Data Lineage and Information Density in Database Visualization

by

Allison Gyle Woodruff

B.A. (California State University, Chico) 1987
M.A. Linguistics (University of California, Davis) 1989

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Michael Stonebraker, Chair
Professor Joseph Hellerstein
Professor James Landay
Professor Ray Larson

Fall 1998

The dissertation of Allison Gyle Woodruff is approved:

Chair Date

Date

Date

Date

University of California at Berkeley

Fall 1998

Data Lineage and Information Density in Database Visualization

by
Allison Gyle Woodruff

Abstract

Data Lineage and Information Density in Database Visualization

by

Allison Gyle Woodruff
Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Michael Stonebraker, Chair

Visual representations of data help users interpret and analyze information. We have identified two key issues in existing visualization systems: data lineage and information density. This dissertation defines these problems and details solutions for them. We show that our techniques can be applied in database visualization systems, and we discuss how they improve the usability of these systems.

The data lineage problem occurs when users apply a sequence of processing steps to input data sources; when viewing the final result, these users may wish to trace certain elements in the result back to the original input items. We call these types of queries *data lineage queries*. Current systems, *e.g.*, geographic information systems or scientific visualization systems, provide little support for this task. In the first part of this dissertation, we discuss techniques for allowing users to access intermediate results efficiently while performing data lineage queries. We then introduce weak inversion and verification and show how they can be used to reconstruct the (approximate) lineage of derived data. Because they eliminate much of the irrelevant source data, weak inversion and verification can greatly reduce the amount of source data the end user must examine while performing a data lineage query.

Visualizations often display too much information, making it difficult for users to interpret them. Similarly, visualizations often display too little information, thereby underutilizing display space. In the second part of this dissertation, we describe the general principle of constant information density. We show how both semantic and spatial transformations based on constant information density can be applied to create visualizations

with appropriate density, thereby minimizing clutter and sparseness in the display. We describe an end-user programming environment in which users can construct visualizations with constant information density.

Professor Michael Stonebraker
Dissertation Committee Chair

To all the teachers and colleagues who have taught me so much.

Contents

List of Figures

List of Tables

Acknowledgements

I am grateful to my advisor, Professor Michael Stonebraker, for his insightful comments and support. I wish to thank the other members of my committee, Professors Joseph Hellerstein, James Landay, and Ray Larson, for much helpful feedback and many interesting discussions. I would also like to thank the members of the Tioga and DataSplash development teams, particularly Jolly Chen, Michael Chu, Vuk Ercegovac, Mark Lin, Chris Olston, and Mybrid Spalding.

The people who have provided encouragement are too numerous to mention. I will simply thank my friends and family for their invaluable support and enthusiasm.

This work was partially supported by NSF under grants #IRI-9400773 and #IRI-9411334.

Chapter 1

Introduction and related work

In this chapter, we motivate our work. We introduce the general concepts that we explore during the remainder of the dissertation. Specifically, in Section 1.1, we define data lineage and introduce our techniques for supporting data lineage queries. In Section 1.2, we introduce the Principle of Constant Information Density and show how we have applied it to the construction of visualizations with appropriate density.

1.1 Data lineage

In this section, we introduce dataflow systems, which provide the context for our work. We then define data lineage and discuss its significance in dataflow systems. Finally, we introduce our techniques for supporting data lineage in dataflow systems.

1.1.1 Dataflow systems

Dataflow systems allow users to specify the flow of data between different processing steps. These systems then manage the flow of data accordingly when the steps are executed. Tioga [41] is one such system, a graphical application development tool that uses the boxes and arrows notation popularized by scientific visualization systems such as AVS [49], Data Explorer [25], and Khoros [34]. Tioga improves upon these systems by providing sophisticated data management using the POSTGRES database management system (DBMS) [43].

In the Tioga programming model, boxes represent user-defined database queries or browsers that display data, and edges between boxes represent flow of data. Although only

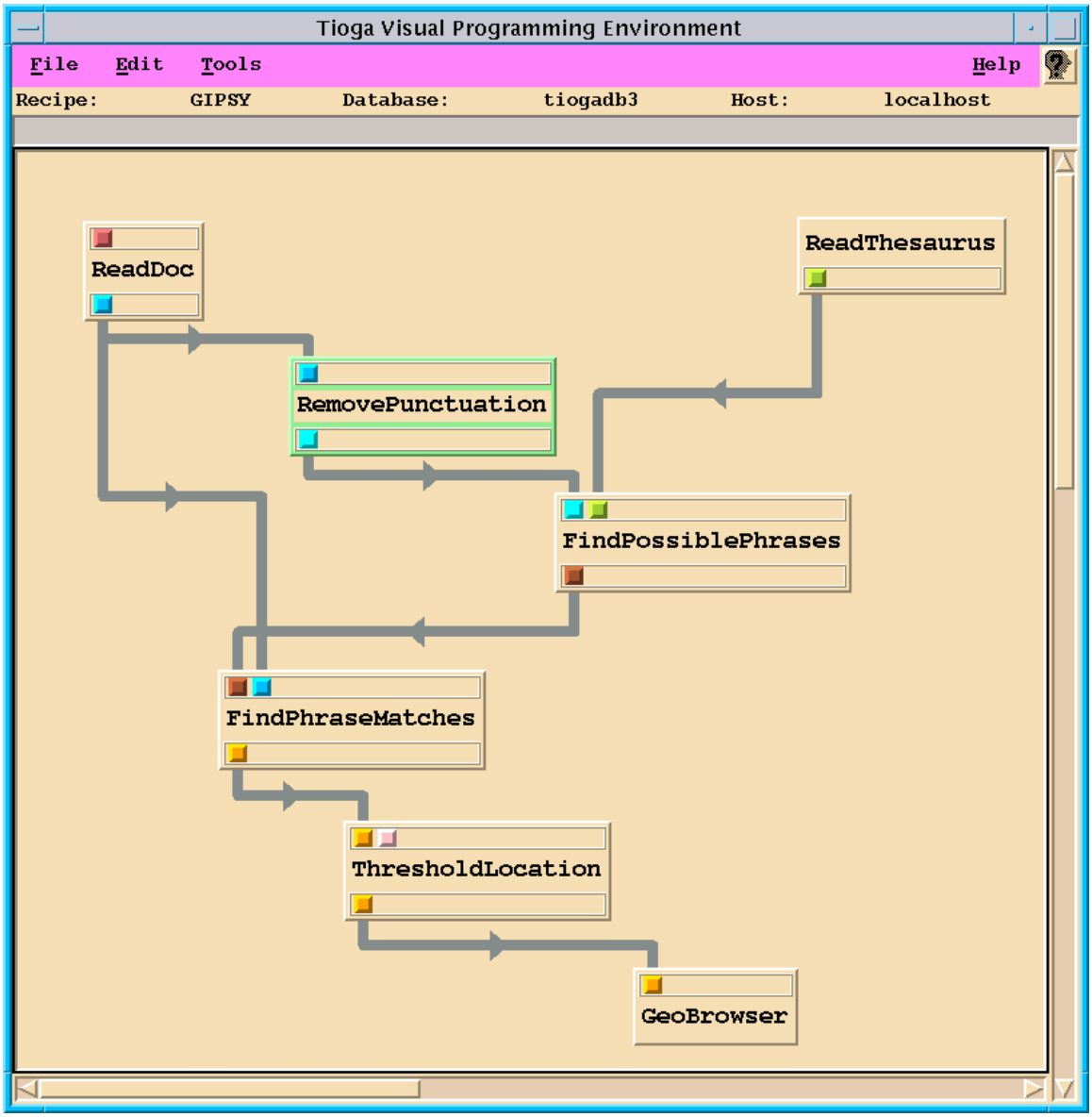


Figure 1.1: A sample Tioga recipe.

a few dozen boxes have currently been implemented, additional boxes may be programmed by users. Nonexperts build visual programs called *recipes* by interactively connecting boxes together using a graphical editor. Current applications include a photographic slide library and a geoindexing system.

Figure 1.1 shows the geoindexing application. Boxes in the recipe include database queries which index text documents according to the geographic locations to which these documents refer. The directed edges between the boxes represent the flow of data between the functions.

At the end of the recipe is a browser box which displays documents according to the geographic indexes the recipe creates. The default Tioga browsing paradigm allows users to visualize data results in a multidimensional space. Users navigate through their data using a flight-simulator interface. Additional browsers may be implemented by advanced users.

In Figure 1.2, the browser displays the final results of the recipe in Figure 1.1. Documents appear as rectangles in a latitude/longitude viewing space centered on California. Shading indicates the system's confidence in assigning a given document to a given location. In this figure, a rectangle has been selected, and the associated text from an environmental impact report appears in the display.

1.1.2 Motivation and related work

In the computational sciences, sequences of processing steps such as those that occur in Tioga dataflow diagrams are common. In these applications, the *lineage* of a datum consists of its entire processing history. This includes its origin (*e.g.*, the identifier of the base data set, the recording instrument, the instrument's operating parameters) as well as all subsequent processing steps (algorithms and respective parameters) applied to it.

Lineage is valuable to users for a variety of applications including investigation of anomalies and debugging. For example, lineage information allows the user to trace the impact of faulty source data or buggy programs on derived data sets. It also allows the user to investigate the source data or programs that produced an anomalous data set.

The perceived importance of data lineage has grown in step with the increased volume and widened dissemination of processed data sets. The amount of support for data lineage has grown as well. For example, new scientific data standards (*e.g.*, the Spatial

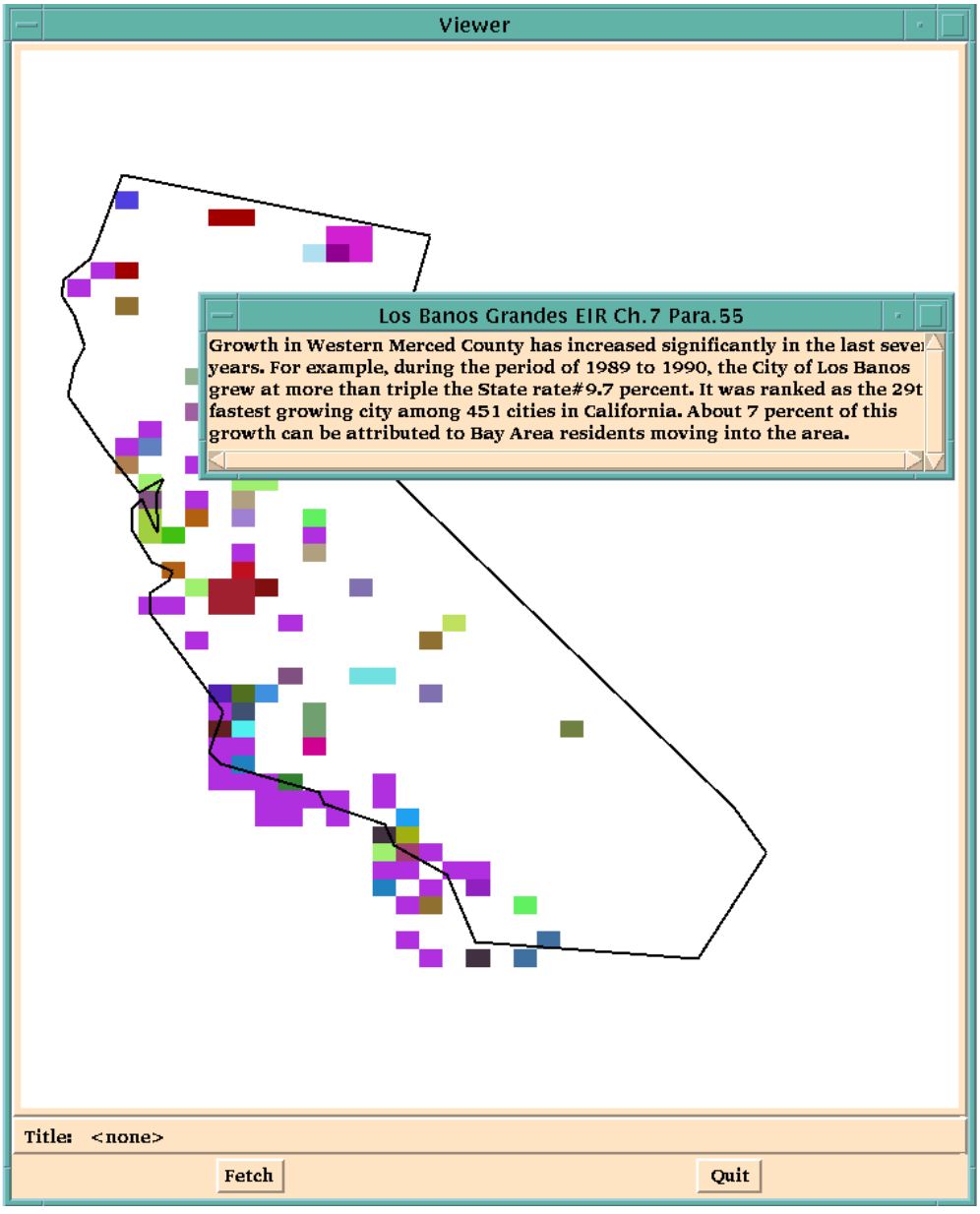


Figure 1.2: Data displayed in a Tioga browser.

Data Transfer Standard [28], the Spatial Archive and Interchange Format [44], and the draft Content Standard for Digital Spatial Metadata [14]) provide slots for annotations relating to lineage. Recent scientific workflow systems (*e.g.*, GIS databases such as Geolineus [24] and geophysical databases such as BigSur [9]) automate the process of lineage tracking by providing direct support in the workflow infrastructure. BigSur, for example, can supply the entire graph of processing steps that were applied to generate an image, or a list of all images produced using a given processing step.

1.1.3 Solution

We have identified two major deficiencies in existing data lineage systems. First, the systems do not provide adequate support for real-time access to the results of intermediate processing steps, even though such intermediate results are key to interactive data lineage queries. Second, the systems do not provide lineage that is sufficiently detailed to allow users to trace from a single element in an output data set to specific relevant items in the input data sets. We discuss these problems and our resolution of them in turn.

In Chapter 2, we discuss efficient implementation techniques to support queries to intermediate nodes (outputs of processing steps between the input source data and the final result). Because it is impractical to store all intermediate results, our approach is to perform selective buffering of certain results to reduce latency. We define the optimal buffer allocation problem to be the determination of the buffer contents that minimize the average response time to a sequence of user queries to intermediate nodes. We show that this problem has several characteristics that render traditional buffer management strategies ineffective. Since optimal buffer allocation is NP-hard, as we prove, we propose heuristic methods for buffer management of intermediate results.

We present a simulation of the behavior of these heuristics under a variety of conditions: we vary both the structure of the graph (*e.g.*, its size) and the user query pattern. We argue that history mechanisms that track user access patterns can be used to improve performance. We further show that graph structure and access pattern determine the factor of improvement that is possible. The performance enhancements we describe can be applied to minimize query response time in visual dataflow languages.

In Chapter 3, we examine a different aspect of the data lineage problem. The current state-of-the-art for processing data lineage queries is primitive. Most systems are

naïve and are not able to identify even the input sources. The most advanced systems assume that the processing history and source data associated with a datum can be stored as metadata [9]. A metadata-based approach to data lineage such as that followed by [24] and BigSur [9]) assumes relatively coarse-grained information will suffice.

However, some scientific applications require lineage at a much finer granularity than previously considered. For example, some applications require lineage at the level of pixels in images [4]. Imagine an application that takes many satellite images as input and outputs a composite. Such a program may involve a pre-processing step to register the images to a common grid. For debugging purposes, the user may wish to know which pixels of which original raster images were used in the construction of the composite. Such investigations may be difficult or impossible without data lineage: the user may not be familiar with processing steps that were written by an expert programmer. Further, tracing back manually through a number of processing steps is tedious and time-consuming, particularly if large data sets are involved.

For such applications, a metadata approach to data lineage would require storing information for each pixel. Outside of the database field, *e.g.*, in areas such as dataflow program debugging [56], detailed information has been stored. In [56], the data tags are used in realtime and then discarded. However, our applications require that the metadata be available after the program executes. It is impractical to store this volume of information, especially since the metadata does not lend itself well to compression because the information for each pixel may differ significantly.

We propose a novel approach for supporting such fine-grained data lineage. Our approach uses a limited amount of information about the processing steps to infer the necessary lineage information lazily (*i.e.*, on demand). In this way, the system avoids computing and storing such information in advance. The fine-grained data lineage technique presented here therefore complements coarse-grained metadata techniques.

In the absence of metadata, the most obvious way to identify relevant inputs is to invert the processing steps. Observe that each processing step is a function. A function f is said to be invertible if there exists some function f^{-1} such that for each element a input to f, $f^{-1}(f(a)) = a$. Unfortunately, only a limited number of functions (those that are one-to-one and onto) are invertible.

We introduce the notion of *weak inversion*, which applies to a larger class of functions. Each function that is weakly invertible has a corresponding function f^{-w}. f^{-w}

maps from the output of f to the input of f, but is not guaranteed to be perfectly accurate. Instead, the accuracy of f^{-w} is described by a number of weaker guarantees. We also introduce the notion of *verification*. Verification functions refine the set identified by f^{-w}.

We propose the implementation of weak inversion and verification as extensions to an object-relational database management system. We assume that expert users register their data processing functions in the DBMS and that the DBMS manages the application of these functions. These users register weak inversion and verification functions in the DBMS as well. Given a specific datum to invert, an inversion planner process infers which weak inversion and verification functions must be invoked, constructs a plan (a correct ordering of the weak inversion and verification functions), and then executes the plan by calling the corresponding sequence of functions within the DBMS.

Using the techniques described in this section, we are able to support data lineage queries. We now turn to a different visualization problem, that of constructing visualizations with appropriate information density.

1.2 Constant information density in zoomable interfaces

In this section, we first introduce the information density problem. We then describe previous work related to this problem. We next present the context for our solution, and the solution itself. Finally, we outline the remainder of this dissertation.

Visual representations of information can often help users understand complex data sets. However, if these visual representations have too much information, *i.e.*, if they are cluttered, their effectiveness can be significantly reduced. For example, clutter can result in overplotting, in which certain objects are not visible because they are occluded by other objects. In fact, multiple studies have shown that clutter in visual representations of data can have negative effects ranging from decreased user performance to diminished visual appeal. For example, Phillips and Noyes demonstrated that visual clutter decreases map reading performance [33]. In a related study, Springer showed that directory assistance operators locate targets more slowly on screens with more information [39]. For a review of a number of other studies of clutter, see [48].

In the situation that is the converse of clutter, visual representations have too little information. Such sparse visualizations can be unappealing and ineffective. For example, sparsity can result in inefficient use of the available display space, limiting the number of

objects a user can investigate and compare simultaneously.

1.2.1 Related work on clutter and sparsity

Researchers have attempted to address the clutter and/or sparsity problem in a number of ways. Previous work has examined appropriate amounts of information density for specific character displays, user interface screens, or images. Such work indicates that increasing density increases the time required to find information [19]. However, we know of no studies of information density in interactive visualizations.

Other work has focussed on the automated construction of displays with appropriate detail. Previous work on automatically creating visually appealing displays has considered the layout of objects [23]. However, the layout problem detailed in [23] has somewhat artificial objectives, requiring that no objects overlap and that the minimal amount of space be wasted (*i.e.*, that density be maximized). Other systems include: APT, which examines data set to be visualized and tries to pick the best presentation based on certain effectiveness criteria [26]; SageBrush, in which users specify some properties of the display and unspecified properties are inferred by the system [36]; and AUTOGRAPH, which takes as input a data set, a description of the data, and a statement of the intended use of the visualization, and outputs a graphical display [37]. These systems have several limitations. For example, they are largely designed to support small data sets and a restricted number of display types.

In other related work, researchers in the area of cartography have studied automated generation of maps of given scales. Many of these researchers use expert systems that generally are unable to produce maps of sufficient quality for commercial use [10]. Other cartographers have applied a new data structure (the Multi-scale tree) to the same problem [16]. Given a set of maps produced (manually or with a computer-assisted tool) at different scales, the algorithm is designed to automatically produce different views of the data as the user zooms. These views would have a constant number of active pixels (pixels that are being used to represent objects in the display). However, the algorithm described in this approach has several major limitations. It gives the user no control over which representations are chosen for display or over the final presentation. It supports only one density metric (ink, *i.e.*, the number of active pixels). Finally, examples are limited to the cartographic domain, and it is not clear the technique would generalize to other

applications.

As an alternative to automated techniques, a number of interactive visualization systems provide clutter reduction mechanisms that allow users to view detail selectively. These systems assume that a programmer or end user manually adjusts the data or the visual representations of the data to minimize clutter. For example, non-linear magnification schemes can be used to minimize clutter in the display. Fisheye views are one type of non-linear magnification scheme, in which objects are viewed as though through a fisheye lens so that objects in the center of the screen are given more display space than objects on the periphery [17]. Objects on the periphery can be displayed with reduced detail or can be omitted. Users can refocus the lens to change the detail shown for given objects. However, non-linear magnification displays can be disorienting for the user.

Ahlberg and Shneiderman use an alternative mechanism that allows the user to filter objects in the display dynamically [1]. This technique does not differentially filter subdivisions of the display. Therefore, if the underlying data is of non-uniform distribution, the resulting visualizations can have regions that are too dense and/or too sparse. Fishkin and Stone extend Ahlberg and Shneiderman's model by providing movable filters that the user can position manually in the display [15]. However, Fishkin and Stone's work requires the user to manually choose the areas to which filtering is applied. Note that these techniques do not modify the visualization, but rather the data set being visualized.

A final type of interactive system, the *semantic zoom* system, addresses clutter and sparsity by supporting multiple graphical representations of objects. The user is provided with an interactive visualization, and a representation with appropriate visual complexity for the current view appears in the display.

More concretely, semantic zoom systems typically use a spatial metaphor in which objects are displayed in a two-dimensional canvas over which the user can pan and zoom. The user's distance from the canvas is known as their *elevation*. Note that when the user views an object from a high elevation, it may appear quite small. However, when they zoom in on it, it gets larger, occupying a larger percentage of the display. As the object gets larger, it is appropriate to make its graphical representation more complex. Therefore, different graphical representations are appropriate for different elevations.

For example, suppose a city is represented as a dot when the user is at a high elevation. When the user zooms in on the city, a text label may be added to the representation, or a city map may replace the text. Significant examples of semantic zoom systems

include Pad and Pad++ [32, 5].

For a variety of reasons, semantic zoom systems lend themselves to a more sophisticated approach to the problems of clutter and sparsity. In the next subsection, we outline one such system, which provides the context for our solution.

1.2.2 The DataSplash environment

In this subsection, we describe the DataSplash environment. DataSplash is a direct manipulation semantic zoom system in which users can construct and navigate visualizations [2]. This system has been implemented on top of the POSTGRES object-relational database management system [43] and released as freely available software [35].

