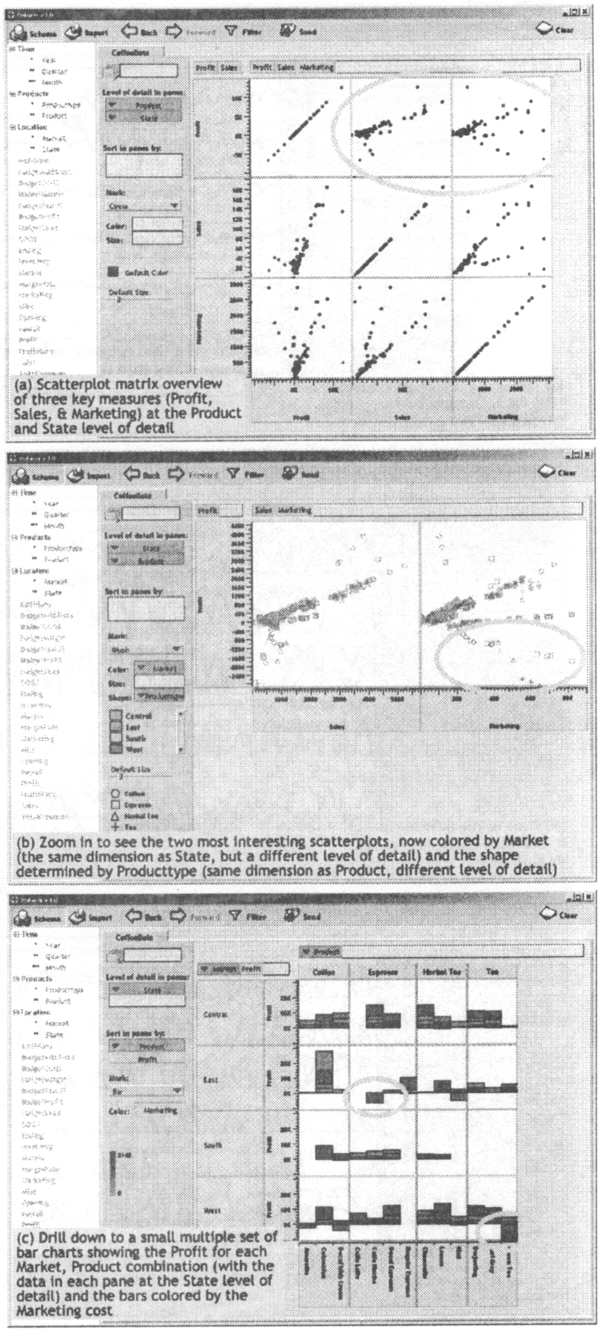


(a) Overview of the election results, coloring each State by the winner

(b) Drill down into the County level of detail

(c) Further zoom to the southern tip of Florida, also adding additional layers to display the county name and a glyph sized by the total number of votes in that county

Figure 7: Analysis of the results of the 2000 presidential election.



(a) Scatterplot matrix overview of three key measures (Profit, Sales, & Marketing) at the Product and State level of detail

(b) Zoom in to see the two most interesting scatterplots, now colored by Market (the same dimension as State, but a different level of detail) and the shape determined by Producttype (same dimension as Product, different level of detail)

(c) Drill down to a small multiple set of bar charts showing the Profit for each Market, Product combination (with the data in each pane at the State level of detail) and the bars colored by the Marketing cost

Figure 8: Analysis of sales data for a hypothetical coffee chain.

any patterns in time, so she creates a series of line charts in Figure 6(a) showing packet count and size versus time for the most common applications, broken down and colored by the direction of the traffic. In these charts, the analyst can see that the web is the most consistently used application, while session is almost as consistent. File transfer is the least consistent, but also has some of the highest peaks in both incoming and outgoing ftp traffic. Note the log scale on the y-axes.

Given this broad understanding of traffic patterns, the next question posed by the analyst is how the application mix varies depending on the research area. The analyst pivots the display to generate a single line chart of packet count per research area over time, broken down and colored by application class (Figure 6(b)). From this breakdown, the analyst can see that the graphics group was responsible for the large incoming and outgoing file transfers. She can also see that the systems group had atypically high session traffic.