In DataSplash, all objects in a canvas are organized into layers. Each object is a member of exactly one layer. Each layer is associated with exactly one database table. DataSplash uses the spatial metaphor of a two-dimensional canvas common to semantic zoom systems. Each row in the table is assigned an x,y location in the canvas, *i.e.*, the rows are scattered across the canvas, giving an effect similar to a scatter plot. The x,y locations are derived from data values in the rows. For example, if the user has a table of United States cities with latitude and longitude columns, x and y can be assigned to the longitude and latitude values of each city.

At any time, the user can create an object in DataSplash's paint program interface and duplicate that object for every row in the database table. As a result of this duplication operation, a copy of the object appears at the x,y location of every row in the table. The effect is like splashing paint across the canvas, coating every scattered row. The user may also associate display properties of objects with columns in the table, *e.g.*, height, width, color, and rotation of each splash object can be derived from values in the columns of its row. Continuing the example of a visualization of United States cities, the user may specify that a circle is to be drawn at the x,y location of each city. The user may further specify that the radius of each circle be proportional to the population of that city.

Users can pan and zoom above the resulting two-dimensional canvas. When they zoom, they change their elevation above the canvas. Elevation is expressed as a percentage of a user-specified maximum elevation. DataSplash allows users to control the range of elevations at which each layer is rendered. To this end, each layer appears as a vertical bar in a *layer manager*. The top of the layer bar represents the highest elevation at which

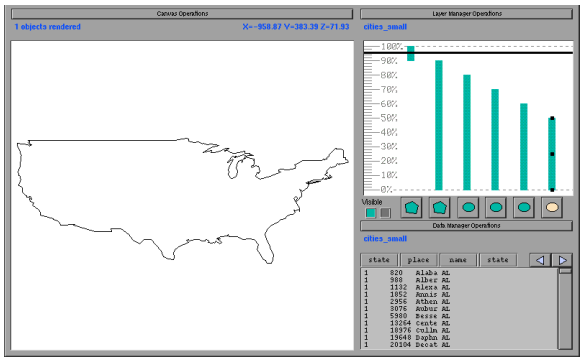


Figure 1.3: DataSplash, showing United States cities application seen from a high elevation.

objects in the layer are rendered. Similarly, the bottom of the layer bar represents the lowest elevation at which objects in the layer are rendered. The user's current elevation is shown with a horizontal *elevation bar*. Any layer bar that is crossed by the horizontal elevation bar is considered to be active and objects in the corresponding layer are rendered. An icon of the type of object displayed by each layer appears in the button below its layer bar.

Users can graphically resize layer bars in the layer manager. In addition to resizing layer bars, users can also shift them up and down and add or delete new layer bars. These operations are performed in the same way as in traditional paint programs, *e.g.*, to resize a layer bar, the user drags a resize handle. DataSplash has a number of additional features, but we do not discuss them here because they are not relevant to information density.

Figures 1.3 and 1.4 show the display and layer manager of a sample application. The application contains six layers. The first layer contains an outline of the United States. The second layer contains outlines of each state in the United States. Note that these two representations are mutually exclusive, *i.e.*, their elevation ranges are disjoint, so only one can be visible at a given time.

The third through sixth layers are based on city data. The data has been segmented according to population size; the third layer contains cities with the highest population, the fourth layer contains slightly smaller cities, *etc.* Each city in each entry is drawn in a circle and cities with higher populations are drawn as larger circles.

In Figure 1.3, the layer manager appears in the white rectangle in the upper-right. The horizontal line (elevation bar) indicates that the user is at a high elevation. The only layer active at this elevation is the United States outline layer. This layer is rendered in the

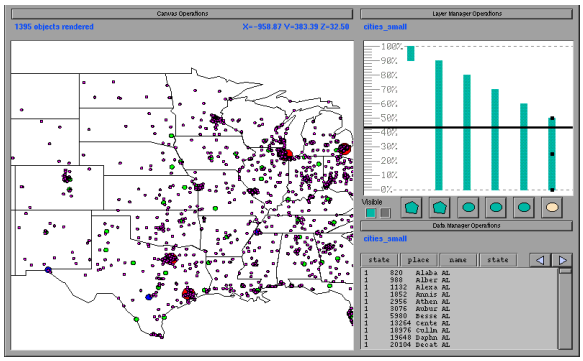


Figure 1.4: United States cities application seen from a low elevation.

display on the left.

In Figure 1.4, the user has zoomed to a lower elevation, as can be seen from the position of the horizontal elevation bar in the layer manager. Note that the elevation bar now intersects five layers; at this elevation, the state outlines and city circles layers are active. Therefore, the objects associated with these layers are all visible in the display on the left.

Another feature of DataSplash is its support for *portals*. Portals are windows that open onto other canvases. DataSplash users can automatically generate a portal for every row in a database table. For example, the user can easily specify that each city in a visualization of the United States should have a portal that goes to a map of that city. A portal history mechanism allows users to go backwards and forwards between canvases.

The features described above have been implemented and tested informally. Although users generally respond positively to the system, we have observed that they have difficulty constructing applications that display an appropriate density at all elevations. To understand this, consider the simplest possible application - one in which the representation of the objects never changes (scaling aside) as the user zooms in and out. The *display* seen by the user is a fixed-size viewport onto an underlying, or native, coordinate space defined by the x,y values of the objects. The objects never change their native x,y position, so object density in the native space obviously never changes. However, any change in elevation implies a change in the area of the native space visible in the display, which implies (in general) that the display contains a different number of objects. This in turn implies a change in display object density. As a result, the same visualization can be appealing at

one elevation, cluttered at higher elevations, and sparse at lower elevations.

Unfortunately, when users see a visualization at a given elevation, they find it difficult to imagine how that same visualization will appear when viewed from other elevations. Therefore, poor visualization quality may result if users do not visit many elevations whenever they modify applications, checking for appropriate detail. This is a tedious, highly iterative process, which is made even more difficult by two additional facts: (1) objects may change representation as users zoom (if they have different graphical representations in different layers) and (2) users of current systems must visually and subjectively judge appropriate density. In the following subsection, we discuss how we have ameliorated this process.

1.2.3 Solution

We have identified a cartographic precept that defines appropriate transition points between different graphical representations. The Principle of Constant Information Density states that the number of objects per area should remain constant. More generally, the amount of information should remain constant as the user pans and zooms above a display [45, 16]. In Part II, we describe our application of this principle to interactive visualizations of cartographic and non-cartographic data.

In Chapter 4, we discuss a pilot study of user response to interactive visualizations with varying densities. Our results, though highly preliminary, suggest that users avoid higher density elevations in preference for lower density elevations and that users pan more frequently in displays that have lower density. We propose that application designers use our solutions, as described in Chapters 5 - 7, to ensure constant information density and thereby minimize unexpected user navigation patterns.

In Chapter 5, we introduce a system that helps users construct interactive visualizations with constant information density. This work is an extension of the DataSplash database visualization environment. Recall that our experience with DataSplash indicates that users find it difficult to construct visualizations that display an appropriate amount of detail. In Chapter 5, we introduce VIDA$_0$ (VIDA is an acronym for Visual Information Density Adjuster). VIDA$_0$ is an extension to DataSplash based on the Principle of Constant Information Density. This extension gives users feedback about the density of visualizations as they create them. In this way, our extension helps users create interactive applications

that display the appropriate amount of detail at all times.

The techniques described in Chapter 5 help users manually construct applications in which overall display density remains constant. In the context of semantic zoom systems, this approach ensures uniformity in the z (elevation) dimension, but does not extend naturally to ensuring uniformity in the x and y dimensions.

In Chapter 6, we present an extension to VIDA_0. This new version, VIDA, automatically selects user-provided layers for display such that the resulting visualizations are uniform in the x, y, and z dimensions. In VIDA, users express constraints about visual representations that should appear in the display. The system applies these constraints to subdivisions of the display such that each subdivision meets a target density value. We describe our algorithm, implementation, and the advantages and disadvantages of our approach.

In Chapter 7, we discuss an extension to VIDA that semi-automatically modifies the contents of layers, thereby constructing visualizations that have constant information density. While the techniques of Chapter 5 and Chapter 6 focus on deciding when to display given representations, this chapter focuses on ways to actually change those representations. To that end, we present a taxonomy of data manipulation and graphical operations that can be performed to adjust density. We discuss interfaces with which users can semi-automatically adjust the density of a group of objects and adjust the density of an entire visualization. While the concepts in this chapter have not been implemented, the design is presented in the context of the VIDA architecture.

Finally, in Chapter 8, we discuss a new navigation technique for constant information density systems. We introduce a novel zoom method, *goal-directed zoom*. In a goal-directed zoom system, users specify which representation of an object they wish to see. The system automatically zooms to the elevation at which that representation appears at appropriate detail. We have extended VIDA to support end-user construction of visualizations that have goal-directed zoom. We present a sample user interaction with this environment.

In Part III, we discuss future directions and conclusions.

Part I

Data lineage

Chapter 2

Buffering

2.1 Introduction

In Section 1.1, we defined data lineage queries over dataflow programs.[1] Further, we saw that data lineage queries require access to intermediate computations. When such results are not buffered effectively, access to them can incur significant performance penalties. In this chapter, we discuss buffering of intermediate results in dataflow diagrams to reduce latency.

Dataflow languages apply a sequence of operations to specified inputs. In many cases, the final output of a dataflow diagram is the only result examined by a user. However, when performing tasks such as debugging or tuning, a user may wish to view intermediate results. In a naïve implementation, intermediate results are not saved when a dataflow diagram is executed. As a consequence, if a user asks to view intermediate results, these results need to be recalculated. Since computation costs may be extremely high, the delay in response time can be substantial. A more sophisticated implementation can support buffering of intermediate results. Because blindly buffering all intermediate results may not be feasible, such a system must attempt to select for buffering those intermediate results that minimize latency.

We examine strategies for buffering of intermediate results in dataflow diagrams in the context of Tioga [41]. Tioga was motivated by the needs of scientific users in the

[1] Much of the material in this chapter appears in [53]. Copyright 1995 IEEE. Reprinted, with permission, from *Proceedings of the 1995 IEEE Symposium on Visual Languages*, Darmstadt, Germany, Oct. 1995, pp. 187-94.

Sequoia 2000 project [42]. In a typical task, these users will construct a recipe, run it on a specified set of inputs, and view the final result. If this result contains an anomaly or some unintuitive or unwanted result, users might want to perform the following types of actions:

- **search query**. In this case, the user examines intermediate results of Tioga boxes to locate data of interest, *e.g.*, the source of an anomaly. Intermediate results may be viewed by placing browsers at arbitrary points in the recipe.
- **modification query**. Users may want to tune a recipe. In this case, they will rerun it using different parameters as input to specific functions. Alternatively, they may wish to modify the code of a particular box and rerun the entire recipe with the new box. Finally, they may incrementally add boxes to develop an application, as supported by systems such as Weaves [21].

Note that some tasks may involve both search and modification queries. For example, debugging may entail a sequence of search and modification queries to locate and correct a faulty processing step or data.

Attempts to reduce the latency of search and modification queries raise several interesting issues. For example, we observe that buffering of intermediate results can significantly improve performance. In this chapter, we use a simulator to examine buffer management strategies to improve the performance of Tioga on search queries. Because modification queries have many similar characteristics, such as the dependencies between results, our findings have direct relevance to modification queries.

In Section 2.2, we define the problem of optimal buffer allocation. In Section 2.3, we present our assumptions, and in Section 2.4, we describe our mechanism for generating graphs that are input to our model. In Section 2.5, we present our simulation model and our heuristics, and in Section 2.6, we present our results. Finally, in Section 2.7, we conclude.

2.2 Problem definition

Consider the recipe graph of Figure 2.1 in which box A is an input and box I represents the final output. Suppose the existence of a buffer with limited space. The buffer can contain the results of some set of boxes; the size of these results must sum to less than or equal to the space available in this buffer. Now suppose the buffer is currently empty and

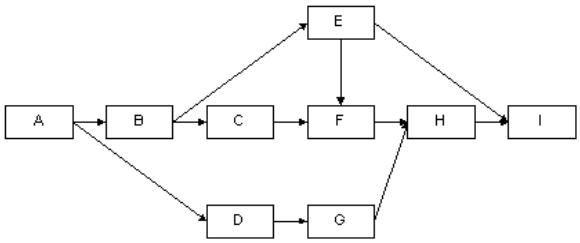


Figure 2.1: Structure of a sample dataflow diagram.

the user queries box F. The results from boxes A, B, C, and E are calculated as part of the computation of box F, and the results of A, B, C, E, and F are therefore all candidates for buffering. However, if there is insufficient buffer space to store all these intermediate results, the system must choose which box results to buffer. This is a complex problem because the *worthiness* of box results (their potential to reduce latency) depends on a variety of complex considerations, beginning with compute cost and buffer space requirements.

The worthiness of each box result is also significantly affected by the structure of the recipe graph. Box results in the buffer can save time during the computation of other boxes by obviating the need to compute all their ancestors. We say that a box is *guarded* if some box along each path from the box to the target (the queried box) is buffered. For example, if the results from boxes C and E are in the buffer and box F is being queried, only box F needs to be computed during this move (since boxes C and E have guarded boxes A and B, making their computation unnecessary). We additionally observe that the current residency of the buffer pool affects the worthiness of other box results.

Finally, the probability that a box result will be accessed impacts its worthiness. (In our simulator, this probability is constructed according to a probabilistic move model described in detail in Section 2.5.) We define a *query* to be the user's request to view the results of a single box. A *query path* is a sequence of queries. The current query is the user's *position* in the query path. At each position, a possible *buffer allocation* may be made. A set of buffer allocations for the entire query path is called a *solution*.

We assume the existence of a *configuration* that includes the following information: the structure of the recipe graph, the user's current position in the recipe graph, the current contents of the buffer, and the probability distribution of the user's expected movements. Given such a configuration, we define the optimal buffer allocation at a given position as the one that will minimize the average response time to an unknown (but probabilistic)

future sequence of queries on intermediate results. The buffer allocation at a given position is drawn from a set of *candidate* results, those that are already in the buffer pool or must be computed at the given position.

As a result of the complications listed in the discussion of worthiness above, calculating the optimal buffer allocation is in fact NP-hard. This can be shown as follows by a polynomial reduction from the Knapsack problem [20] to our problem of optimal buffer allocation. In such a reduction, compute times and buffer space requirements correspond to the value and size of objects to be placed in the knapsack. Intuitively, the search space is extremely large since it must consider the impact of all possible allocations on the latency of all possible query paths that could be followed in the future (and in turn all the buffer allocations for each position in each path). Performing an exhaustive search of all buffering solutions for all future paths is not feasible. Naïvely calculating the optimal allocation for positions in short query paths through graphs with a small number of boxes (*e.g.*, 10) took days on a DEC Alpha computer.

Attempts to apply known techniques for solving related problems fail, either because they ignore critical aspects of our problem or because they assume information not available to us. Traditional buffer management strategies such as LIFO and LRU are ineffective due to the following characteristics of our problem: (1) box results can guard other boxes; (2) the sizes of box results vary; (3) the compute times of boxes vary; and (4) the future reference stream is impossible to predict. Because traditional caching and buffer management techniques [38, 13, 11] do not consider guarding, items of variable size, or variable fetch costs, they make poor buffer allocation decisions for intermediate results. Register allocation is in many ways more similar to the optimal buffer allocation problem we are considering. However, heuristics to solve register allocation are predicated on an understanding of the future reference stream [40]. This understanding allows the register allocation heuristics to eliminate many results from consideration. Because the reference stream in search query paths in dataflow diagrams is unpredictable, it can not be constrained. Finally, although techniques exist that minimize buffer usage during the execution of a dataflow diagram [46], these techniques do not consider retaining intermediate results after computation is complete.

Although strategies such as dynamic programming can reduce the computation time, a less compute-intensive strategy is to use heuristics to make near-optimal buffer allocations. As a consequence, we have developed novel heuristics that are more appropriate

to the optimal buffer allocation problem. In this document, we compare the behavior of a number of these heuristics on a variety of graph types and user access patterns.

2.3 Assumptions

We make several assumptions with respect to the operation of the dataflow system and the structure of the dataflow graph. We assume the existence of recipe graphs that have already executed and materialized final results. All inputs and final results are saved as tables in the underlying DBMS. However, we assume that intermediate results of recipes are not saved. We assume that the system has bookkeeping information that records the exact compute time for each box, as well as the exact size of its output.

For simplicity, we assume the common case in which each graph has one box which is a *terminal* box. The terminal box has no outputs and is saved in a DBMS table. We refer to boxes that do not output data except to the terminal box as *sinks*. These boxes are of interest because queries to them can require the computation of a large number of boxes, therefore providing a large set of results as candidates for the buffer pool. Boxes that do not take input from other boxes are called *sources*. In a query path, a user views the results of the terminal box and then performs a sequence of queries on the recipe graph, visiting a number of boxes before terminating the search. We assume that this sequence is not known in advance and that the terminal box is never revisited.

We assume the existence of a workspace in which computations are performed. For each recipe, we assume the existence of a separate buffer of limited size. Each time a user asks to view an intermediate result, our strategies determine which box results they would like to retain in the buffer after its computation. Desired box results in the workspace are copied into the buffer. We assume that box results may be buffered only in their entirety. We assume without loss of generality that this buffer space is on disk. In most cases the buffer space available will not be sufficient to store all intermediate results. Our goal is to choose which intermediate results to buffer to minimize the expected average latency of future search queries.

2.4 Graph generation

Because only a limited number of recipes have been developed in Tioga to date, we randomly generate recipe graphs for our tests. The parameters for generating these graphs are based on conversations with Earth scientists from the Sequoia 2000 project [42] and on observations of graphs developed in Tioga and other dataflow systems such as AVS [49], Data Explorer [25], and Khoros [34]. That is, we developed an algorithm and chose parameters that produced graphs that "look right" to users.

Our first observation is that boxes in dataflow diagrams, both in Tioga and in other systems, are typically composed of groups. While groups have a relatively high degree of interconnectivity, there tends to be a relatively low degree of connectivity between groups in a graph.

Therefore, we begin by generating groups as follows. The group generator creates a certain number of boxes (a value randomly chosen from a specified range). The graph generator takes as input a range of orders of magnitude for buffer sizes and compute times. To assign a value from one of these ranges, we first randomly select an order of magnitude from the specified range. We then randomly select a number from within that order of magnitude. The resulting distribution of numbers generated resembles an exponential distribution.

In our studies, we focus on graphs in which buffer sizes vary by up to two orders of magnitude (between 1 and 100) and compute times vary by up to five orders of magnitude (between 1 and 100,000). These values are based on discussions with Earth scientists about typical dataflow applications. After the boxes have been assigned buffer sizes and compute times, we add edges that result in acyclic graphs with a controlled *branching factor* (the average number of edges per box). In this study, we generated graphs ranging from a low branching factor of approximately 1.2 to a high branching factor of approximately 1.8.

The graph generator makes calls to the group generator a specified number of times (a typical value was three, although larger values were used when we wished to construct larger graphs). It then adds edges between the groups as follows. The graph generator connects a source, sink, or intermediate box in the first group to a source, sink, or intermediate box in the second group according to a specified probability function. In the results presented in this chapter, plausible values are chosen based on informal observations of existing dataflow diagrams. We designate the box in the first group a source, sink, or

intermediate box with probability 10%, 80%, and 10%, respectively. The box in the second group is a source, sink, or intermediate box with probability 25%, 70%, and 5%, respectively. The probability distribution of these connections controls the relative numbers of sources and sinks in the final graph. Finally, after all groups have been connected, all sinks are connected to a terminal box.

2.5 Simulation model

We next implemented a simulator that would measure the performance of a variety of buffering strategies on various graphs. We developed a move model that specifies the sequence of intermediate results examined by a user. Note that for a given pair of boxes, if the output of the first box is passed as input to the second box, we say the first is the *parent* of the second and the second is the *child* of the first. Under these definitions, if a user is positioned at a given box, it is possible for them to:

- move **backward** in the graph to a parent (*e.g.*, from box F to box C in Figure 2.1).
- move **forward** in the graph to a child (*e.g.*, from box C to box F).
- move **sideways** to a spouse box that shares a child (*e.g.*, from box F to box G).
- move to a **random** box in the graph (*e.g.*, from box G to box B).
- **reset** (end the query).

Each of these five possibilities is assigned a probabilistic value; the five values sum to one hundred percent. The probability of resetting indirectly controls the length of a single query path. We studied a variety of probability distributions including, for example, a largely backwards, short query path characterized by 50-10-15-10-15 and a relatively random, long query path characterized by 13-13-18-53-3 (backward-forward-sideways-random-reset).

The simulator generates a complete query path and passes it to procedures that mimic the behavior of buffering strategies on that path. At each position, the buffering strategy has a list of candidate box results that could be retained. This includes results that existed in the buffer previously as well as results that must be calculated at the current position. A *viable* candidate is one that will fit in unallocated space in the buffer. At each

position, the strategies assume they have the entire buffer space to allocate and fill it according to their heuristic. The allocation ends when the strategies determine the buffer is full or when they determine that none of the unbuffered candidate box results is viable.

For each heuristic, the cost of the entire query path using its solution is recorded. We have examined a large number of strategies in this way. For the sake of brevity, we discuss only the following in this document:

- **No Buffering**. No intermediate results are cached. This represents the worst case.
- **First-in-first-out (FIFO)**. At each position in the query path, FIFO buffers box results in reverse timestamp order. Timestamps represent the creation time of a box result within a query path. Within a position, we assume the necessary boxes are computed in topological order, since no box may be computed until its ancestors are computed. Timestamps are therefore assigned according to a post-order, depth-first traversal.
- **Random Average**. At each position in the query path, this strategy selects box results uniformly at random and attempts to buffer them. A buffer allocation is complete when the buffer is full or all available box results have been buffered.
- **k-Random**. Random Average as above is run k times on a fixed query path; the k-Random solution is the one with the best running time. In this way, we use the best allocation of a randomly chosen set of allocations to approximate roughly the optimal allocation. For data in this chapter, values for k range between 64 and 256; separate simulations established that in most cases higher values of k yield only marginal improvements.

  Note that k-Random approximates the behavior of an offline algorithm, or an oracle that acts with knowledge of future moves. Such behavior may not be achievable in the general case; however, it gives us a value against which to compare our heuristics.
- **Path Cut**. At a high-level, Path Cut's heuristic is to minimize the cost of hypothetical backward query paths. These hypothetical paths are the set of all paths that begin at any node and consist only of backward and reset moves. The cost of such backward paths can be decreased by the buffering of their midpoint. Results of boxes that have some combination of the following characteristics are therefore desirable: (1) midpoint

of multiple backward query paths; (2) midpoint of at least one expensive backward query path; or (3) midpoint of at least one backward query path that is likely to occur. At each position in the fixed query path, Path Cut assigns the midpoint of each hypothetical backward path a path cut value (PCV). The worthiness of a box result is the sum of its PCVs for all paths. A greedy algorithm is used to attempt to buffer box results in the expected order of their worthiness.

The worthiness function is computed as follows: assume that the replacement algorithm is making a buffering decision while computing a current box c. For every box in the recipe graph, the algorithm constructs all paths to each of its ancestors. It calculates the sum S_p of the compute costs for all boxes in a path p from a box n to an ancestor a. The compute cost of a single box is the sum of the costs of all its ancestors (all the boxes on which it is dependent for input) assuming that no box results are buffered. The algorithm identifies the computational midpoint m of path p. m is the box along p such that the sum of the compute costs between a and m is greater than or equal to 50% of S. The algorithm then calculates two probabilities. For this calculation, the algorithm assumes that it has perfect information about the probability distribution of the moves described above. First, it calculates the probability $\mathrm{P}(p|n)$ of the path between n and a occurring given that the user reaches box n (this is the probability of a backwards move to the power of the length of p). It then calculates the probability $\mathrm{P}(c \rightarrow n)$ that n will be the box visited immediately after c. The PCV of m is equal to $S_p \times \mathrm{P}(p) \times \mathrm{P}(c \rightarrow n)$, or the cost of the path weighted by the probability the path will be reached and followed from the current box c. The worthiness is the sum of all PCVs for a given box. This value approximates the ability of the box to minimize the cost of hypothetical backward query paths.

- **Path Cut No Probabilities (NP)**. This heuristic is identical to that above, with the exception that the PCV assignment considers no probabilities, *i.e.*, the PCV of m is equal to S_p.

We simulated the above strategies on a variety of graphs and with a variety of access patterns. We present the results of these simulations in the next section.

2.6 Results

This section quantitatively demonstrates the benefits of buffering of intermediate results. First we show that significant gains can be achieved by such buffering. We next show how these benefits vary with heuristic, graph structure, and access pattern. We then examine the behavior of heuristics for varying buffer sizes. Finally, we discuss the results of additional experiments we conducted.

We begin by characterizing the maximal reduction in latency that can be achieved by buffering of intermediate results. We assume the buffer is empty when computation begins. The best possible performance occurs with 100% buffering, *i.e.*, when every box result that is computed is inserted in the buffer and not removed until queries on the graph are complete. (The computation cost in this case is not simply the cost of executing the recipe since a user may choose to examine only a subset of the intermediate results.) In this situation, the buffer space needed is at most the sum of the space requirements of all boxes in the graph, since each box result need be stored at most once.

Figure 2.2 presents the relative performance of No Buffering and 100% buffering solutions to query paths for a variety of graph structures and access patterns that can be characterized as described in Table 2.1. Recall that the values chosen are based on discussions with Earth scientists regarding typical applications. The *Size* column notes the average number of boxes in the graph. Graphs with an average number of 18 boxes consist of three groups. The *BF* column characterizes branching factor (low of 1.2 and high of 1.8). The *C* and *B* columns indicate the number of orders of magnitude variation among the compute times and buffer space requirements of box results in a recipe graph. *P(back)* and *P(random)* indicate the probability of backwards movement and random movement. *PL* represents the path length, indirectly controlled by the probability of resetting. For simplicity, in the table the values of the last three items are classified as high and low. The complete move probability distributions are 50-10-15-10-15 (baseline), 13-13-18-53-3 (random), 53-13-18-13-3 (bushy), 50-10-15-10-15 (variable), and 53-13-18-13-3 (big).

For each type described above, we generated dozens of graphs and ran thousands of query paths within each graph, measuring the compute costs in each case. Figure 2.2 presents the average of the results in the case of No Buffering and 100% buffering. We observe that if no buffering is done, three independent conditions can make query paths expensive. First, longer query paths are more expensive. Second, query paths through

	Size	*BF*	*C*	*B*	*P(back)*	*P(Random)*	*PL*
baseline	18	low	6	1	high	low	short
random	18	low	6	1	low	high	long
bushy	18	high	6	1	high	low	long
variable	18	low	6	3	high	low	short
big	54	low	6	1	high	low	long

Table 2.1: Graph structures and access patterns.

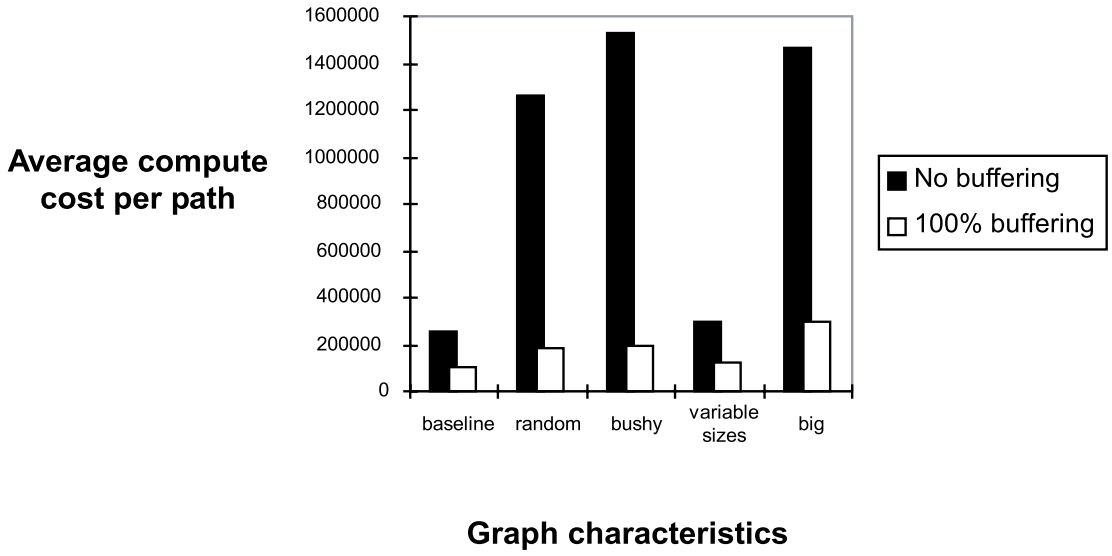


Figure 2.2: Compute costs for varying graph structures and access patterns.

larger graphs are expensive (because the computation of a single box may depend on the computation of a larger number of ancestors). Third, query paths in bushy graphs are more expensive (a higher degree of connectivity also implies that the computation of a single box may depend on the computation of a larger number of ancestors).

It is apparent that buffering 100% of the results as they are computed significantly reduces the average compute time per path. (Recall that there is a cost associated with computing the results the first time, so even complete buffering can not reduce computation time to 0.) The greatest gain is achieved for the random graph; 100% buffering in this case is 13% of the cost of No Buffering. This data clearly demonstrates that buffering of intermediate results is desirable.

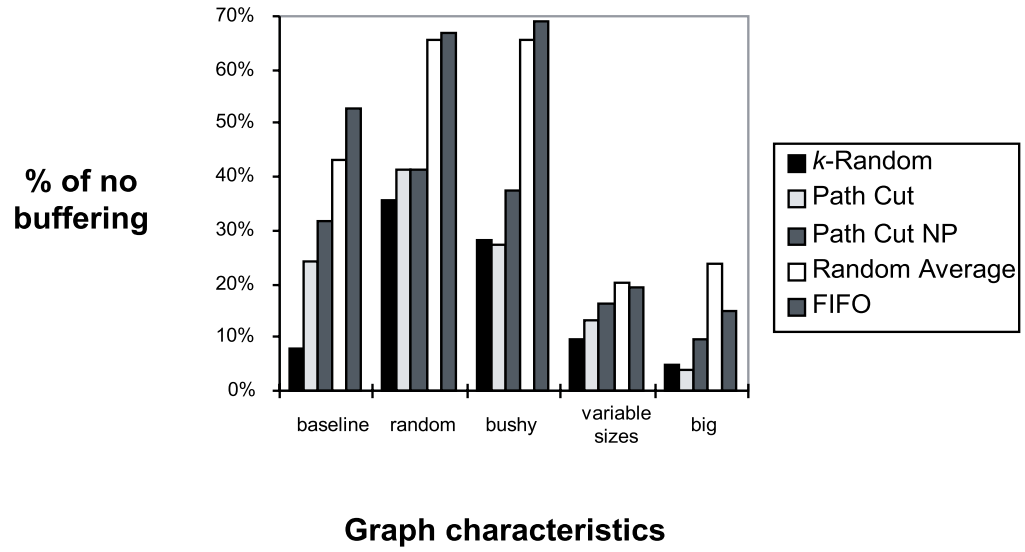


Figure 2.3: Heuristic performance with 10% buffering.

We next examine the improvements that would be possible with a smaller buffer. In Figures 2.3 - 2.5, we show the relative benefits of the various buffer management schemes under a variety of conditions. We note that there is a certain minimum computational cost that will be incurred by any solution. Note that this cost is represented by 100% buffering. Therefore, we can compare solutions according to the computational cost they incur above this minimum. Specifically, we eliminate the minimum computation cost before comparing each heuristic solution with the No Buffering solution as follows. If a heuristic cost is h, the 100% buffering cost is B_{100}, and the No Buffering cost is NB, the Y axis contains $\frac{h-B_{100}}{NB-B_{100}}$. Values close to 0% mean the heuristic closely approximates 100% buffering; higher values mean the heuristic performs in a manner similar to No Buffering.

Figure 2.3 presents the performance of various heuristics on the same graphs, access patterns, and query paths detailed in Figure 2.2, assuming a buffer 10% of the size of the 100% buffer. Observe that graph structure and access pattern affect the behavior of the heuristics. Considering graph structure, observe that the heuristics are clustered together fairly tightly for the variable size graph set. This occurs because each heuristic attempts to fill the fixed size buffer completely. As the buffer fills, an increasingly small number of box results are viable candidates. As a consequence, there is a certain set of box

results with small space requirements that is chosen by most heuristics. This similarity in buffer allocations compresses the difference among the heuristics. Considering variations in access pattern, observe that in the random graph set (which has longer query paths), the performance of the heuristics is quite poor compared to that in the other graph sets. This is not because the heuristics are making poor buffering decisions in the random graph, but rather is a result of the small buffer size. Even in the optimal solution for these query paths, items must be removed from the buffer and calculated again later.

Also observe that in many cases, heuristics come close to the performance of 100% buffering. Specifically, k-Random and Path Cut tend to do extremely well. The relative performance of the heuristics for each graph type and access pattern is relatively consistent, with a few interesting exceptions. For example, in most cases Path Cut tends to outperform Path Cut NP. However, it loses its advantage on query paths with a high degree of randomness. This implies that the worthiness assignment being used by Path Cut may be most effective for access patterns in which the user has a high probability of moving backward through the graph. Note that in the bushy and big graphs, Path Cut actually outperforms k-Random. This is because k-Random has a much lower chance of finding a good solution for a long query path than for a short query path (since k is fixed and the set of solutions for a long query path is much larger than the set of solutions for a short query path). Also observe that FIFO does quite poorly in general, often worse than Random Average. Intuitively, since FIFO buffers the most recently generated box results, it always attempts to buffer the result of the box that is being visited. Since the user most often moves away from that box, often to ancestors that have no dependence on it, this is a poor strategy.

Figures 2.4 and 2.5 show the performance of heuristics as a function of the size of the buffer. Error bars represent 95% confidence intervals. Note that the performance of the heuristics quickly converges as the buffer size increases. Observe also that for certain strategies, a buffer that is approximately half the size of the maximum buffer can yield the same performance as the maximum buffer. This is largely because many query paths do not access all box results, and so much of the maximum buffer remains unused.

We tested many variants of the heuristics described in this chapter. In general, the other heuristics we examined performed slightly worse than Path Cut and slightly better than Path Cut NP.

We also investigated the usefulness of approximate rather than perfect information about the probability distribution of the user's movements. The performance of heuristics

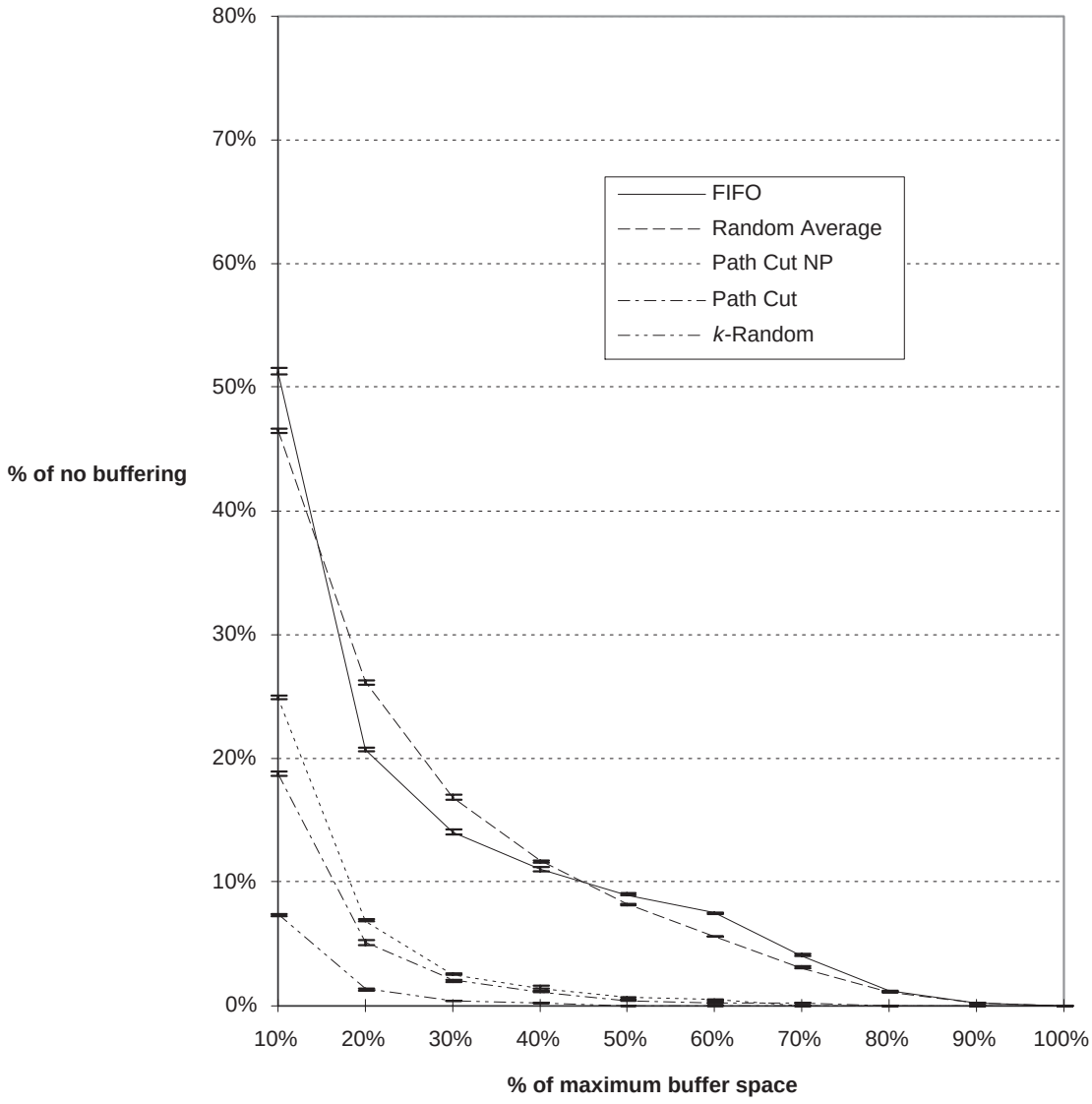


Figure 2.4: Heuristic performance for baseline case.

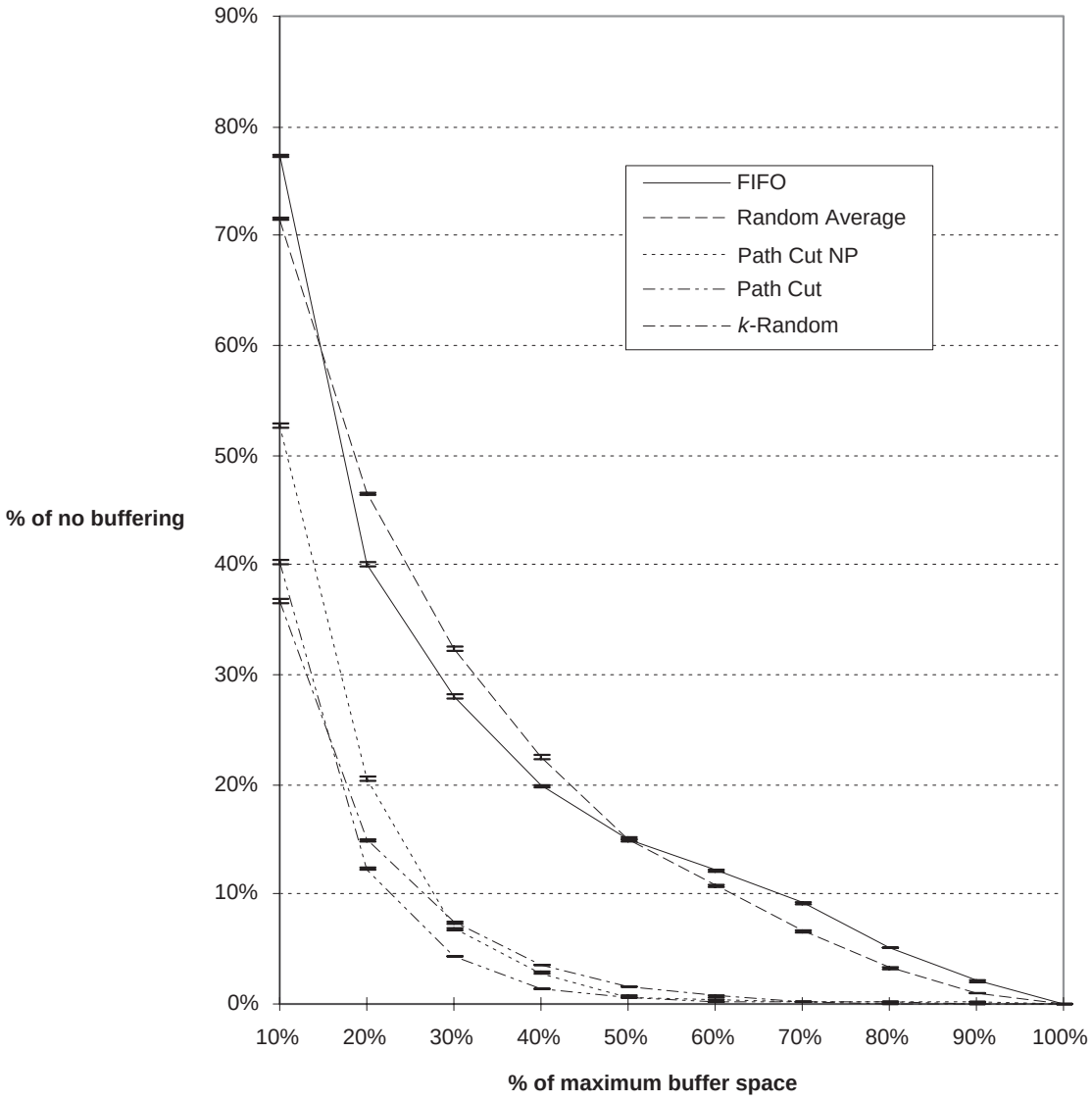


Figure 2.5: Heuristic performance with high branching factor and long query paths.

using approximate information is quite close to that of the heuristics using perfect information. Since history mechanisms may be used to approximately predict the user's future movements, we conclude that keeping history about the user's access patterns is advisable. We further observe that information about performance can be presented to the user. With such a mechanism, the user can make informed decisions about their queries, *e.g.*, they can avoid boxes that will be expensive to compute.

2.7 Conclusions

We have seen that buffering of intermediate results can significantly reduce the latency of search queries in dataflow diagrams. Use of a relatively small buffer can provide substantial improvements over no buffering. Further, traditional strategies such as FIFO are much less effective than the new heuristics that we propose. These heuristics approach the maximal improvement possible. The most effective heuristics make predictions about the user's access pattern, suggesting that a history mechanism is warranted. By buffering intermediate results using these heuristics, we can significantly reduce the latency of data lineage queries.

Chapter 3

Weak inversion and verification

3.1 Introduction

Data lineage support should be judged not only by its efficiency, but by its quality as well.[1] In Chapter 2, we discussed a method by which intermediate results can be retrieved effectively when queried by the user. In this chapter, we turn to the problem of retrieving more refined results.

In this chapter, as an example, we consider the scenario of a scientist applying a series of processing steps to an atmospheric data set and then viewing the result, a plot of cyclone tracks, in a database visualization system. Suppose the scientist sees an anomaly and wants to identify the input data that contributed to the unexpected value. The database system may be able to trace the lineage of the anomaly at a coarse level, using metadata. For example, it may be easy to determine that the cyclone track point originated in a particular time step of the atmospheric data. However, tracing from a specific cyclone track point in the processed data set to a particular atmosphere pressure value in the source data set is not feasible using such an approach; the amount of metadata required is comparable to the size of the data set.

We propose a novel method to support this type of fine-grained data lineage. Rather than relying on metadata, our approach computes lineage on-demand using a limited amount of information about the processing operators and the base data. We introduce the

[1] Much of the material in this chapter appears in [54]. Copyright 1997 IEEE. Reprinted, with permission, from *Proceedings of the 13th International Conference on Data Engineering*, Birmingham, England, Apr. 1997, pp. 91-102.

notions of weak inversion and verification. While our system does not perfectly invert the data, it uses weak inversion and verification to provide a number of guarantees about the lineage it generates. Weak inversion and verification can eliminate much of the irrelevant source data, thereby greatly reducing the amount of source data that the end user must visualize. We propose a design for the implementation of weak inversion and verification in an object-relational database management system.

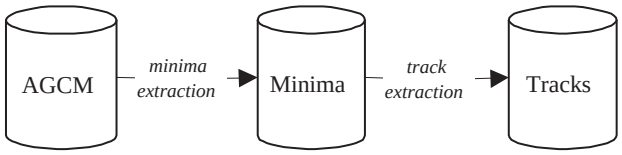
In the remainder of this section, we present a sample scenario that is used to illustrate our principles throughout the chapter. Section 3.2 defines our abstract model of weak inversion and verification. Section 3.3 describes how this model can be extended to a database environment and details the process by which expert users may register weak inversion and verification functions in an extensible database. Section 3.4 describes the inversion planner. Finally, Section 3.5 presents conclusions.