Curious, the analyst then drills down further to see the individual project groups (Figure 6(c)), discovering that the large file transfers were due to the rendering group within the graphics lab, while the robotics lab had vastly different behavior depending on the particular group (the mob group dominated by session traffic, while the learning group had more web traffic, for example).

7.2 2000 Presidential Election Results

Figure 7 shows Polaris being used to explore and analyze the results of the 2000 presidential election. This data is particularly interesting because the visualizations used to explore it are created from two separate data sets. The first data set is a relational database of approximately 500,000 tuples (stored in Microsoft's SQLServer) describing detailed polygonal outlines of the states and counties in the USA. Additional levels of detail have been constructed by polygon simplification, and the resulting levels of detail form a Location hierarchy. The second data set is stored as a data cube (in Microsoft's Analysis Server) and contains detailed county-by-county vote results (also with a Location dimension). In the first two visualizations, these data sets are explicitly joined before being imported into Polaris. In the final visualization, we use the ability in Polaris to visually join data sets using layers. In this analysis, the execution time for the queries varied from less than one second for the overview visualizations to two seconds for the detailed visualizations, where the retrieved relation included tens of thousands of tuples.

In Figure 7(a), the analyst has generated an overview of the entire country at the State level in the Location hierarchy, coloring each state by which candidate won that state. The analyst is interested in more detailed results for the state of Florida, so she filters on the Latitude and Longitude measures to focus on Florida and changes the level of detail to County, generating Figure 7(b). In the final visualization, shown in Figure 7(c), the analyst further focuses on the southern tip of Florida (by again filtering the Latitude and Longitude measures). Furthermore, she adds two additional layers to the visualization (read directly from the data cube) and displays both the name and the total number of votes counted in each county.

7.3 Historical Profit/Sales for a Coffee Chain

The final analysis is shown in Figure 8. The data being analyzed is two years of business metrics for a hypothetical nationwide coffee chain, comprising approximately 5,000 tuples stored in a data cube. The data is characterized by three main dimensions (Time, Products, and Location), each with multiple levels of detail. We consider a scenario where the analyst is concerned with reducing marketing expenses and is trying to identify products that are not generating profit and sales proportional to their marketing costs. The typical query time for the visualizations created in this scenario was between 0.1 and 0.2 seconds.

The first visualization created, Figure 8(a), is an overview of three key measures (Profit, Sales, and Marketing) as a scatterplot matrix. The analyst has drilled down using the Level of Detail shelf to the Product and State level. The two charts circled in orange show that several of the distributions do not reflect the positive correlations that the analyst was expecting. To further investigate, the analyst reduces the scatterplot matrix to two graphs and colors the records by Market and Producttype (Figure 8(b)), thus identifying espresso products in the East region and tea products in the West region as having the worst marketing cost to profit ratios.

In the final visualization, Figure 8(c), the analyst drills down into the data to get a more detailed understanding of the correlations: She creates a small multiple set of stacked bar charts, one for each Market and Producttype. Within each chart, the data is further drilled down by individual Product and State. Finally, each bar is colored by the Marketing cost. As can be seen in the visualization, several products such as Caffe Mocha in the East have negative profit (a descending bar) with high marketing cost (a bright red bar). Having identified such poorly performing products, the analyst can modify the marketing costs allocated to them.

7.4 Summary

Each of these case studies demonstrates how analysis progresses from a high level of abstraction to detailed views of the data. Furthermore, each example shows the importance of being able to easily change the data being viewed, pivot dimensions, and drill down during the analysis process.

8. DISCUSSION

In this section, we focus on two points of discussion. First, we discuss the different roles Polaris can play in the knowledge discovery process, and second, we discuss how our formalism can be applied to the development of generalized visualization systems, particularly level of detail systems.