Example scenario

As a real-world application of our techniques, we consider a scenario for extracting cyclone tracks from atmospheric simulation data, based on [27]. In this subsection, we present the processing steps used in this scenario. Throughout the chapter, to illustrate portions of the model, we discuss the weak inversion and verification of functions in this example.

Validation of atmospheric simulations involves the comparison of model data to observational data. Cyclone tracks form one type of reference for such comparisons. The cyclone track extraction process begins with data generated by an Atmospheric General Circulation Model (AGCM), a simulation of the global climate. Two functions are applied to data output by the AGCM. The first function extracts local minima in sea level pressure (SLP), each of which may be the center of a cyclone. The second function assigns these minima to cyclone tracks. In the remainder of this subsection, we describe in detail the two processing steps (Figure 3.1a) and the schema of the data (Figure 3.1b).

Each AGCM tuple contains a time stamp corresponding to one time step in the simulation, as well as a multi-dimensional array. Each multi-dimensional array is divided into cells representing a spatial location, and each cell contains a variety of data about atmospheric conditions at each location, *e.g.*, SLP value, wind velocity, and wind direction. Figure 3.1c shows a visualization of one such array. Two dimensions are used to represent x and y location. The SLP value of each location is shown by the degree of gray shading



(a) Dataflow of cyclone track extraction.

AGCM

Time	Array

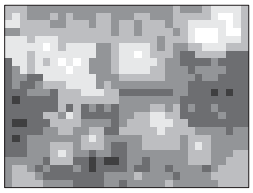
Minima

Time	Location	Wind Vel.	Wind Direct.

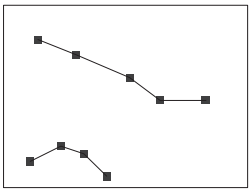
Tracks

Time	Location	Track

(b) Schema of cyclone track extraction.

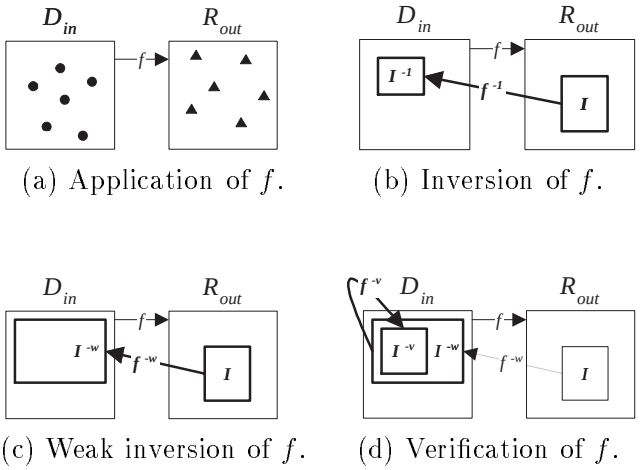


(c) SLP plot (single timestep).



(d) Cyclone track plot (multiple timesteps).

Figure 3.1: Cyclone track extraction.



(a) Application of f. (b) Inversion of f.

(c) Weak inversion of f. (d) Verification of f.

Figure 3.2: Weak inversion and verification.

used at that position in the plot.

A feature extraction algorithm is applied to this data to locate minima. It is a neighborhood algorithm (described in more detail in Section 3.3.1) that outputs the following data about each minimum (shown in the Minima table in Figure 3.1b): time, location, wind velocity, and wind direction.

Cyclone track identification is performed on the Minima table. The track identification algorithm attempts to assign each minimum to the trajectory of some cyclone. To be assigned to a given track a, the minimum at time t must (1) have a certain proximity to the minimum in track a at the previous timestep $t-1$ or (2) be consistent with the wind velocity and direction of such a minimum. Some minima are not in fact cyclone centers and are not assigned to any cyclone track. The output of the cyclone track identification phase is a list of the time, location, and track number of the minima that were successfully assigned to tracks.

At this point, the user views the results of the data processing. Figure 3.1d is a plot of the spatial movement of cyclone tracks as extracted by the process described above. The scientist may be puzzled by one of the cyclone tracks because it does not match their expectations or agree with the observational data. They would like to select the apparently anomalous track and see all inputs that contributed to it. □

Note that we use a □ symbol to mark the end of examples.

3.2 Abstract model

In this section, we define the model that will allow us to identify relevant inputs for the user. We begin with a function f that maps from a domain D to a range R. Now suppose the existence of sets $D_{in} \subseteq D$ and $R_{out} \subseteq R$ such that $R_{out} = \{f(d)|d \in D_{in}\}$. In Figure 3.2a, function f has been applied to each dot in set D_{in}, yielding the set of triangles in R_{out}. Note that D_{in} and R_{out} represent the elements in each set, not the enclosing rectangles themselves.

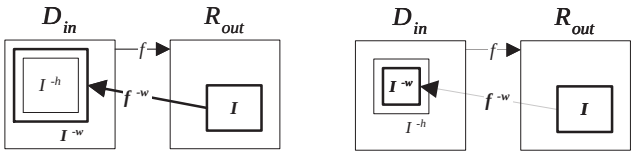
We are interested in inverting some subset of R_{out}. We call this subset I (the *image* of f). If f is invertible, there exists some function $f^{-1} : R \rightarrow D$ such that $f^{-1}(f(d)) = d$ for any $d \in D$. We say that $I_f^{-1} = \{f^{-1}(i)|i \in I\} = \{d \in D_{in}, f(d) \in I\}$ is the *inverse image* of I, *i.e.*, the relevant members of D_{in} that map onto I under f. To reduce notational clutter, we use I^{-1} to mean the inversion of an image I under some function f (the f is understood). Additionally, when necessary for this discussion, we use accents to distinguish particular functions, *e.g.*, f, $\hat{f}$, and $\tilde{f}$. When we define such functions, their corresponding images and inverse images are annotated $\hat{I}$, $\hat{I}^{-1}$, *etc.*

Figure 3.2b shows f^{-1} mapping from the image I to the inverse image I^{-1}. Again, note that the image and the inverse image both represent sets of elements, rather than the enclosing rectangles.

Not all functions are invertible; further, even if a function is invertible, its inverse may not be known. In these cases, we call the hypothetical inverse image I^{-h}. I^{-h} is the set of elements that actually map on to the image I; note that I^{-h} may not be derivable in practice.

To approximate I^{-h}, we begin with a weak inverse function $f^{-w} : R \rightarrow D$. f^{-w} is, strictly speaking, a relation rather than a function (members of the range may map to more than one member of the domain). To avoid confusion with database relations, we refer to f^{-w} as a function. Applying f^{-w} to I produces $D^{-w} = \{f^{-w}(i)|i \in I\}$, a set of values in the domain D.

Recall that our goal is to approximate I^{-h} as closely as possible. Note that by definition f^{-w} generates D^{-w} without examining D_{in}; therefore it may contain members of D that are not in D_{in}. However, any members that are not in D_{in} can not have contributed to the image, and therefore, do not belong in the inverse image. Hence, we can refine D^{-w} by intersecting it with D_{in}. The result is called I^{-w}, *i.e.*, $I^{-w} = D_{in} \cap D^{-w}$. I^{-w}



(a) Complete. (b) Pure.

Figure 3.3: Properties of weak and verified inverse images.

approximates I^{-h}.

Figure 3.2c shows f^{-w} mapping from I to I^{-w}. Recall that f^{-w} does not produce I^{-w} directly; rather, I^{-w} results from the intersection of D^{-w} (the output of f^{-w}) and D_{in}.

As an example, suppose f is a function that maps from integers to their squares and that I is the singleton set $\{9\}$ in R_{out}. f^{-w} can conclude that $D^{-w} = \{3, -3\}$. If D_{in} contains 3 but not -3, $I^{-w} = \{3\}$.

Because f^{-w} does not necessarily find the hypothetically perfect inverse image, it is not guaranteed that $\{f^{-w}(f(i))|i \in I^{-h}\} = I^{-h}$. Instead, we specify the relationship between I^{-w} (the set that is identified) and I^{-h} (the set that is actually of interest) with the following properties:

- **complete**: $I^{-w} \supseteq I^{-h}$. I^{-w} is complete if it contains all items of interest. In this case, there are no false negatives (items that should be included in I^{-w} that are not), *i.e.*, f^{-w} does not exclude any items of I^{-h} from I^{-w}. In Figure 3.3a, we see an image I which inverts to an inverse image I^{-h} and a function f^{-w} which yields I^{-w} when applied to the members of I. In this illustration, I^{-h} is contained by I^{-w}, so I^{-w} is complete with respect to I^{-h}.

- **pure**: $I^{-w} \subseteq I^{-h}$. I^{-w} is pure if it contains only items from I^{-h}. In this case, there are no false positives, *i.e.*, f^{-w} does not include any items in I^{-w} that are not in I^{-h}. In Figure 3.3b, I^{-w} is contained by I^{-h}, so I^{-w} is pure with respect to I^{-h}. Complete and pure are related to the traditional information retrieval metrics *recall* (the fraction of relevant documents that are retrieved) and *precision* (the fraction of documents that are retrieved which are relevant) [12]. Specifically, complete represents perfect recall and pure represents perfect precision.

Note that if D^{-w} is both complete and pure, $D^{-w} = I^{-h}$, *i.e.*, $\{f^{-w}(i)|i \in I\}$ is I^{-h}.

	Relationship to other sets	*Description*	*Definitions*
f	$f : D \rightarrow R$	function applied to D_{in} to yield R_{out}	
D_{in}	$D_{in} \subseteq D$	input set	
R_{out}	$R_{out} \subseteq R$	output set	$R_{out} = f(D_{in})$
I	$I \subseteq R_{out} \subseteq R$	portion of R_{out} being queried (the image)	
I^{-1}	$I^{-1} \subseteq D_{in} \subseteq D$	inverse image of I	
f^{-w}	$f^{-w} : R \rightarrow D$	weak inversion function of f	
D^{-w}	$D^{-w} \subseteq D$	filter on D_{in}	$D^{-w} = \{f^{-w}(i) \mid i \in I\}$
I^{-w}	$I^{-w} \subseteq D_{in} \subseteq D$	weak inverse image of I	$I^{-w} = D_{in} \cap D^{-w}$
f^{-v}	$f^{-v} : R \times D \rightarrow D$	verification function of f	
I^{-v}	$I^{-v} \subseteq I^{-w} \subseteq D_{in} \subseteq D$	the verified inverse image of I	$I^{-v} = f^{-v}(i, I^{-w})$

Table 3.1: Definitions.

In general, we use the term *closeness* to describe the relationship between the weak inverse image and I^{-1}. Note that not all sets with a given property are equivalent. For example, if two weak inverse images A and B are complete, with $|A| < |B|$, A is closer because it contains fewer irrelevant items and therefore more closely approximates the actual inverse image. (We use the notation $|A|$ to represent the cardinality of a given set A.) Similarly, if two non-equal weak inverse images are pure, the larger one is closer because it contains more relevant items (again, more closely approximating the actual inverse image).[2]

We next observe that not all functions have useful f^{-w}s. (The most general f^{-w} outputs D, saying that it is complete, since it must contain all items that contributed to the output items. We define a useful f^{-w} as one that is more restrictive, *i.e.*, one that outputs $D^{-w} \subset D$).) Therefore, we introduce a verification function, f^{-v}, which has access to the values in D_{in} (the input set). $f^{-v} : R \times D \rightarrow D$ takes I^{-w} and I as input and outputs a set $I^{-v} \subseteq I^{-w}$. In Figure 3.2d, the verification function f^{-v} is applied to I and I^{-w} to yield I^{-v}. I^{-v} can be described by the same properties as I^{-w}, *i.e.*, I^{-v} can be complete or pure. In addition, f^{-v} can require that the input set I^{-w} be complete or pure. We term such restrictions *application conditions*.

Our basic notation is summarized in Table 3.1.

[2] A possible general formulation of closeness involves a unit-cost similarity metric: $|A \cap I^{-h}| - |A \backslash I^{-h}| < |B \cap I^{-h}| - |B \backslash I^{-h}|$. (Consider that the number of correct items in A is $A \cap I^{-h}$ and the number of incorrect items is $A \backslash I^{-h}$. By subtracting the number of incorrect items from the number of correct items, we find an approximate measure of accuracy.) For simplicity, during the rest of the chapter we consider only cases in which we compare two pure or two complete sets.

3.3 Concrete model

This section applies the concepts of the abstract model to a specific environment, the Tioga database visualizer [41, 2]. As described previously, Tioga adopts the boxes-and-arrows programming paradigm popularized by AVS [49], Data Explorer [25], and Khoros [34]. Every box is a user-defined function and arrows represent the flow of data between these functions. Certain boxes are database browsers that visualize data and display it to the user. Tioga functions are written by expert users and registered in POSTGRES, an object-relational DBMS [43]. This registration mechanism can be extended so that the expert user can register weak inversion and verification functions.

In this section, we show how the set entities of the abstract model presented in Section 3.2 map onto database tuples and attributes. The fact that tuples have multiple attributes complicates the definition and application of the weak inversion and verification functions; we extend our model to address this issue. We then extend our model to allow the weak inversion and verification of portions of attributes, *e.g.*, an element in an array. Next, we extend the model from the inversion of single functions to the inversion of arbitrary dataflow graphs. Finally, we present the procedure the expert follows to register weak inversion and verification functions in POSTGRES.

3.3.1 Extending the abstract model to a database environment

Each function in Tioga takes as input some table D_{in} and yields as output some table R_{out}. These tables are a generalization of the input and output sets D_{in} and R_{out} of the abstract model; the sets in the abstract model can be considered to be single-column tables.

In this subsection, we discuss the inversion of attributes as well as the inversion of elements within complex attributes, *e.g.*, arrays.

Attributes

We begin with the image I that is to be weakly inverted. I in general consists of a set of tuples, which may be expressed as a selection on R_{out}. For clarity, we consider that I consists of a single tuple in this discussion. However, it is a straightforward generalization to consider images that contain multiple tuples or are the result of applying a selection predicate to R_{out}.

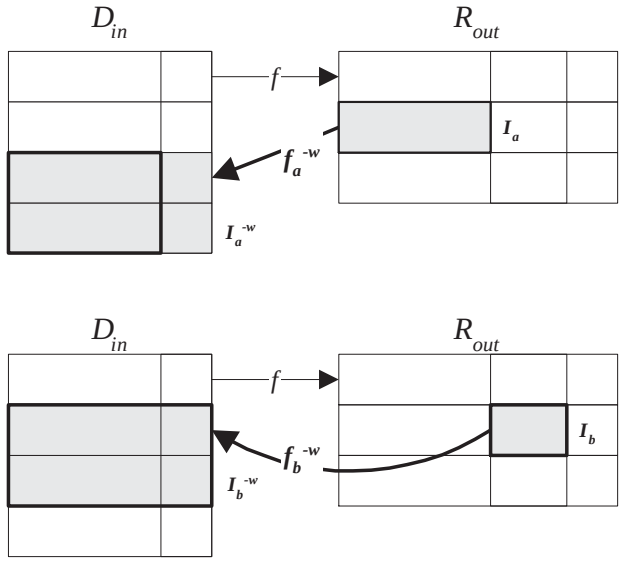
We have chosen to support weak inversion and verification at attribute-level granularity. This has two primary advantages over tuple-level granularity. First, the user is only required to provide weak inversion and verification functions for attributes in which they are interested or which they understand. Second, it allows more precise inversion.

The inverse image I^{-1} consists of the tuples in D_{in} that contain attributes that affected I. There is a separate weak inversion and verification process for each attribute within R_{out}, and each yields a single weak inverse image and verified image. Therefore, f^{-w} for a tuple is comprised of a number of functions $f_1^{-w}...f_n^{-w}$. Each f_k^{-w} weakly inverts a specific attribute k of R_{out}. We define I_k as the projection of I on k. $D_k^{-w} = \{f_k^{-w}(i) | i \in I_k\}$ describes a subset of the domain that might have contributed to I_k. Figure 3.4a shows the inversion of single attributes within a single tuple. In the top of the figure, attribute a is weakly inverted, resulting in I_a^{-w}, which is shaded gray. In the bottom of the figure, attribute b is weakly inverted, resulting in I_b^{-w}, which is shaded gray. Note that the weak inverses of these attributes may be different tuples in D_{in}; we will explore the significance of this later in the discussion.

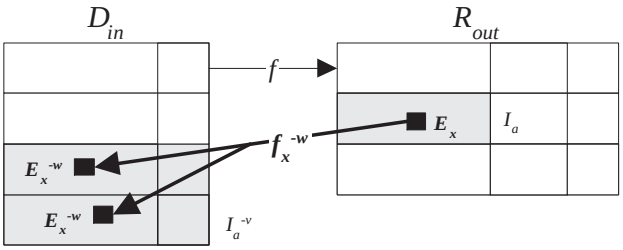
In our concrete model, each f_k^{-w} outputs a Boolean expression σ_k^{-w} containing selections on D_{in}. D_k^{-w} is the result of σ_k^{-w} applied to D. Similarly, I_k^{-w} is the result of σ_k^{-w} applied to D_{in}. I_k^{-w} may be complete or pure with respect to I_k. Additionally, I_k^{-w} may possess a user-defined property with respect to I_k.

As with f^{-w}, the verification function f^{-v} for a tuple is comprised of a number of functions $f_1^{-v}...f_n^{-v}$. Each f_k^{-v} takes two inputs and verifies a specific attribute k of R_{out}. The first input to an f_k^{-v} is I_k. The second input is the weak inverse image I_k^{-w}. The user may register requirements (application conditions) for the weak inverse image, *i.e.*, an f_k^{-v} may require that an input I_k^{-w} possess a specific property or properties with respect to I_k. The output of an f_k^{-v} is I_k^{-v}; I_k^{-v} can be described by the properties we have previously defined (pure, complete, or user-defined).

The expert user writes and registers each f_k^{-w} and f_k^{-v} individually. This user may register zero or more weak inversion or verification functions for each attribute. If the user does not provide a weak inversion or verification function for a given attribute, the system provides a trivial default function (discussed below). Multiple weak inversion or verification functions for a given attribute may be desirable when different weak inversion or verification functions for an attribute have different properties. For example, one can imagine registering one f_k^{-w} that is complete but not pure and another that is pure but not



(a) Weak inversion of attributes a and b in I.



(b) Weak inversion of an element E in attribute a.

Figure 3.4: Weak inversion of multiple levels.

complete.

Although the weak and verified inverse images consist of entire tuples in D_{in}, the user may want to know which attributes in D_{in} are related to some specific attribute in R_{out}. (we assume the existence of an interface through which the user specifies the attribute(s) in R_{out} of interest.) Such information can be inferred using the registration tables that are described in Section 3.3.2 and presented to the user.

Weak and verified inverse images of the same attribute may be combined. Further, the weak and verified inverse images of different attributes in the image can be combined to find the weak and verified inverse images of the entire image. The resulting weak and verified inverse images of the entire image can be described as having a given property or

properties (pure, complete, or user-defined). In Section 3.4 we discuss methods of combining weak and verified inverse images and the properties that result from these combinations. We assume that a small amount of bookkeeping is done during the combination of weak and verified inverse images to preserve information about which attributes in D_{in} are related to which attributes in R_{out}.

As an additional complication, attributes in an image may be the output of either *aggregate* or *scalar* functions. As an example of an aggregate function, if an attribute a in R_{out} is the maximum value of some attribute in D_{in}, f_a is aggregate. However, if an attribute a in R_{out} is derived from values in exactly one tuple in D_{in}, f_a is scalar. In Section 3.4, we discuss the different rules for combining the outputs of these two types of functions.

Example of weak inversion and verification of an attribute

We now return to our cyclone extraction scenario. Recall the schema described in Figure 3.1b. It is possible to write a trivial weak inversion function for the time field in Minima. Specifically, if I_{Time} consists of the single value t (recalling that for purposes of this discussion I is a singleton set), σ^{-w}_{Time} is "`select * from AGCM where AGCM.Time = t`". □

Complex Attributes and Elements

We have assumed above that I consists of simple attributes within tuples in R_{out}. However, POSTGRES supports a variety of complex attributes, *e.g.*, arrays, tuple types (in which an attribute may be broken down into a number of other attributes), and user-defined types (which can only be manipulated using the methods defined for the type). Observe that any attribute, whether simple or complex, can be weakly inverted and verified within the model described in Section 3.3.1.

We define an *element* to be a member contained in a complex attribute, *e.g.*, a cell within an array or an attribute in an instance of a tuple type (we consider tuple to be a type generator that takes a list of types as its arguments). The user may wish to query an element within a complex attribute in R_{out}. For example, a scientist may wish to invert a specific pixel within a satellite image. In this subsection, we extend the definition of weak inversion and verification functions to operate on elements within complex attributes.[3]

[3] We do not currently support these operations for subparts of arbitrary user-defined types that by nature

Therefore, the user may register an f_k^{-w} for any dimension k in an array. Similarly, the user may register an f_k^{-w} for any attribute k of a tuple type. In either case, appropriate f_k^{-v}s may also be registered. (Note that each element of a complex attribute may in turn be a complex attribute. Therefore, weak inversion and verification functions may exist for an element within an element.) We assume the existence of some interface through which the user can specify some element that they wish to invert.

For example, suppose there exist tables D_{in} and R_{out}, each containing an attribute of the array type, as shown in Figure 3.4b. Now imagine that E is an element within a specific array attribute a and that the goal is to invert dimension x in a. Weak inversion and verification functions are applied to identify I_a^{-v} in D_{in}; the relevant tuples in D_{in} are shaded gray. Then, f_x^{-w}s are applied to each member of I_a^{-v}.[4] The result is E_x^{-w}, which is shaded black in D_{in}.[5]

Put more informally, we wish to invert an element E contained by an array a. First we weakly invert and verify the higher level object a, yielding I_a^{-v}. This result is shown in gray. This result is the verified inverse of all the elements in a, and we are interested in a specific element, so now we must invert dimension x, the dimension of interest in element E. We do not need to apply this inversion to all members of D_{in}, since we have already eliminated many from consideration by our inversion of a. Therefore, we apply all f_x^{-w}s to the members of I_a^{-v}. The result is specific elements in the tuples in I_a^{-v}, shown in black. We can then apply verification functions to these elements (not pictured).

Weak inversion and verification functions are not required to return values at the same *level* as their arguments. We use the term level to describe the degree of nesting of an attribute or element within a tuple. The top (first) level consists of the attributes in a given table, the second level consists of the attributes or dimensions contained by the top level, *etc.* Because weak inversion and verification functions do not necessarily return values at the same level as their arguments, the weak inversion of an element might yield a simple attribute. Similarly, the weak inversion of an attribute might yield an element.

Example of weak inversion and verification in the presence of complex attributes

do not have accessors known to the database. For example, if the user has registered a black-box circle type with a radius method, we do not assume the radius is visible to our system. By contrast, if the circle type is implemented as a tuple type, the radius and center-point are visible as attributes of the tuple type.

[4] The specific process is discussed in more detail in Section 3.4.2 below.

[5] We use I and E in this example to distinguish between the attribute and the element within the attribute. However, in general, we still consider the output of an f_k^{-w} (f_k^{-v}) to be I_k^{-w} (I_k^{-v}).

Information	*Type*
name of f_k^{-w}	string
name of f	string
nature of f_k	aggregate or scalar
image type	type of attribute or dimension k within R_{out} being weakly inverted
inverse image types	types of attributes or dimensions within D_{in} that app ear in I_k^{-w}
properties of output I_k^{-w}	complete and/or pure and/or user-defined

Table 3.2: Information to register for weak inversion functions.

Information	*Type*
name of f_k^{-v}	string
name of f	string
nature of f_k	aggregate or scalar
image type	type of attribute or dimension k within R_{out} being verified
inverse image types	types of attributes or dimensions within D_{in} that appear in I_k^{-v}
properties of output I_k^{-v}	complete and/or pure and/or user-defined
application conditions for I_k^{-w}	complete and/or pure and/or user-defined

Table 3.3: Information to register for verification functions.

Suppose the goal is to identify the inverse image of a specific minimum I in Minima in our cyclone track extraction scenario. We now consider a method of inverting Minima.Location. Recall that the function that extracts minima from AGCM.Array is a neighborhood algorithm. Minima at a location (x, y) are identified if they meet one of two criteria:

1. All immediate neighbors of (x, y) have a higher SLP than SLP(x, y).
2. The average SLP of the 5x5 neighborhood centered at (x, y) (but exclusive of (x, y)) is higher than SLP(x, y).

Therefore, the classification of I as a minimum may have resulted either from values of its immediate neighbors or from values of the 5x5 neighborhood surrounding it. Since there is no way to distinguish between these two cases without examining the values in the AGCM table, the weak inversion of $I_{Location}$ returns all cells in the 5x5 neighborhood centered at $I_{Location}$. This weak inverse image is complete but not pure. The verification function has an application condition that specifies that the weak inverse image must be complete (it must be able to examine the entire 5x5 neighborhood).

The verification function examines the contents of the 5x5 neighborhood and determines which criteria applied. If the first applied, the verified inverse image consists of the immediate neighborhood. If the second applied, the verified inverse image consists of the 5x5 neighborhood. In both cases, the verified inverse image is both complete and pure with respect to $I_{Location}$. □

Dataflow graphs

Thus far, the model has not addressed multiple inputs or outputs to functions. However, a general dataflow graph is a DAG. We address this issue by restructuring the dataflow graph into groups of functions with one input and one output. We invert these groups separately. We then combine the results of the inversions.

More specifically, we define a *chain* in a dataflow graph to be a linear sequence of functions from an input to an output. Each function in the chain is called a *step*. An arbitrary dataflow graph may be broken down into a number of such chains. Each chain is inverted separately (the specific process for inverting a chain is discussed in Section 3.4). The results of the inversions of each chain are unioned to find the inversion of the entire dataflow graph.

3.3.2 Registration procedure

The expert user must register several pieces of information about weak inversion and verification functions. This information is used by the inversion planner described in Section 3.4 to infer which functions should be used for weak inversion and verification.

The user begins by identifying the name of the function that will perform the weak inversion or verification. The user next identifies the function f that is being weakly inverted and verified. The user also specifies whether the attribute being inverted results from an aggregate or scalar function, *i.e.*, f_k is described as aggregate or scalar.

The user must also register information that allows the inversion planner to infer which weak inversion and verification functions apply to a given attribute. Therefore, for each inversion function, the user specifies the types of the relevant attributes (or dimensions) in the image and in the inverse image. The inversion planner searches for attributes (or dimensions) in D_{in} and R_{out} that match these specified types.[6]

[6]This typing system may lead to ambiguities, *e.g.*, if two attributes of the same type appear in D_{in} and

Finally, the user enters information about the properties of the sets output by the weak inversion and verification functions. Note that if a property is specified for an output set, it is guaranteed to be true. However, if it is not specified, it might or might not pertain. The information the expert user enters to register an f_k^{-w} or an f_k^{-v} is summarized in Table 3.2 and Table 3.3.

As mentioned in Section 3.3.1, in addition to the f_k^{-w}s or f_k^{-v}s registered by the user, every attribute or element resulting from every function has a default f_k^{-w} and a default f_k^{-v}. The default f_k^{-w} outputs a filter σ_k^{-w} consisting of no selections, *i.e.*, $I_k^{-w} = D_{in}$. The default I_k^{-w} is therefore guaranteed to be complete, but it is not guaranteed to be pure or to possess any user-defined properties. The default f_k^{-v} outputs I_k^{-w}. Therefore, the default I_k^{-v} has precisely the same properties as the I_k^{-w} it takes as input. Note that if both defaults are used, $I_k^{-w} = I_k^{-v} = D_{in}$.

User-defined properties are registered in a separate mechanism in which the user specifies the name of the property and the combination rules that apply to it (either complete or pure rules as described in Section 3.4.1).

3.4 Inversion planner

The inversion planner is responsible for devising a plan to weakly invert and verify the image selected by the user. The result of the execution of this plan must match the user's specification of properties as closely as possible (*e.g.*, the user may specify that they wish the verified inverse image to be complete or pure). The plan specifies which weak inversion and verification functions will be applied to which tables in what order.

In this section, we discuss how properties of weak and verified inverse images can be preserved during the combination of weak inversion and verification functions. We then present the algorithm the inversion planner follows to invert a chain.

We make several simplifying assumptions. In Chapter 9 we consider further optimizations which may be made if these assumptions are relaxed.

- We assume all tables (including intermediate results) are materialized.

the registered information for f does not permit us to infer which function produces which attribute. In an alternative design, the user might specify the precise names of the attributes in the image and inverse image. However, such a design limits the degree to which weak inversion and verification functions may be easily reused, forcing the user to explicitly register weak inversion and verification functions for every attribute that is to be inverted.

- We assume there is a per-tuple cost of applying an f_k^{-w} or an f_k^{-v} (as opposed to, *e.g.*, a fixed or per-byte cost).[7]
- We assume that the planner is trying to find the closest possible verified inverse image.
- We assume that the desired properties as specified by the user are the same for all verified inverse images in a chain.

3.4.1 Preservation of properties

We have already discussed several properties of sets (complete, pure, user-defined). Some combinations of sets preserve such properties and some do not. We begin our discussion of the preservation of properties by detailing inversion of a single function (simple attributes and complex attributes). Next, we discuss inversion of multiple functions.

Preservation of properties during the inversion of simple attributes

This subsection covers two primary types of combinations. First, for a given dimension k, I_k^{-w}s or I_k^{-v}s may be combined to improve the closeness of the inversion of k.[8] Second, to this point, we have only considered weak inversion and verification of attributes in the image. However, after all individual attributes have been fully inverted, a higher-level combination may also be performed. Specifically, the results of the inversion of multiple attributes may be combined to assemble the inverse image of an entire tuple. (Similarly, the results of the inversion of multiple dimensions within a complex attribute may be combined to assemble the inverse image of the complex attribute.) In the latter part of this subsection, we describe how such combinations may be advantageous.

We begin by considering the case in which R_{out} contains exactly one attribute, y. Recall that multiple weak inversion functions f_y^{-w} may be registered for a single attribute (*i.e.*, the inversion planner may have the choice of several weak inversion functions). We have already observed that it is desirable to have different weak inversion functions that have different properties. However, it is also desirable to have multiple weak inversion functions with the same property to increase the closeness of the weak inversion. There

[7] Note that these are not the only alternatives, *e.g.*, a function that has cost quadratic in its input does not have a fixed per-tuple cost.

[8] In general, we use the term dimension to refer to either an attribute that is being weakly inverted and verified or a dimension within an array that is being weakly inverted and verified.

are two interesting cases. First, suppose we have two weak inversion functions, each of which returns a pure set (call them A and B).[9] If $A \neq B$, the union of A and B yields a strictly larger pure set (the larger a pure set, the more accurate it is). Second, if the weak inversion functions yield complete sets, the intersection of A and B yields a strictly smaller complete set (the smaller a complete set, the more accurate it is). Observe that these rules of combination apply whether f_y is scalar or aggregate. Figure 3.5a shows the hypothetical inverse of a given image I. Figure 3.5b shows two different weak inversions and their results A and B. Note that because A and B are complete, their intersection contains the hypothetical inverse (shown in black).

Now consider the combination of the weak inverse images of multiple dimensions. First, we discuss the case in which the f_ks are scalar. If a tuple in I^{-1} is relevant to a tuple in I, it must be relevant to each attribute of I, *i.e.*, for all k, it must be a member of I_k^{-1}. Therefore, in general, if multiple attributes are scalar, the weak inverse images of these should be intersected to find the weak inverse image of the entire tuple (or if multiple dimensions are contained in a complex attribute, the intersection of the weak inverse images of each dimension finds the weak inverse image of that complex attribute).

As a concrete example, consider the case in which R_{out} contains precisely two attributes x and y. Suppose the image I consists of exactly one tuple that we wish to invert.

Now suppose that I_x^{-w} is complete in relation to x. (This does not guarantee that it is complete in relation to y.) Also assume that I_y^{-w} is complete in relation to y. (Again, this does not guarantee that it is complete in relation to x). Observe that any tuple that is relevant to I must be relevant to both the x and y values in I; since both I_x^{-w} and I_y^{-w} are complete, all relevant tuples must appear in both sets. Therefore, $I_x^{-w} \cap I_y^{-w}$ contains all possible tuples that may be relevant to I, *i.e.*, $I_x^{-w} \cap I_y^{-w}$ is complete in terms of I. However, some of the tuples in the intersection may not be relevant. Figure 3.5c shows two functions f_x^{-w} and f_y^{-w}. The weak inverses they create, I_x^{-w} and I_y^{-w}, are complete. Therefore, the intersection of these inverses contains the hypothetical inverse (shown in black).

Pure sets follow the same combination rules, although the logic is slightly different. Suppose that I_x^{-w} is pure in relation to x and I_y^{-w} is pure in relation to y. Any tuple that appears in both sets is guaranteed to be relevant to both x and y. The intersection therefore

[9] Examples in this section refer to the combination of I_k^{-w}s. The same rules apply to D_k^{-w}s and I_k^{-v}s.

contains only relevant tuples, although it may not contain all relevant tuples.

Next, we discuss the case in which f_ks are aggregate. Suppose R_{out} contains two attributes x and y. In this situation, there is no guarantee that a single tuple in D_{in} must be relevant to I. For example, suppose x is the maximum of an attribute a in D_{in} and y is the maximum of an attribute b in D_{in}. The weak inverse images of x and y may be disjoint; however, both are relevant to I. Therefore, all weak inverse images associated with aggregate f_ks (whether complete or pure) should be unioned.

Finally, consider the case in which some f_ks are scalar and some are aggregate. All weak inverse images associated with scalar f_ks should be intersected as specified. Then, all weak inverse images associated with aggregate f_ks should be unioned. As the last step, both of the resulting sets should be unioned. Observe that whether the system is combining sets within one attribute or combining sets for multiple attributes, there is no case in which it is desirable to combine a pure set with a complete set.

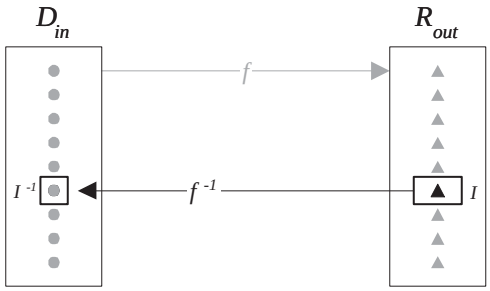
In some cases, the system may invert a subset of the attributes in an image (either because the user has specified that only those attributes are of interest or because interesting weak inversion and verification functions are not available for all attributes). Notationally, if a set has certain properties with respect to multiple attributes $1...k$ in the image, we say that it has those properties with respect to $I_{1...k}$.

Example of weak inversion and verification of multiple scalar attributes

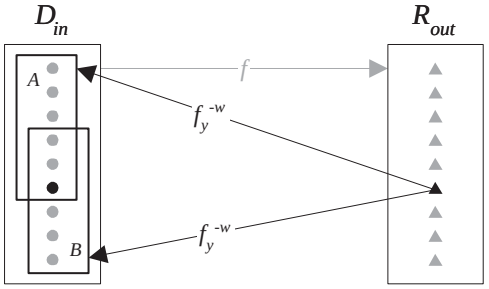
Returning to our cyclone track extraction example, consider the weak inversion and verification of an image in the Tracks table. Suppose that weak inversion and verification functions have been registered for the attributes Time and Location. Observe that f_{Time} and $f_{Location}$ are scalar. Also note that each of these attributes has trivial weak inversion and verification functions that yield sets that are complete and pure with respect to the individual attributes Time and Location (the values in Minima are identical to those in Tracks). Therefore, since the forward functions are scalar, the system intersects I^{-v}_{Time} and $I^{-v}_{Location}$. The result is complete and pure with respect to $I_{Time,\ Location}$. □

Preservation of properties during the inversion of complex attributes

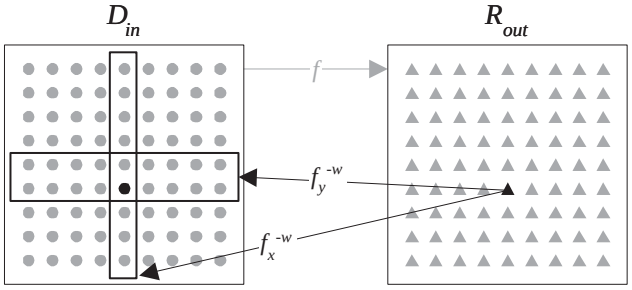
Recall from Section 3.3.1 that the weak and verified inverse images of an image can exist at multiple levels. In this subsection, we consider the properties of these different levels in the weak and verified inverse images. First, we discuss the properties of a single



(a) Inversion of an attribute.



(b) Combination of two complete I_y^{-w}s.



(c) Combination of a complete I_x^{-w} and a complete I_y^{-w}.

Figure 3.5: Combination of weak inverse images.

level in the weak and verified inverse images. Second, we discuss the properties of multiple levels in the weak and verified inverse images.

In Section 3.4.2, we present the specific process by which the weak and verified inverse images are identified. For now, it is sufficient to understand that each level in the weak and verified inverse images is calculated by a set of independent weak inversion and verification functions. The weak inversion and verification functions for a single level in the inverse image may invert different levels in the image. For example, one function may weakly invert an attribute in the image to tuples in the weak inverse image; another function may weakly invert an element in the image to tuples in the weak inverse image as well. In general, the resulting weak or verified inverses are combined according to the combination rules described in Section 3.4.1 above.

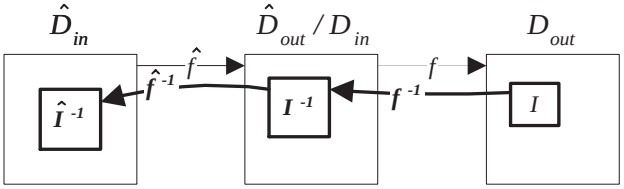
Since each level in the inverse image is computed separately, each level in the inverse image can have different properties with respect to the various levels in the image. However, the properties of a level in the inverse image affect the properties of all levels below it. Specifically, the weak or verified inverse image of a lower level can only have a given property with respect to a level in the image if all higher levels in the inverse image have that same property. This implies that levels in the inverse image are computed in a top-down manner, which in turn implies that each level in the inverse image is a subset of the higher levels.

For example, suppose that in Figure 3.4b, f_a^{-v} identifies tuples containing satellite images that contributed to an aggregate satellite image I_a and that the output of f_a^{-v} is complete with respect to I_a. Now suppose that f_x^{-w} identifies the region of each satellite image that contributed to E_x and that its output E_x^{-w} is pure with respect to E_x. Observe that f_x^{-w} might be applied to a member of I_a^{-v} that is not relevant to I_a; the resulting E_x^{-w} is therefore not relevant to I_a. Consequently, E_x^{-w} is not pure with respect to I_a (it would be pure only if I_a^{-v} were pure).

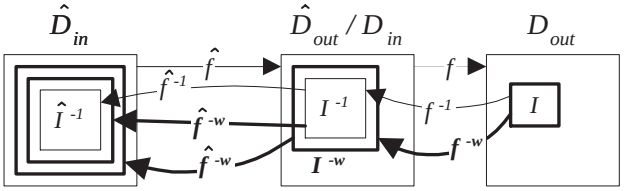
Example of preservation of properties during inversion of complex attributes

Consider the inversion of an image I in the Minimum table of the cyclone track extraction example. This inversion is performed in two steps.

First, the system weakly inverts and verifies the top level of AGCM. We assume weak inversion and verification functions are available for Minima.Time and Minima.Location. The system applies weak inversion and verification functions to identify



(a) Inversion of two functions.



(b) Weak inversion of two functions.

Figure 3.6: Weak inversion of a chain.

$\hat{I}^{-v}_{Time}$ (both $\hat{D}^{-w}_{Time}$ and $\hat{f}^{-v}_{Time}$ restrict AGCM.Time). $\hat{I}^{-v}_{Time}$ is complete and pure. Applying the weak inversion function to $\hat{I}_{Location}$ yields a complete set $\hat{I}^{-w}_{Location}$ that consists of all tuples in AGCM. Applying the verification function to $\hat{I}^{-w}_{Location}$ yields a complete and pure set $\hat{I}^{-v}_{Location}$ that also consists of all tuples in AGCM (the verification function is able to verify that the array in each tuple contains $\hat{I}_{Location}$). At this point the system uses the combination rules: it intersects $\hat{I}^{-v}_{Time}$ and $\hat{I}^{-v}_{Location}$ to find a verified inverse image $\hat{I}^{-v}$ that is complete and pure with respect to $\hat{I}_{Time,\ Location}$.

Next, the system weakly inverts and verifies the second level of AGCM. It uses the weak inversion function described in Section 3.3.1 to generate a filter $\tilde{\sigma}^{-w}_{Location}$. It applies this filter to every member of $\hat{I}^{-v}$. The result $\tilde{E}^{-w}_{Location}$ is complete with respect to $\hat{I}_{Location}$. It then applies the verification function $\tilde{f}^{-v}_{Location}$ to $\tilde{E}^{-w}_{Location}$, which yields a complete and pure set $\tilde{E}^{-v}_{Location}$. Since the weak inversion and verification of both levels is complete and pure, the result of the second level inversion is complete and pure with respect to $\hat{I}_{Time,\ Location}$. □

Preservation of properties during the inversion of multiple functions

In this section, we show how our abstract model generalizes to chains. Observe that the properties of weak and verified inverse images are transitive.

For example, consider a chain with two functions $\hat{f}$ and f in which the output of $\hat{f}$ is input to f. Suppose that an expert user has registered functions that provide weak inversion and verification of each of these functions.

Now suppose an end user wishes to find the inverse image of an image in R_{out} (see Figure 3.6a). The user would like to identify the relevant inputs in both D_{in} and $\hat{D}_{in}$. Ideally, the system would use f^{-1} to invert the image and identify I^{-1} in D_{in}. Then, it would treat I^{-1} in D_{in} as an image in $\hat{R}_{out}$ and find its inverse image $\hat{I}^{-1}$ in $\hat{D}_{in}$.

The system can apply its weak inversion functions in this situation as follows (we assume these weak inversion functions are complete). It begins by finding a weak inverse image in D_{in}.[10] However, the user wishes to see the relevant inputs from $\hat{D}_{in}$ as well. This is accomplished by using I^{-w} as an image in $\hat{R}_{out}$. Recall that the weak inverse image can differ from the actual inverse image. Chaining weak inversion functions together amplifies this inaccuracy. Figure 3.6b shows the image from Figure 3.6a extended to show the results of applying weak inversion functions. Observe that applying $\hat{f}^{-w}$ to I^{-w} yields a larger (and more inaccurate) set than applying $\hat{f}^{-w}$ to I^{-1}.

Despite this loss of accuracy, the system can still make certain guarantees about the relationship of weak inverse images to inverse images in these situations. The key observation is that completeness and purity are both transitive properties. Specifically, if both f^{-w} and $\hat{f}^{-w}$ are complete, so are their outputs. In Figure 3.6b, therefore, both $\hat{f}^{-w}$ applied to I^{-1} and $\hat{f}^{-w}$ applied to I^{-w} are complete, though neither is pure.

Example of preservation of properties during inversion of multiple functions

Consider the dataflow diagram in Figure 3.1a. In Section 3.4.1, we showed that the weak inversion and verification of certain attributes in Tracks yields a complete and pure verified inverse image in Minima. Similarly, in Section 3.4.1 we showed that the weak inversion and verification of certain attributes in Minima yields a complete and pure verified inverse image in AGCM. Therefore, the weak inversion and verification of an image in Tracks yields complete and pure verified inverse images in both Minima and AGCM. □

[10] It would also be possible to find a filter or a verified inverse image and use it as the image. Each of these cases has different performance implications, as discussed in Section 9.2.1. For simplicity, we assume that we are using the weak inverse image for the remainder of this section.

3.4.2 Inversion planner algorithm

In this subsection, we first discuss ordering constraints for weak inversion and verification. We then present an algorithm that follows these constraints as well as those presented in Section 3.4.1.

We have discussed several stages of weak inversion and verification of a chain. If weakly inverting or verifying some part of the chain impacts the weak inversion or verification of some other part, we say the former part *affects* the latter. For example, consider Figure 3.6b. If the weak inversion and verification of f were more accurate, then I^{-w} would be more accurate. The improved I^{-w} could be used as an image input to $\hat{f}^{-w}$ resulting in a more accurate $\hat{I}^{-w}$. Therefore, we say that the weak inversion and verification of f affects the weak inversion and verification of $\hat{f}$.

There are two critical observations about which parts of the chain can affect others:

1. The inversion of a step can affect the inversion of any step to its left (although it can not affect the inversion of steps to its right).

2. Within a step, one inversion can affect another inversion that yields a lower level in the inverse image (since the lower levels in the inverse image are a subset of the higher levels). Conversely, one inversion can not affect another inversion that yields a higher level in the weak or verified inverse image.

These observations suggest a natural ordering of the inversion process:

1. Steps should be weakly inverted and verified proceeding from right to left.

2. Within a step, each level in the inverse image should be computed (weakly inverted, verified, and combined) before the levels below it are computed.

Combining the constraints of Section 3.4.1 (preservation of properties) and Section 3.4.2 (ordering) we arrive at the algorithm presented in Figure 3.7.