In this paper, we have demonstrated the effectiveness of Polaris as a stand-alone tool for visual mining of large, hierarchical databases. Equally important is how Polaris can be coupled with automated data mining systems to help analysts better understand not only their data, but also the models generated by the algorithms. First, Polaris can be used as a precursor to data mining: The analyst benefits from an understanding of the overall structure of the data that helps her steer the discovery process and provides context for "hidden information" discovered by the algorithms. Second, Polaris can also be used to validate and comprehend the models and results generated by algorithmic analysis. Analysts do not want to treat an algorithm as a black box and blindly trust its output. One technique for using Polaris for validation is to construct hierarchical dimensions from the output generated by classification algorithms. The analyst can then drill down and roll up the data, traversing the classification hierarchy and inspecting the records sorted into each bucket, further developing understanding and trust.

A second point of discussion is the application of our formalism to the development of general visualization systems. Although we have only demonstrated our formal language as an underlying technology for the Polaris interface, we believe it is a promising basis for the development of a wide-range of visualization systems. One example is in the development of interactive "semantic-zooming" visualization systems. Programmers developing such systems need a mechanism for describing a wide range of visual displays, with each display being associated with a different level of detail view of the data. Using our formalism, these programmers could simply describe each visual display with a succinct specification. When the user interacts with the interface to move to a different level of detail, the system need only feed the appropriate specification into our interpreter. The interpreter would generate all of the drawing operations and queries necessary to generate the display. In addition to simplifying the development of such systems, the presence of an underlying formalism also serves to help clearly define the semantics of the interface, as demonstrated by Polaris.

9. CONCLUSIONS AND FUTURE WORK

We have extended Polaris, an interface for the exploration and analysis of large multidimensional databases, to fully support and expose the hierarchical structure of data cubes. These dimension hierarchies play an indispensable role in the analysis of large databases where, in order for the analysis task to be manageable, it is necessary to perform the analysis at multiple levels of aggregation, moving from visual overviews to details on demand. In extending Polaris, we have extended not only the interface, but also the underlying algebraic formalism. Furthermore, we have developed an efficient mechanism for interpreting the formal specifications as a collection of multidimensional queries.

We have many plans for future work in extending this system. As databases continue to grow in size, developing tools and techniques for the interactive exploration of data at multiple levels of detail is crucial. As we discussed in Section 8, we believe our algebraic formalism provides a solid foundation upon which to build visualization systems. We are currently building systems that, a la Pad++ [18], automatically and interactively change the visual representation as the analyst changes level of detail. This research has many interesting challenges, including transitions between different visual representations, mapping representations to levels of detail, and maintaining interactivity while exploring large data warehouses.

A second area of future research is the visual presentation of metadata. Hierarchically structured dimensions are one instance of an increasingly popular trend: the augmentation of data with rich domain-specific metadata. This metadata is as important to the analysis process as the underlying database itself. An important area of future research is the development of visualization techniques that display this metadata and leverage it in the display of the described data.

10. ACKNOWLEDGMENTS

The authors especially thank Robert Bosch and Chris Olston for their reviews of manuscript drafts and many useful discussions. The coffee chain data is courtesy of Stephen Eick and Visual Insights. This work was supported by the US Department of Energy through the ASCI Level 1 Alliance with Stanford University.

11. REFERENCES

[1] B. Becker. Visualizing Decision Table Classifiers. In Proc. of Information Visualization, October, 1998, pp. 102-105.

[2] B. Becker, R. Kohavi, and D. Sommerfield. Visualizing the Simple Bayesian Classifier. In KDD Workshop on Issues in the Integration of Data Mining and Data Visualization, 1997.

[3] C. Brunk, J. Kelly, and R. Kohavi. MineSet: an integrated system for data mining. In Proc. of the 3rd International Conference on Knowledge Discovery and Data Mining, 1997, pp. 135-138.

[4] A. Buja, D. Cook, and D. F. Swayne. Interactive High-Dimensional Data Visualization. In Journal of Computational and Graphical Statistics, 5(1), 1996, pp. 78-99.

[5] M. Derthick, J. Kolojejchick and S. F. Roth. An Interactive Visualization Environment for Data Exploration. In Proc. of Knowledge Discovery in Databases, August, 1997, pp. 2-9.

[6] S. Eick. Visualizing Multi-Dimensional Data. In Computer Graphics, February 2000, pp. 61-67.