Example application of algorithm

We have discussed all the weak inversions and verifications necessary to trace the relevant inputs of the cyclone track extraction scenario presented in Section 3.1. According to the algorithm for the inversion planner, the complete weak inversion and verification of an image I in Tracks would consist of the following steps:

- Weakly invert and verify I, yielding a complete and pure verified inverse image I^{-v} in Minima (as described in Sections 3.3.1 and 3.4.1). Specifically, the system will:

  1. Weakly invert and verify the attributes Tracks.Time and Tracks.Location.
  2. Intersect I^{-v}_{Time} and $I^{-v}_{Location}$ (see Figure 3.8a[11]).

- Weakly invert and verify I^{-v} yielding a complete and verified inverse image $\hat{I}^{-v}$ in AGCM (as described in Sections 3.3.1 and 3.4.1). Specifically, the system will:

  3. Weakly invert and verify the attributes Minima.Time and Minima.Location, finding the top level of the verified inverse image.
  4. Intersect $\hat{I}^{-v}_{Time}$ and $\hat{I}^{-v}_{Location}$. The result is $\hat{I}^{-v}$ in AGCM. $\hat{I}^{-v}$ is the set of the members of AGCM that are relevant to I^{-v} (see Figure 3.8b).
  5. Weakly invert the attribute Minima.Location, finding the second level of the weak inverse image. The result is $\tilde{E}^{-w}_{Location}$ which is complete but not pure.
  6. Verify $\tilde{E}^{-w}_{Location}$ yielding $\tilde{E}^{-v}_{Location}$ (the result of applying the verification function is shown in Figure 3.8c). $\tilde{E}^{-v}_{Location}$ is both complete and pure (as discussed in Section 3.4.1). □

3.5 Conclusions

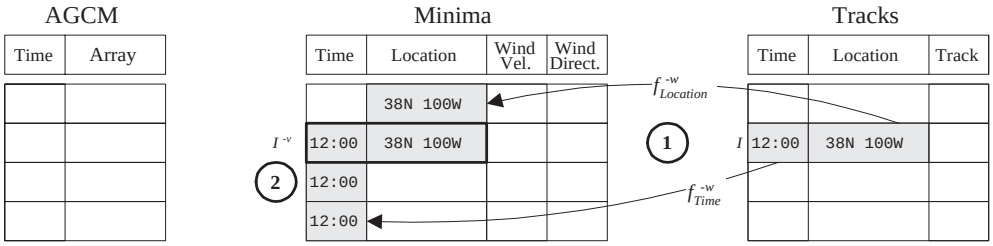
We have proposed a method to support fine-grained data lineage. Rather than relying on metadata, our approach computes lineage on demand using a limited amount of information about the processing steps. This approach incorporates weak inversion and verification. While the system does not perfectly invert the data, it provides a number of guarantees about the lineage it generates on the fly.

We have proposed a design for the implementation of weak inversion and verification in an object-relational DBMS. While we have not had the opportunity to implement this design, the functionality it describes has a number of interesting applications. For example, in the context of this dissertation, when users perform data lineage queries, weak inversion/verification can eliminate irrelevant source data, thereby reducing clutter in the display.

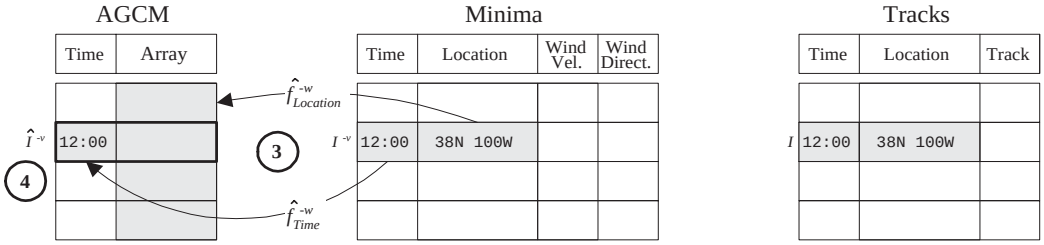
[11] For clarity, Figure 3.8 illustrates only attributes that are in the weak and verified inverse images rather than the entire tuple. As mentioned in Section 3.3.1, a small amount of bookkeeping is done to facilitate such a presentation to the user.

```
for each step num_steps to 1
    for each left_level 1 to num_left_levels
        for each right_level
            for each dimension k in 1 to num_attributes
                find all f_k^-v s that have the desired property
                find all f_k^-w s with matching application conditions
                remove the f_k^-v s that can't be satisfied
            end;
        end;
    end;
    for each left_level 1 to num_left_levels
        for each right_level k in 1 to num_attributes
            for each dimension
                apply all f_k^-w s
                apply the filters to get the I_k^-w s
                combine the I_k^-w s for the dimension
            for each right_level
                apply all f_k^-v s
            combine the I_k^-v s within each dimension
        end;
        combine the I_k^-v s across dimensions
    end;
end;
```

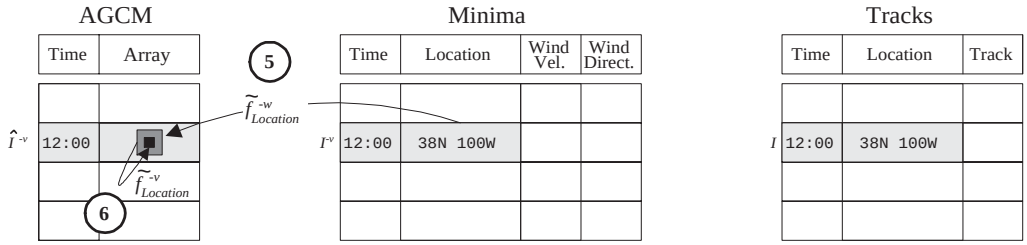
Figure 3.7: Algorithm for inverting a chain.



(a) Identifying the verified inverse image of the track extraction.



(b) Identifying the top-level verified inverse image of the minima extraction.



(c) Identifying the second-level verified inverse image of the minima extraction.

Figure 3.8: Inversion of cyclone track extraction.

Part II

Constant information density in zoomable interfaces

Chapter 4

Pilot study

4.1 Introduction

In Part I, we addressed data lineage, the first of two main problems Earth scientists identified in existing visualization systems. In the work described in this part of the dissertation, we address the second problem: achieving appropriate information density in visualizations. Our solution uses a cartographic principle to guide the construction of visualizations. The *Principle of Constant Information Density*, drawn from the cartographic literature [16, 45], states that the number of objects per display unit should be constant. A more general formulation posits that the amount of information (as defined by metrics discussed below) should remain constant as the user pans and zooms.

To investigate the usefulness of this principle in database visualization, we performed a small, informal study of user navigation behavior in applications with and without constant information density. To our knowledge there have been no similar studies. Therefore, our purpose in this study was to gain intuition about navigation patterns and to identify interesting directions for future research. Note that our intent was not to examine density metrics and appropriate values for these metrics, but rather to examine user response to density variance according to a given metric.

The rest of this chapter is organized as follows. We first describe our method, apparatus, participants, procedure, and task. We then present the results of the study and discuss them. We next describe user response and limitations of the study, and finally, we conclude.

4.2 Method

Visualizations of data for Fortune 500 and Global 500 companies were used in the study. The visualizations displayed scatterplots with % profit and number of employees on the x and y axes, respectively.

We wished to present participants with displays that were uniform in the x and y dimensions, because non-uniformity could have biased their behavior, *e.g.*, they could have shown more interest in areas with more data. Therefore, we sampled the data such that it was distributed nearly uniformly in the x and y dimensions. (Certain areas of the scatterplot had no data whatsoever, so our spatial sampling did not result in a perfectly uniform distribution.)

Using this spatial sampling, we produced a a total of four data sets, containing 436, 350, 100, and 48 members. Each smaller data set was a strict subset of all larger data sets.

For each of the data sets, we created a layer that contained exactly one object (a circle) for each member of its data set. The x axis represents the percent profit (profit dollars divided by revenue dollars) of the company during a given year. The y axis represents the number of employees of the company. The color of the circle represents the profit of the company in dollars.

We used our extensions to DataSplash (described below in Chapter 5) to create four visualizations that contained these layers. Each visualization had a top layer that was visible at higher elevations and a bottom layer that was visible at lower elevations. The top layer in each visualization contained either 48 objects (low density) or 350 objects (high density). The bottom layer in each visualization contained either 100 objects (low density) or 436 objects (high density). All possible combinations of top and bottom layers with low or high density yielded four versions of the visualization. Note that while this design varied the number of objects visible in different layers, it did not vary the type of information visible in different layers, *e.g.*, no text labels appeared as the user zoomed. This control was included to allow us to focus on participants' responses to different numbers of objects.

The active elevation ranges of the top and bottom layers were the same in all visualizations. Therefore, the transition (the elevation at which the top layer became invisible and the bottom layer became visible) was the same in all visualizations. At the highest elevation, the entire native data space was visible in the display. At any other elevation,

Layer		*Low Low (LL)*		*Low High (LH)*		*High Low (HL)*		*High High (HH)*	
Top	Vis. at top	48	48	48	48	350	350	350	350
	Vis. at transition		31		31		224		224
Bottom	Vis. at transition	100	64	436	279	100	64	436	279
	Vis. at bottom		36		157		36		157

Table 4.1: Number of objects visible at different elevations in visualizations.

participants who wished to examine the entire native data space at that elevation could do so by panning.

Table 4.1 presents details about the number of objects visible in the display in each layer in each visualization. Each group shows the base number of objects in the layer as well as the number of objects visible at the highest and lowest elevations of the layer. Observe that significant discontinuities in density occur at the transition point in all visualizations except High/High. (Hereafter, we will refer to each visualization by its two-letter abbreviation.)

4.3 Apparatus

We wished to reach as wide an audience as possible. Therefore, we developed a simple Java applet that displays visualizations generated in DataSplash. For purposes of this experiment, the applet was embedded in a World-Wide Web page so that anyone with World-Wide Web access could easily participate.

The applet supports only simple navigation tasks. Specifically, it only supports pan and zoom functionality. To pan, the user clicks in the display; the point on which they click becomes the center of the visualization. To zoom, the user presses one of two buttons, "Zoom In" and "Zoom Out." In this experiment, these buttons took users to one of nine possible elevations. Dynamic labels on the x and y axes indicated the current range of values of visible objects. In the study, the applet displayed one of the versions of the visualization described above. A screenshot of the applet appears in Figure 4.1. In this figure, the user is at the highest possible elevation and the canvas shows a top layer of high density.

The applet recorded data about each of the panning and zooming commands made by the participants. Data collected included the command type, the x, y, and z location at which the command was made, and the time of the command.

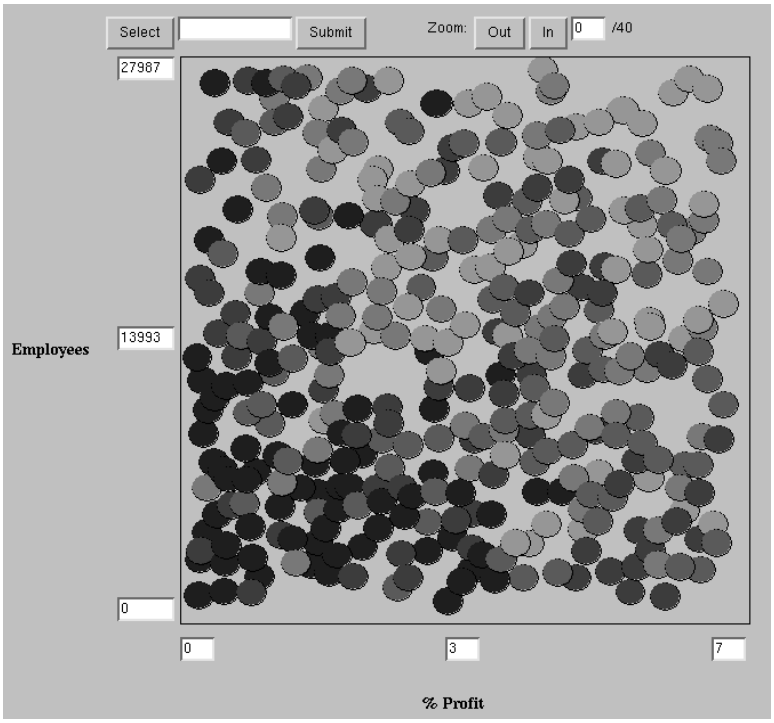


Figure 4.1: The applet used in the pilot study.

4.4 Participants

Seventy-nine participants were recruited through technical mailing lists and news groups. Participants' ages, levels of education, and self-reported levels of exposure to graphical representations of data varied widely.

4.5 Procedure

The recruitment instructions stated that we were conducting a study of how people interpret visualizations of data. It asked volunteers to visit a site on the World-Wide Web. Visitors to this site viewed a "Welcome" page. As an incentive, we stated that respondents would be entered in a drawing for a free T-shirt. We also stated that respondents who got the correct answer would be entered in a drawing for an additional T-shirt. If individuals decided to participate in the study, they proceeded to the next page.

At this point, participants were provided written instructions about the functionality of the applet. The spatial metaphor and panning and zooming operations were described

in detail. The instructions explicitly stated that when participants zoomed in or out, the contents of the display might change. Specifically, they were told that objects might appear, disappear, or change form when they used the zoom operation.

Participants were told the visualization represented data about selected companies from Fortune Magazine's Fortune 500 and Global 500 lists. The meaning of the axes and color assignments were described.

Each participant viewed exactly one version of the visualization. The version presented was chosen randomly. Instructions were the same for all versions. All users began the experiment centered in the x,y coordinates of the data space and positioned at the highest possible elevation.

4.6 Task

Participants were asked to locate the company they thought had the highest revenue growth (revenue growth is expressed as percent change from the previous year). Because this information was not explicitly present, they were expected to infer it from the other data displayed in the visualizations; the inferential nature of the task is discussed further below.

When participants identified a company they thought might have the highest revenue growth, they were to press the "Select" button. When they did so, the x,y values of that company appeared in a text box to the right of the "Select" button. Participants were allowed to perform the select operation as many times as they wished. When they were finished, they pressed the "Submit" button and proceeded to an exit survey in which they answered brief questions about the visualization and provided limited demographic information.

4.7 Results and discussion

Out of the seventy-nine participants, fifty-seven zoomed to the transition point at least once. Because users who did not zoom to the transition point could not have been affected by the variation in density between the two layers, we present the data from those fifty-seven traces only. These traces are relatively evenly distributed among the four

Version	*LL*	*LH*	*HL*	*HH*
% Participants	80%	89%	71%	80%

Table 4.2: Percentage of participants returning to top layer.

Version	*LL*	*LH*	*HL*	*HH*
Top	20.5	17	11.5	11
Bottom	11.5	3	5	1

Table 4.3: Median number of pan operations.

versions.[1]

To assess the effect of constant information density on zooming behavior, we measured the percentage of participants who zoomed back to the top layer after visiting the bottom layer. Results appear in Table 4.2.

Zooming appears to be influenced by constant information density. Observe from the data in Table 4.2 that users of the LL and HH visualizations were equally likely to return to the top layer. This suggests that the constant information density increased the likelihood that users would move between layers. By contrast, users of the LH visualization were disproportionately likely to return to the top layer. A probable explanation is that they did not like the clutter in the lower elevation and therefore wanted to return to the higher elevation that we hypothesize is more visually appealing. Similarly, users of the HL visualization seemed reluctant to return to the high elevation; we hypothesize they preferred staying in a visually appealing lower elevation to returning to a cluttered higher elevation.

To assess the effect of density on panning, we measured the number of pan operations in both the top and bottom layers. Nearly all pan operations were performed at one of two elevations: the highest elevation in the top layer and the lowest elevation in the bottom layer. Table 4.3 shows the median number of pan operations in these layers at these elevations.

Our data do not suggest that panning is influenced by constant information density. It appears to be more strongly influenced by the properties of individual layers. Table 4.3 illustrates that participants were more likely to pan in the top layer than in the bottom

[1]The LL version had 10 participants, the LH version had 18 participants, the HL version had 14 participants, and the HH version had 15 participants.

layer. It also shows that users panned more often in low density layers than high density layers. An interesting observation is that the ratio of panning in the top and the bottom layers of the LL visualization is comparable to that in the HL visualization. Similarly, the ratios in LH and HH are similar. This suggests that the ratios are driven more strongly by the density of the bottom layer than by other factors.

Since information density appears to affect user navigation, designers of zoomable applications should take density measurements into account when designing applications. If they do not, users may be influenced by the information density and behave in ways not intended by the designer.

4.8 User response

Many users responded positively to the applet, saying for example that the visualization was "Easier to read than most 3D graphs." However, there were two recurring complaints. First, quite a few users found the panning mechanism difficult to use. Several articulated that they would have preferred scroll bars. Second, several users stated that they found the task confusing. Our intent was that participants would infer a relationship between the variables presented and the variable they were supposed to predict (revenue growth). For example, they might infer that companies with fewer employees would be smaller and therefore more likely to experience rapid growth. We hoped that they would infer this relationship by exploring the relationships that were explicitly present in the graph. The participants who said they had been confused appeared from their comments to have performed this inferential task, but were not certain they had been correct in doing so. In retrospect, the inferential nature of the task should have been made more clear.

4.9 Limitations

There are a number of obvious limitations in this pilot study. While the World-Wide Web was a good way to reach a broad spectrum of users, the participants and the testing conditions were not controlled. Further, the speed of the computer was not controlled. Based on user comments, *e.g.*, one of the users of the HH version said the applet was "nicely responsive," we do not have reason to believe computer speed influenced the results. However, a more formal study should minimally ensure that all layers within a

visualization are equally responsive.

4.10 Conclusions

We have performed an informal study of user navigation in applications with and without constant information density. Our preliminary results suggest that information density affects user navigation and should therefore be taken into account during the construction of visualizations.

Chapter 5

End-user control of information density

5.1 Introduction

Previously, we introduced DataSplash, a direct-manipulation interface for constructing zoomable database visualizations [2].[1] In DataSplash, objects appear in a two-dimensional canvas. Users view the canvas as if with a camera that moves in three-dimensional space but always points straight down at the canvas. Users can pan across the canvas (changing the x,y location of the camera). Users can also zoom in and out above the canvas (changing the z location, or elevation, of the camera). Because a given set of objects looks different when seen from different elevations, a visualization that is appealing at one elevation is likely to be unappealing at another. Therefore, DataSplash objects change representation as users zoom closer to them (recall that this functionality is known as semantic zoom). For example, when a user zooms closer to a circle representing a city, the name of the city may appear next to the circle. DataSplash provides a unique mechanism, the layer manager, which allows users to visually program the way objects behave during

[1] Much of the material in this chapter appears in [51].

zooming. The resulting program is called an *application*.

However, as we discussed, programming the behavior of objects during zooming is a challenging task. As a result, objects in semantic zoom applications may have inappropriate visual complexity. Additionally, the amount of information in the display can vary significantly as the user pans and zooms.

A guiding principle that helps users construct visualizations with appropriate detail can be derived from the *Principle of Constant Information Density*, drawn from the cartographic literature [16, 45]. This principle states that the number of objects per display unit should be constant. A more general formulation posits that the amount of information (as defined by metrics discussed below) should remain constant as the user pans and zooms. To maintain constant information density, either (1) objects should be shown at greater detail when the user is closer to them, or (2) more objects should appear as the user zooms into the canvas, or (3) both.

We define a *well-formed* application as one that conforms to the Principle of Constant Information Density. By definition, as the user pans and zooms in a well-formed application, the number of objects visible in the display should remain constant. We present a system that interactively guides users in the construction of well-formed applications. The system not only provides the user with feedback about specific applications, but also teaches the user about the properties of density functions in general. For example, a user who has not thought explicitly about the relationship between zooming and the number of visible objects receives an intuitive introduction to the concept by using our system, which we call VIDA_0. VIDA is an acronym for Visual Information Density Adjuster; we use the notation VIDA_0 to distinguish this version of the system from the one discussed in Chapter 6.

We believe our system is the first environment that interactively instructs and guides users in the construction of applications with constant information density. It has the added advantages of being a direct manipulation interface and of producing general-purpose applications, rather than being limited to a specific domain such as cartography.

In the following sections, we discuss how we have modified the DataSplash environment to provide visual feedback about the density of applications, and we draw conclusions.

5.2 Density feedback

Users of the original DataSplash layer manager find it difficult to construct visualizations that have appropriate detail at all elevations. In this section, we describe how users can express their preferences for application information density, how the system lets them know when these preferences are not being met, and how the user can correct such conditions. We then briefly discuss alternative schemes for density measurement.

We begin by considering data that is distributed uniformly in the x and y dimensions. At the end of this section, we discuss skewed distributions.

5.2.1 Measuring information density

We have designed a software framework in which we can explore generalizations of the Principle of Constant Information Density. Since it is likely that different density metrics will be appropriate for different applications, we do not limit ourselves to a single specific metric. Instead, we provide extension interfaces so that expert users can customize density metrics for their applications. We support two such interfaces, one to measure density and the other to bound it.

Information density metrics are expressed using density functions. Density functions return the associated density metric value for a given layer at a given elevation. Expert users may write C++ code to define new density functions to supplement those already included in the system.

Since we have assumed uniform distribution in the x and y dimensions, the density of a layer is well-defined at all elevations. Therefore, for each elevation, the system sums the density values of all the active layers to find a cumulative density. The system maintains maximum and minimum bounds on this cumulative density.[2] These bounds, which can be modified by the expert user at run-time, define a range of acceptable densities; therefore, rather than being literally constant, the application's information density at each elevation is expected to fall within this range. We use this technique rather than defining acceptable density in terms of a single constant value because the latter approach would require changing the displayed information every time the elevation changed.

[2]Note that the system does not enforce the density bounds. Instead, it provides users with feedback using the mechanisms described below.

5.2.2 Providing visual density feedback

We have modified two of the display objects contained in the layer manager so that their visual properties give the user an indication of the application's information density. Specifically, we have changed the shape of the layer bars and the color of the layer manager trim to provide such feedback.

First, the width of each layer bar now reflects the density of the corresponding layer at the given elevation. The original DataSplash layer manager does not associate the layer bar width with any property of the layer. We have extended the layer manager so that the width of a layer bar at a given elevation is exactly proportional to the layer's density at that elevation. The scale is relative to the maximum cumulative density, such that a single layer bar of maximum width represents a layer with 100% of the maximum cumulative density. (This implies that the cumulative layer bar widths at a given elevation in a well-formed application are no greater than this maximum width.)

Exceptional conditions can occur because the system does not enforce the cumulative density bounds. We crop layer bars at the maximum width to allow the bars to have fixed horizontal spacing. We also enforce a minimum width to prevent layer bars from becoming invisible.

Second, the layer manager now relates the cumulative density value at each elevation to the density bounds. Notice the tick marks along the left side of the layer manager display in Figure 5.1. There are three possible conditions for a given elevation: it may lie within the density bounds, it may fall below the minimum density bound, or it may exceed the maximum density bound. Each tick mark is assigned one of three colors to indicate which condition pertains to a given elevation.

Figure 5.1 shows a visualization of selected companies from Fortune Magazine's Fortune 500 and Global 500 lists. These companies are displayed as circles in an interactive scatterplot. The x axis represents the percent profit (profit dollars divided by revenue dollars) of the company during a given year. The y axis represents the number of employees of the company. The color of the circle represents the profit of the company in dollars.

The density function in the current implementation measures density by counting the number of objects visible in the display at a given elevation. This metric is that used by cartographers in the original formulation of the Principle of Constant Information Density [45]; as we have mentioned before, many other density metrics are possible. In Figure 5.1,

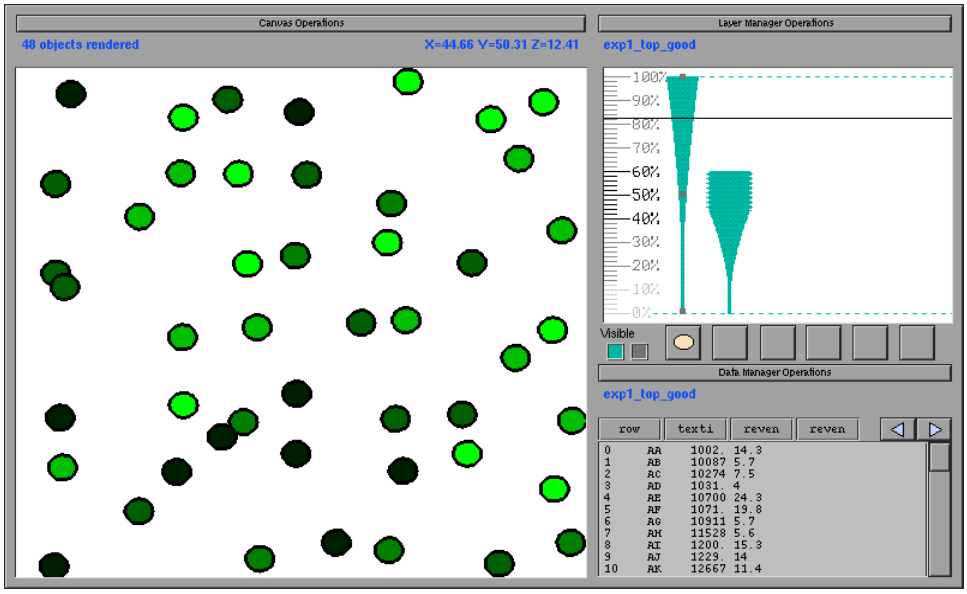


Figure 5.1: A visualization of selected companies from the Fortune 500 and Global 500.

the minimum and maximum density bounds are set to 10 and 100 objects, respectively. The colors of the tick marks (shown in this reproduction as shades of gray) on the left side of the layer manager indicate the density values for this metric at given elevations. Elevations 40%-60% are too dense, elevations 14%-38% and 62%-100% have appropriate density, and elevations 0%-12% are too sparse.

Figure 5.1 highlights some of the interesting properties of the object density metric. First, note that the width of each of the layer bars in Figure 5.1 grows quadratically. To understand this, assume for the moment that the number of objects per unit area in the native data space remains the same. The area of the native space visible in the display increases quadratically as the elevation increases. Consequently, the number of objects visible in the display, and therefore the object density, increases quadratically as well. Second, observe that the rate of change in width is more pronounced for the layer bar on the right. Because the right-hand layer bar contains more objects, its density increases more quickly. While these properties are complex, our graphical illustrations make them more accessible by eliminating the need for users to compute them mentally.

5.2.3 User interaction with the new layer manager

Based on the feedback provided by the extensions described above, users can modify applications as they are constructing them. As the layer manager is currently implemented, there are two primary ways users can change the density of their applications. (In Chapter 7, we propose an extension to the layer manager with which users can prompt the system to create layers with specified densities.) First, users can modify the layer manager, *i.e.*, change the elevations at which layers are active. Second, they can change the contents of layers.

In the DataSplash environment, users may graphically modify the layer manager in two ways. First, they may adjust the top or bottom elevation of a layer bar. When this happens, the shape is extended (according to the width calculation function) as the user makes the adjustment. Users may also graphically drag the entire layer bar up and down to shift the elevation range at which the layer is visible. When this happens, the shape of the bar changes as the user drags it. Additionally, as the user modifies the bar in either of the ways just described, the colors of the tick marks change to reflect the modification. Intuitively, the user moves the bar around, trying to maximize the number of green tick marks (by default our interface uses green to indicate appropriate density).

Second, users can modify the contents of layers. To do this, they may use the paint program interface. For example, to modify the number of objects, they may add or delete objects. If other density metrics (*e.g.*, the number of vertices) are considered, a variety of other operations, such as changing the shapes or colors of objects, affect the density values as well. Users may also modify the contents of layers by using the visual select and join mechanisms described in [30]. These operations affect the number of rows in the table associated with a layer, thereby affecting the number of objects rendered. When the user modifies the contents of a layer using either the paint program interface or the visual select and join mechanisms, the layer bars and tick marks are automatically updated to reflect the change.

5.2.4 Density metrics

Our system currently supports two density metrics, number of objects and number of vertices. There are a number of other metrics that could be used, *e.g.*, Tufte's data density (a count of the number of data values represented by a visualization) [47]. For a thorough

review see [29]. Because the focus of our work is on maintaining constant information density for a given metric rather than on determining good density metrics, we have not yet implemented any additional metrics. However, the interface is independent of the density metric and we have designed the system such that expert users may register their own density functions.

It is our belief that the interface will be particularly useful in teaching application developers about the properties of different density metrics under zooming conditions. For example, the ink metric (the percentage of live pixels) is not elevation-sensitive in the case of uniformly distributed data. As an illustrative example, imagine that the canvas contains a chessboard and that black pixels are live. Since half the pixels are white and half are black, 50% of the pixels are live. Now imagine zooming closer to the chessboard. The view changes considerably, but the pixel distribution remains the same. Therefore, in visualizations that have this type of self-similarity, *e.g.*, fractal visualizations, the information density according to the ink metric is constant at all elevations, which is shown graphically by constant-width layer bars.

5.2.5 Non-uniform data

Recall that the width bars and the cumulative density values are based on the *average* density of each layer. For this reason, they are most appropriate for uniform data sets. However, our techniques can improve visualizations of non-uniform data sets as well. In this subsection, we present an example application of width bars to a non-uniform distribution and discuss its advantages and disadvantages.

Figure 5.2 shows the application of Figures 1.3 and 1.4 augmented with width bars and tick marks colored according to cumulative density. The four layer bars to the right reference cities of different populations. The shapes of the bars indicate that there are few cities in the higher layer bars (the ones associated with larger cities). They illustrate that there are many cities in the two rightmost layers (the ones associated with smaller cities). The width bars and tick marks suggest that this application is cluttered not only at the current elevation, but at other elevations as well. The user can easily adjust the tops and the bottoms of the layer bars based on interactive visual feedback, yielding Figure 5.3. This visualization is a significant improvement over the original.

However, it still contains some regions that are too dense or too sparse. To ad-

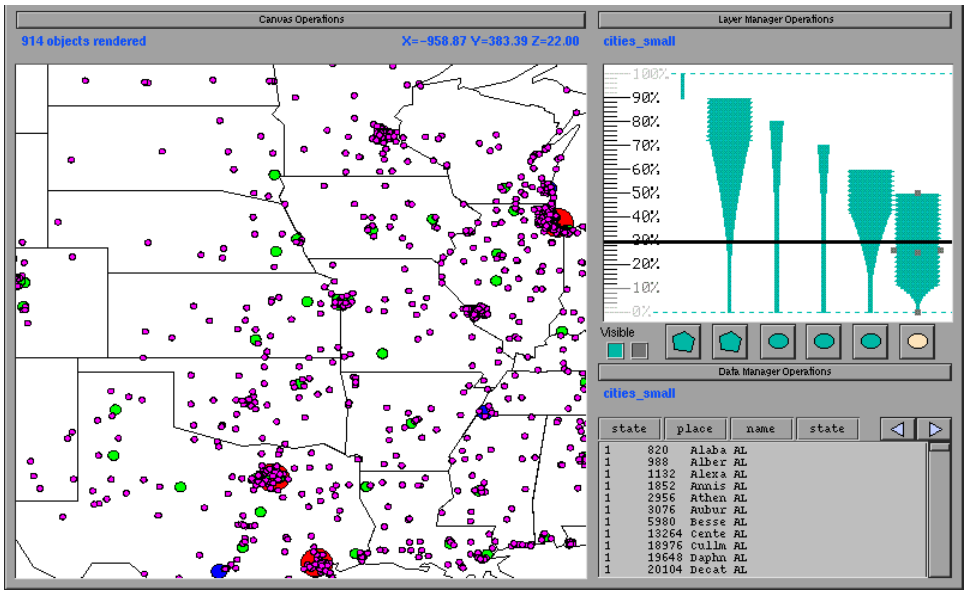


Figure 5.2: A cluttered application.

dress this problem, we augment VIDA$_0$ with additional mechanisms to handle non-uniform distributions. These are discussed in Chapter 6.

5.3 Conclusions

We have introduced the notion of well-formed applications, ones that display an appropriate amount of information (as defined by user-parameterized constraints) at any given elevation. The notion of well-formedness applies not only in our system, but in other visualization systems that support multiple representations as well, *e.g.*, Pad [32, 5]. We have introduced a system, VIDA$_0$, that helps users construct well-formed applications in the DataSplash database visualization environment, both by providing visual feedback on application density and by suggesting modifications to user-constructed applications.

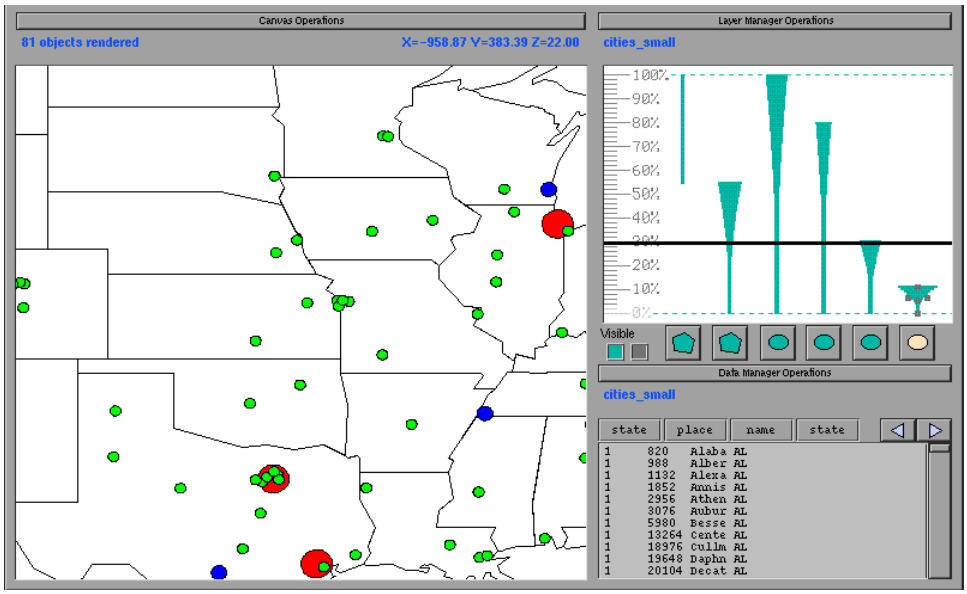


Figure 5.3: A less cluttered application.

Chapter 6

Constant density visualizations of non-uniform distributions of data

6.1 Introduction

In Chapter 5, we presented a system, VIDA_0, which helps users manually construct applications in which overall display density remains constant.[1] Although the total amount of information in the display remains constant as the user zooms, the information within the display may not be distributed uniformly. In other words, this approach ensures uniformity in the z dimension, given a fixed x, y center, but does not ensure uniformity in the x and y dimensions, given a fixed z, nor does it extend naturally to providing such uniformity. Because many naturally occurring data sets have non-uniform distributions, clutter and sparsity commonly occur in subdivisions of the display in VIDA_0 visualizations.

In this chapter, we present a new system that automatically creates displays that are uniform in the x, y, and z dimensions. In the new system, users express constraints about visual representations that should appear in the display. The system applies these constraints to subdivisions of the display such that each subdivision meets a target density

[1] Much of the material in this chapter appears in [50].

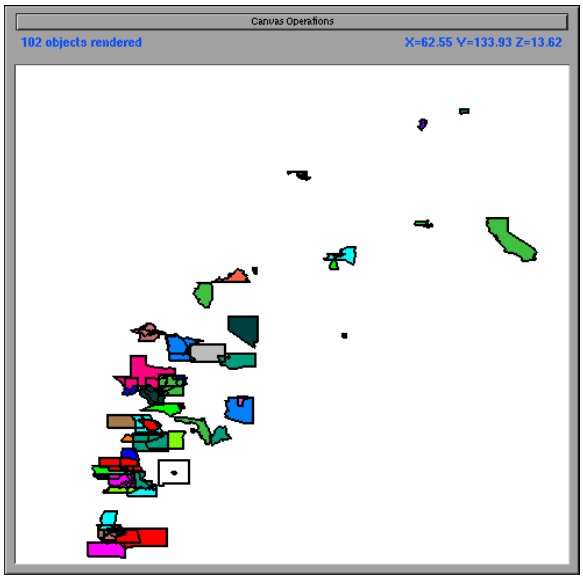


Figure 6.1: DataSplash visualization of census data. x axis shows housing cost and y axis shows income.

value. Henceforth, we continue to refer to the system presented in Chapter 5 as VIDA$_0$, and we refer to the new version simply as VIDA.

Figures 6.1 - 6.3 illustrate the approaches used by DataSplash, VIDA$_0$, and VIDA. Each figure shows an interactive scatterplot of selected states in the United States. On the x and y axes are housing cost and income, respectively. Each state may be graphically represented by a dot or by its polygonal outline. Figure 6.1 shows a visualization of this data that was created in DataSplash. The visualization is obviously cluttered.

Figure 6.2 shows a visualization of this data that was manually constructed based on VIDA$_0$'s feedback about the density of the visualization. In this example, the number of vertices is used as the density metric; density metrics were discussed previously in Section 5.2.1. Because the polygonal representation has high density, VIDA$_0$ recommends changing the display. In response, the user employs the graphical mechanism described in Chapter 5 to replace the polygonal representation with the dot representation. Note that in both Figures 6.1 and 6.2, all objects in a given display are shown with a single graphical representation (a few of the smaller polygons in Figure 6.1 appear to be dots in this reproduction, but they are actually small state outlines).

Figure 6.3 shows VIDA's automatically generated constant information density visualization of this data. Note that different representations are chosen for different objects

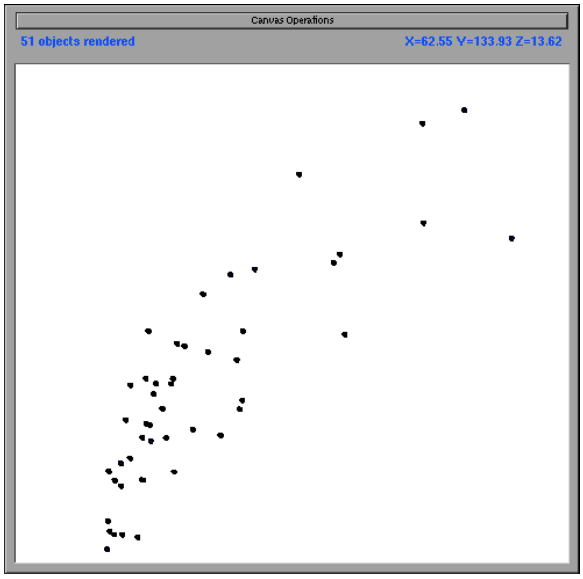


Figure 6.2: VIDA_0 visualization of census data. x axis shows housing cost and y axis shows income.

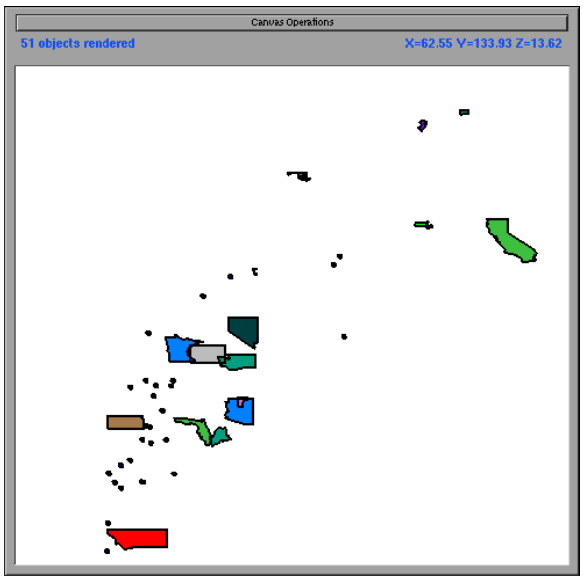


Figure 6.3: VIDA visualization of census data. x axis shows housing cost and y axis shows income.

based on local density. Specifically, VIDA displays dots for objects that appear in dense regions; objects in less dense regions are displayed as polygonal outlines.

In the remainder of this chapter, we discuss VIDA in more detail. In Section 6.2, we elaborate on our technique. We discuss its advantages and disadvantages in Section 6.3, and we conclude in Section 6.4.

6.2 Technique

Recall that the Principle of Constant Information Density states that the amount of information per area should remain constant. Our general approach is to break the screen into subdivisions and fill each subdivision with graphical information such that some target information density value is met.

In this section, we discuss general processes by which density can be changed. We then describe our specific algorithm for creating visualizations and discuss its computational complexity. Finally, we discuss our implementation and provide a number of illustrative examples.

6.2.1 Processes for modifying density

VIDA uses the same method as VIDA_0 to measure density (this method is discussed in detail in Chapter 5). There are two general processes by which this density may be modified. These approaches may be used individually or in combination.

First, different graphical representations of objects appropriate for different elevations, known as *multiscale representations* [18], can be selected according to their density. There are several interesting ways multiscale representations can decrease density (corresponding actions exist to increase density):

- The glyph used to represent an object can be replaced with one of lower density. As one example, a river represented by a polyline with many vertices can be replaced by a simplified representation with fewer vertices.
- Part of the graphical representation of an object may be omitted, *e.g.*, the text labels for a city can be removed, leaving only a dot to represent the city.
- A number of objects can be aggregated. For example, a number of dots in a scatterplot can be replaced by a single larger dot.

Second, objects may be omitted to decrease density. (Correspondingly, objects may be included to increase density.) This is actually a special case of the first item above, in which a glyph is replaced with a glyph of 0 density. However, because this type of replacement has a dramatically different visual effect, we consider it separately.

As described in Chapter 5, the layer manager in VIDA_0 applies these processes uniformly to all objects of a given type. For example, all cities might appear as dots in the display. Alternatively, all cities might be excluded from the display.

However, if we are to provide a uniform display of non-uniform data, objects that differ only in their location in the display must be displayed differently. As an example of non-uniformly applied multiscale representations, a large number of dots in one part of a scatterplot can be replaced with a large circle, while dots in other parts of the same scatterplot can be left in the display. As an example of non-uniformly applied omission, in a traditional map, small cities might not be shown in areas with large populations. However, cities of the same size might appear in less populous areas in the same map. We call this technique *selective omission*.

6.2.2 Algorithm

We have developed an algorithm that controls the display of layers in subdivisions of the screen based on density. (Recall that a layer consists of a data set and a specification of the graphical representation of that data set.) The algorithm first divides the visible screen into a regular nxn grid (n is a parameter, typically set to 10). Every cell in the grid has a goal density. Goal density is the same for every cell in the grid and is configurable. For a given cell, our algorithm calculates the density of each layer.

The algorithm next chooses the layers to fill each cell. We began our experiments with a fairly naïve algorithm that finds the combination of layers yielding the density value closest to the goal density for a given cell. It renders those layers within the boundaries of that cell. While this approach is effective for some applications, it is highly inappropriate for many others. For example, in one application, in some parts of the screen state outlines are rendered without cities while in other parts of the screen cities are rendered without state outlines.

Based on our experience with the naïve algorithm, we decided to allow the user to specify the combinations of layers that are semantically meaningful. We identified two

possible approaches. In the first approach, the user could explicitly register all valid combinations of layers with the system. This is an extensional approach. In the second approach, the user could register rules for combining layers. This is an intensional approach. Because typical DataSplash applications range from 5 to 15 layers, we decided extensional specification would be prohibitively time-consuming for the user. Therefore, we chose the intensional specification mechanism, which we describe in the remainder of this subsection.

To simplify the user's task, we developed intuitive and composable constraints with which the user can express the relationships between layers. Each unit to which constraints may be applied is called a *bundle*; the simplest bundle is a single layer. When a constraint is applied to two bundles, it generates a new bundle to which additional constraints may be applied. If the user wishes to characterize the relationship between two bundles, they specify it using one of the constraints below.

- **Constraint 1** (*mutual exclusivity*): The user can state that two bundles are mutually exclusive. In this case, only one of the two bundles may be displayed within a given subdivision of the screen. This can be useful when one layer is an alternative representation of another. For example, suppose two layers exist for representing cities, one of which represents cities as a dot and one of which represents cities as a circle. For a given city, only one such representation should be displayed. The user specifies a density ordering of the two bundles, *i.e.*, they specify which has higher density.

- **Constraint 2** (*additivity*): The user can state that one bundle may be added to another. In this case, the first bundle may appear alone or with the second bundle in a given subdivision of the screen. For example, the state outline layer may appear alone, or with the city layer. Note that the relationship is not symmetric, *e.g.*, cities may not appear without state outlines.

We illustrate the algorithm with the following example.

Example constraints for cities and labels

Constraint 1 can be used to generate a bundle that specifies that cities may be represented as dots or circles. It can also be used to generate a bundle that specifies that city labels may be represented with a small font or with a large font. Constraint 2 can then be used to specify that the city dots/circles bundle can appear alone or with the city labels bundle. □

We call a set of layers chosen for display a *configuration*. Applying the constraints in our example of cities and labels results in six configurations: (1) a dot, (2) a circle, (3) a dot plus a small-font text label, (4) a dot plus a large-font text label, (5) a circle plus a small-font text label, and (6) a circle plus a large-font text label.

From this set of configurations, our algorithm chooses an ordered list of configurations such that the density of each of the configurations in the list is guaranteed to be non-decreasing. This is desirable because we want to make the choice of representation as consistent and stable as possible (we discuss this issue further in Section 6.3.2). Additionally, we wish to limit the number of configurations to avoid unnecessary visual complexity.