[7] Y. Fua, M. O. Ward, and E. Rundensteiner. Navigating Hierarchies with Structure-Based Brushes. In Proc. of Information Visualization, October 1999, pp. 58-64.

[8] J. Goldstein, S. F. Roth, J. Kolojejchick, and J. Mattis. A Framework for Knowledge-based Interactive Data Exploration. In Journal of Visual Languages and Computing, December 1994, pp. 339-363.

[9] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, H. Pirahesh, and F. Pellow. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. In Proc. of the 12th International Conference on Data Engineering, February 1996, pp. 152-159.

[10] C. G. Healey. On the Use of Perceptual Cues and Data Mining For Effective Visualization of Scientific Datasets. In Proc. Graphics Interface, 1998, pp. 177-184.

[11] Human Genome Project. [online] Available: http://www.ornl.gov/hgmis/about.html, cited February 2002.

[12] H. Jagadish, L. Lakshmanan, and D. Srivastava. What can Hierarchies do for Data Warehouses? In Proceedings of the International Conference on Very Large Databases (VLDB), 1999, pp. 530-541.

[13] D. Keim and H.P. Kriegel. VisDB: Database Exploration using Multidimensional Visualization. In IEEE Computer Graphics and Applications, 14(5), 1994, pp. 40-49.

[14] R. Kohavi. Data Mining and Visualization. Invited talk at the National Academy of Engineering US Frontiers of Engineers, Sept 2000. Available in Frontiers of Engineering: Reports on Leading-Edge Engineering From the 2000 NAE Symposium on Frontiers of Engineering, National Academy Press, 2001.

[15] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki and K. Wenger. DEVise: Integrated Querying and Visual Exploration of Large Datasets. In Proc. of ACM SIGMOD, May, 1997.

[16] S.F. Roth, J. Kolojejchick, J. Mattis and J. Goldstein. Interactive Graphic Design Using Automatic Presentation Knowledge. In Proc. of SIGCHI '94, April 1994, pp. 112-117.

[17] S.F. Roth, P. Lucas, J.A. Senn, C.C. Gomberg, M.B. Burks, P.J. Stroffolino, J. Kolojejchick and C. Dunmire. Visage: A User Interface Environment for Exploring Information. In Proc. of Information Visualization, October 1996, pp. 3-12.

[18] K. Perlin and D. Fox. Pad: An Alternative Approach to the Computer Interface. Proc. of the 20th International Conference on Computer Graphics and Interactive Techniques, August 1993, pp. 57-64.

[19] Sloan Digital Sky Survey. [online] Available: http://www.sdss.org/, cited February 2002.

[20] Spotfire Inc. [online] Available: http://www.spotfire.com, cited February 2002.

[21] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A System for Query, Analysis, and Visualization of Multi-dimensional Relational Databases. In IEEE Transactions on Visualization and Computer Graphics, Vol. 8, No. 1, January 2002, pp. 52-65.

[22] D.Tang and M. Baker. Analysis of a Local-Area Wireless Network. In Proc. of the 6th International Conference on Mobile Computing and Networking, August 2000, pp. 1-10.

[23] K. Thearling, B. Becker, D. DeCoste, B. Mawby, M. Pilote, and D. Sommerfield. Visualizing Data Mining Models. In Information Visualization in Data Mining and Knowledge Discovery. Edited by U. Fayyad, G. Grinstein, and A. Wierse. Morgan Kaufman, 2001.

[24] E. Thomsen. OLAP Solutions: Building Multidimensional Information Systems. Wiley Computer Publishing, New York, 1997.

[25] M. O. Ward. XmdvTool: Integrating multiple methods for visualizing multi-variate data. In Proceedings of IEEE Visualization, 1994, pp. 326-336.

[26] J. Welling and M. Derthick. Visualization of Large Multi-dimensional Datasets. In Proc. of Virtual Observatories of the Future 2000.

[27] A. Woodruff, C. Olston, A. Aiken, M. Chu, V. Ercegovac, M. Lin, M. Spalding and M. Stonebraker. DataSplash: A Direct Manipulation Environment for Programming Semantic Zoom Visualizations of Tabular Data. Journal of Visual Languages and Computing, 12(5), October 2001, pp. 551-571.