Generation of a density-ordered list is simplified by two observations. First, recall that the density ordering of a mutually exclusive bundle is known from the user's specification. Second, observe that the density ordering of an additive bundle is obvious; a bundle appearing alone has density lower than or equal to that of the same bundle appearing with another bundle.

These observations immediately result in a partial ordering of the set of potential configurations listed above. In the example presented above, (1) has lower density than (2). However, the relative density of, for example, (2) and (3) is not known. VIDA's algorithm proceeds in a depth-first fashion, producing a fully ordered list consistent with the partial ordering. In this example, it chooses (1), (2), (5), and (6).

Note that the constraints are lower-level concepts than multiscale representations and selective omission, which were described in Section 6.2.1. The constraints support these higher-level concepts, as illustrated in Section 6.2.4.

6.2.3 Computational complexity

The current density computations are fast enough that performance is not affected visibly when rendering current main-memory-resident data sets (typical sizes range from hundreds to tens of thousands of objects). In fact, in many cases our techniques improve performance, since they significantly reduce the number of objects that must be rendered.

The current algorithm for computing the density of an individual layer runs in linear time (using number of objects or number of vertices as the density metric). The algorithm that selects the layers to display is more computationally intensive. (In fact, the naïve algorithm that does not consider constraints is provably NP-Complete. This can be

shown by a polynomial reduction from Subset-sum [20] to our problem of choosing a set of layers that sum to a given density. In Subset-sum, given a set S of positive integers, we wish to determine whether there is a subset of S such that the sum of the subset is equal to a given value. Similarly, in our problem of choosing layers, given a set of layers L, each with a density value, we wish to determine whether there is a subset of L such the sum of their densities is equal to a target density value.) In practice, we have found that the number of layers is generally low enough that performance is not affected noticeably.

There are at least two situations in which computation costs can be prohibitive. First, user-defined density metrics can be arbitrarily expensive to compute. Second, larger data sets can increase computation time significantly (albeit linearly). In these situations, the following three techniques can be used to reduce computation time:

- The density values can be stored during or across browsing sessions (currently they are dynamically computed each time the user moves).
- Heuristics can be used to estimate density. Because the density metrics are used to choose representations, approximations are acceptable. This can be particularly useful when updates are frequent.
- More advanced data structures can be used to compute or estimate density values. For example, R-Trees can be modified so that internal nodes contain both bounding rectangles and counts of the number of objects contained in each rectangle. A related technique is used in [3].

6.2.4 Implementation and examples

We have implemented our algorithm as part of VIDA$_0$, which is an extension to the DataSplash database visualization environment [2, 35]. We call the resulting system VIDA. VIDA is implemented in C++, the Mesa graphics programming language [31], XForms [57], and POSTGRES [43], and runs on most UNIX platforms.

In this section, we present three sample visualizations of non-uniform distributions of data. These examples illustrate a number of advantages and disadvantages of VIDA's technique for displaying non-uniform data. These issues are discussed in Section 6.3.

We begin by revisiting Figures 6.1 - 6.3, interactive scatterplots of selected states in the United States. On the x and y axes are housing cost and income, respectively. Each state

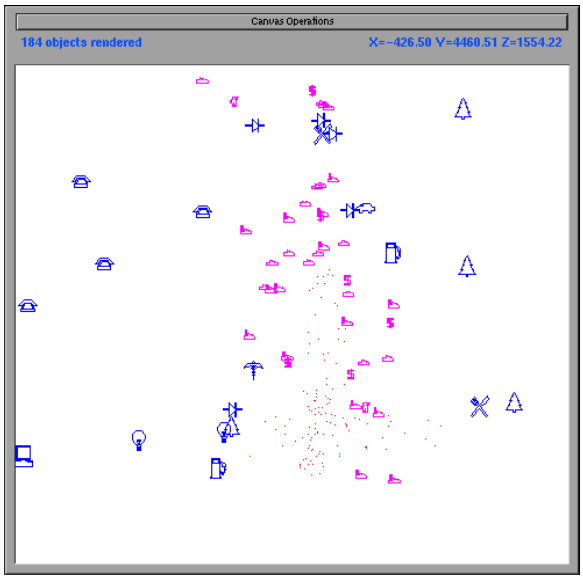


Figure 6.4: VIDA visualization of Fortune 500 data at a high elevation. x axis shows % profit growth and y axis shows number of employees.

may be graphically represented by a dot or by its polygonal outline (each representation is associated with a layer). Previously we discussed Figures 6.1 and 6.2. At this time, we provide more detail about Figure 6.3, which shows VIDA's automatically generated constant information density visualization of this data. In this example, VIDA uses the number of vertices as the density metric. There is a mutually exclusive relationship between the two representations (dot and polygonal outline), so that within a grid cell all states are represented by either a dot or a polygonal outline, but not both.

The second example, pictured in Figures 6.4 and 6.5, shows interactive scatterplots of selected Fortune 500 companies. On the x and y axes are % profit growth and number of employees, respectively. Each company has three potential representations (each associated with one layer): (1) a dot; (2) an icon of the general category of industry to which the company belongs; and (3) an icon of the specific type of industry to which the company belongs. Figure 6.4 shows VIDA's constant information density visualization of this data. As in Figures 6.1 - 6.3, VIDA uses the number of vertices as the density metric. There is a mutually exclusive relationship between the three representations (dot, general category icon, and specific category icon), so that within a grid cell all companies are represented by exactly one of the three possible representations. Figure 6.5 shows a zoomed-in view. Note that if the user zooms in on a fixed set of objects, each object that remains in the display

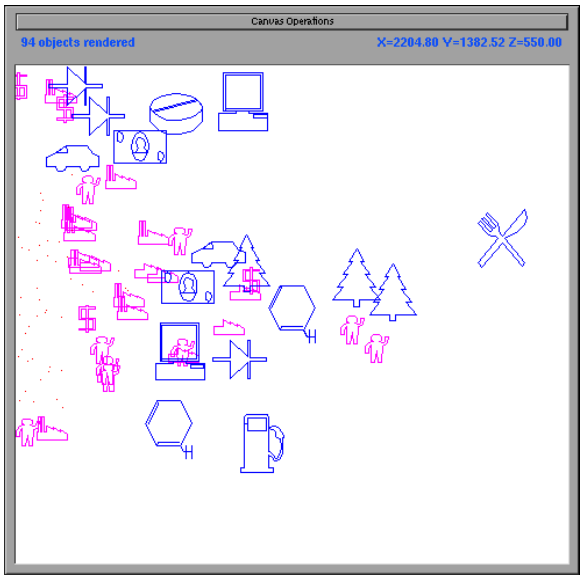


Figure 6.5: Zoomed-in view of visualization of Fortune 500 data. x axis shows % profit growth and y axis shows number of employees.

occupies more screen space. Therefore, when the user zooms in, each object is shown with a more detailed representation.

The third example, shown in Figures 6.6 and 6.7, is a map of cities in the United States (aspects of this visualization were discussed previously in Chapter 5). On the x and y axes are longitude and latitude, respectively. One layer contains the outline of the United States. Four additional layers contain cities partitioned by population. In the naïve DataSplash visualization shown in Figure 6.6, many regions of the visualization are too sparse. In this example, VIDA uses the number of objects as the density metric. The additive relationship pertains among these layers, so that any given grid cell may contain only the outline of the United States, the outline of the United States plus the largest cities, the outline of the United States plus the largest and second largest groups of cities, *etc.* In Figure 6.7, VIDA's use of additivity makes the display more uniform by showing cities of lesser populations in regions where no large cities exist.

Multiscale representations are illustrated in Figures 6.1 - 6.5. In each of these visualizations, a given data point is always visible. Density is modified by changing the representation of the object representing that data point, *e.g.*, from a dot to a polygonal outline. Selective omission is illustrated in Figures 6.6 and 6.7. In each of these visualizations, not all objects are visible at a given elevation. Density is modified by selectively omitting objects,

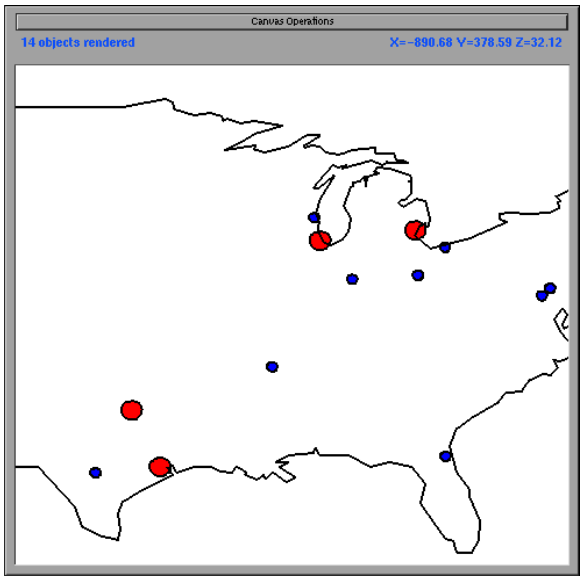


Figure 6.6: Naïve DataSplash visualization of population data.

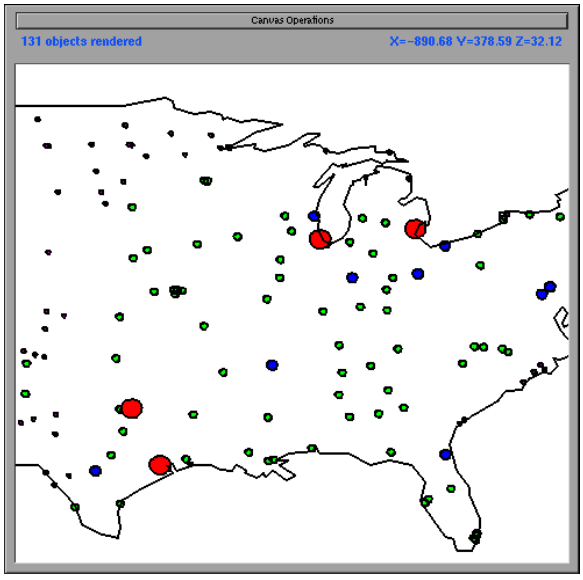


Figure 6.7: VIDA visualization of population data.

e.g., smaller cities are not visible from higher elevations. In these examples, it happens to be the case that multiscale representations are supported by mutual exclusivity and selective omission is supported by additivity. However, each of these processes can be generated by either constraint.

6.3 Discussion

In this section, we discuss the effectiveness of constant information density displays in non-cartographic domains. We then assess our methods for choosing representations to create uniform density displays.

6.3.1 Effectiveness in non-cartographic domains

Historically, the Principle of Constant Information Density has been used in the cartographic domain. In VIDA, we have applied this principle to non-uniform data in both cartographic and non-cartographic domains. In this subsection, we discuss our informal observations of its effectiveness for different tasks.

We begin by discussing the utility of constant information density displays. We then assess individual characteristics of the two processes by which constant information density displays are created, multiscale representation and selective omission. For each technique, we discuss advantages and disadvantages, as well as potential improvements.

Constant information density displays

Constant information density displays have a number of advantages. For example, if the target density value is chosen appropriately, overplotting is minimized and use of the available display space is maximized. Further, the displays are visually appealing, according to informal discussions with viewers.

A potential disadvantage of the technique is that it might give users a distorted perception of the actual density of the underlying data space. We have two responses to this argument.

First, we observe that because cluttered visualizations contain overplotting, they do not give an accurate representation of the distribution of the data. Arguably, the current VIDA representations are at least as clear as traditional representations. However,

both traditional displays and VIDA displays can be improved according to the following observation: enumeration (by simply plotting all items) is not the best way to characterize density. Therefore, we recommend developing visualizations compatible with the Principle of Constant Information Density to show density distributions. For example, a scatterplot can use aggregation in the following manner: when a region of the screen contains more than a given number of dots, these dots can replaced by a single, larger dot of a different color.

Second, we observe that one can apply the Principle of Constant Information Density only to those dimensions in which the user is not explicitly studying distribution. For example, if the user is looking for high-density areas in the x and y dimensions, it might be appropriate to apply the Principle of Constant Information Density only in the z dimension using the technique described in Chapter 5.

Multiscale representation

Note that multiscale representation of data according to density shows outliers with more detail (and therefore often with more screen space). We argue that showing more detail for outliers assists in the detection and investigation of anomalies. However, because this technique makes outliers disproportionately prominent, there is a danger that this technique will overemphasize outliers, which may be inappropriate for some applications.

Selective omission

While omission does achieve constant information density displays, it can be fairly misleading. Consider Figure 6.7. This visualization is consistent with the common cartographic convention of omission in paper maps. However, if this same technique were applied to a different type of visualization, *e.g.*, a scatterplot, the user might incorrectly infer that no data existed in a specific region when in fact such data did exist.

It is our belief the display could be significantly improved by the addition of visual cues that indicate that objects have been removed. Such cues fall into two general categories.

First, cues can indicate the regions from which objects have been omitted. For example, Magic Lenses might be placed over regions in the display to indicate that they are being shown at a different level of detail [8, 15].

Second, cues can encode information about the distortion that has been applied.

For example, the background of the visualization can be colored to indicate density in various regions (note that it would be interesting to color the backgrounds of a set of Magic Lenses in this way). Alternately, objects can be blurred to indicate the accuracy of the representation; blurrier areas can represent areas in which higher distortion has occurred. Similar techniques are used in [3].

6.3.2 Choice of representations for display

In general, the gridding mechanism and simple constraints have proven quite powerful, supporting a number of diverse applications. Much of this utility is evident in the examples provided above. In this subsection, we discuss the limitations we have identified with the gridding and constraint mechanisms and suggest potential improvements.

Perceptible grid boundaries

VIDA generally uses a 10x10 regular grid, and we find that boundaries are not easily perceptible in visualizations of non-uniform data. However, when the data distribution is highly regular, repeating patterns occur, making the boundaries more obvious (see for example Figure 6.8). In cases in which it is desirable to use a less perceptible grid, alternative subdivisions such as hexagonal and non-regular grids could be used.

Flickering representations in the grid

Despite the fact that the grid boundaries are not readily apparent, there is a significant drawback to the grid-based approach. In our current implementation, the grid boundaries are relative to the screen (one can imagine a transparent grid that moves independently above the two-dimensional canvas). In this model, the contents of a given grid cell change as the user pans or zooms.

Consider the case when the user pans. Suppose an object is in a grid cell that has high density. In this position, it is displayed with a low-density representation. If the user pans slightly, the grid shifts. In the new subdivision of the screen, the object may be in a grid cell that has low density and therefore may be displayed with a high-density representation. A similar problem exists when the user zooms. For example, during one continuous inward zoom, a low density representation might be replaced by a high density representation and then be replaced by the original low density representation.

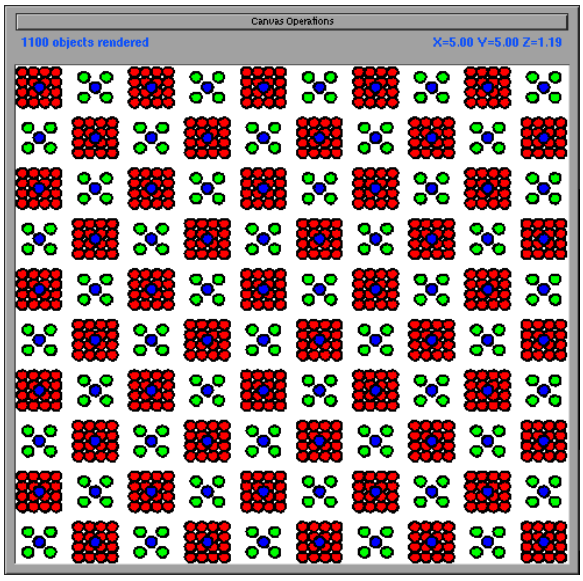


Figure 6.8: Visualization of an artificially-created data set with a regular distribution pattern.

Such flickering representations are highly distracting and create undesirable visual effects. Panning flicker can be solved easily by registering the grid to the canvas, *i.e.*, by embedding it in the underlying x,y space. Unfortunately, zooming flicker can not be solved easily because no single size for a grid cell is appropriate for all elevations. A spatial data structure such as a quad tree that subdivides the space might seem appropriate. With such a structure, different grid-cell sizes could be chosen for different elevations. However, at each transition from one cell size to another, flickering could still occur. Possible solutions include choosing representation on a per-object rather than a per-grid-cell basis. Fortunately, the zooming case is not as visually obtrusive as the panning case.

Aggregated objects in the grid

Our current implementation chooses layers to render within each grid cell. It does not explicitly take into account the semantic relationship between objects in different layers. For example, it does not explicitly consider that the United States is a single object that contains each of the individual states. We would like the system to render either the United States, or all the individual states. However, these semantics are not enforced in the current implementation; in some situations in which the objects are in multiple grid cells, states are rendered in some cells but not in others. An object-based model for display would

ameliorate this problem.

Allocation

In some situations, the user may wish to specify the relative visual resources to be allocated to given bundles. For example, suppose an application has two bundles, one for political boundaries and one for cities. Each of these bundles may contain a number of different representations with widely varying density; the political boundaries bundle may range from a country outline to county outlines, while the city bundle may contain a number of layers with different numbers of cities. It would be useful to provide a mechanism with which the user could specify that at any given time, $n\%$ of the visual density should be allocated to the political boundaries and the remainder should be allocated to the cities. One possible interface would be to provide the user with sliders representing the resources allocated to each bundle. This mechanism could be used, for example, to ensure that a certain type of feature is always present in the display.

6.4 Conclusions

We have presented VIDA, a system that creates visualizations that have uniform information density in the x, y, and z dimensions. VIDA visualizations have this uniformity even when the data sets being displayed are non-uniformly distributed. We have discussed the advantages and disadvantages of the visualizations created by VIDA, particularly in non-cartographic domains. VIDA visualizations are automatically constructed using constraints specified by the user. We have described these constraints and discussed their potential usefulness.

Chapter 7

Semi-automated adjustment of density

7.1 Introduction

In the previous two chapters, we discussed using the Principle of Constant Information Density to choose when to display given layers.[1] In this chapter, we consider using the same principle to determine the contents of layers.

We first consider the case of adjusting the contents of a single layer. We present a taxonomy of data manipulation and graphical operations that can be performed to adjust the density of a given layer. We next discuss ways to generate new visualizations using these operations. Finally, we describe an interface with which users can browse these new visualizations. Note that while this work is designed in the context of the VIDA model, it has not been implemented.

[1] Much of the material in this chapter appears in [51].

7.2 Changing layer density

In this section, we describe *modification functions* that can be applied to a layer to modify its density. These functions operate on one of two components of the layer, the rows in the database table and the graphical representation of the data. Functions come in pairs, one that decreases density and one that increases density.

To modify the data, VIDA can create views of the table. Basic views include simple selection or aggregation queries. Because more complex views may be desirable, VIDA can also consider views created by expert users as potential modifications to the database table. To modify the graphical representation, VIDA can change the glyph for an object.

Table 7.1 details modification functions that decrease visual density. The table uses the following visualization of cities in the United States as an example. Suppose the user has created a layer based on a table of cities that includes fields for latitude, longitude, and population of each city. The graphical representation of the user-created layer is a gray circle placed at the longitude, latitude location of each city. The circle is assigned a size based on the population of the city. This original representation appears in the first row of the table. The visible area is a zoomed-in view of Baltimore and Washington, D.C., in the United States.

The following rows in the table show how that visualization changes in response to the application of various modification functions. For each modification function, the table presents an example of a specific modification, an example of a density metric affected by that modification (in many cases multiple metrics are affected; for brevity we identify only one per modification function), and the visualization resulting from the application of that modification to the original visualization.

The first three modifications (select, aggregate, and reclassify) apply to the data. When the given select operation is applied, only the largest cities remain visible. When the given aggregate operation is applied, the system aggregates cities by states. Chesapeake Bay can be seen in the resulting visualization (the water is white, while the land is gray). Both select and aggregate reduce the number of visible objects. The given reclassify operation classifies cities into two groups according to their population; the resulting visualization has fewer sizes than the original.

The remaining four operations (change shape, change size, remove attribute asso-

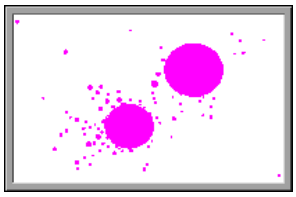
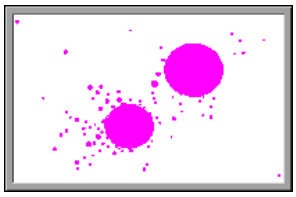
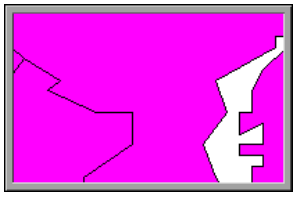
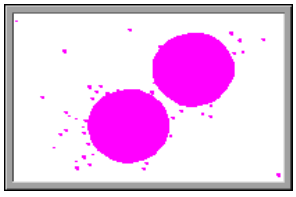
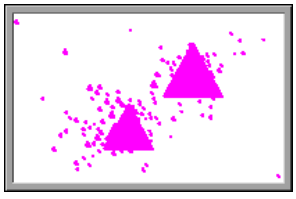
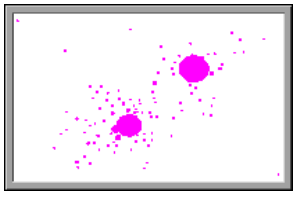
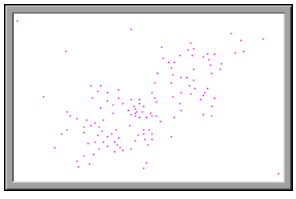
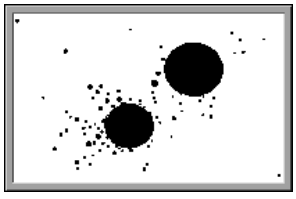
Original visualization	
Select *Restrict to cities with population > n* Decreases number of visible objects	
Aggregate *Aggregate cities by state* Decreases number of visible objects	
Reclassify *Assign to population brackets* Decreases number of sizes	
Change shape *Change circles to triangles* Decreases amount of ink	
Change size *Scale circle radius* Decreases amount of ink	
Remove attribute association *Disassociate size from population* Decreases data density	
Change color *Change color from gray to black* Decreases number of colors	

Table 7.1: Modification functions to decrease density.

ciation, and change color) are modifications to the graphical representation of each object. When the shapes are changed from circles to triangles, or when the size of the circles is changed, the amount of ink is decreased. When the association between population and circle size is removed, the data density (the number of data values represented) is decreased. Finally, changing the color may affect the total number of colors in the visualization (in this example, this effect only occurs when other layers are considered as well).[2]

Observe that not every modification function decreases/increases density for a given metric. For example, the size of an object can be decreased (to reduce the amount of ink) or increased (to increase the amount of ink). However, this operation may have little or no effect on the number of objects being displayed. Also observe that in some cases a modification function may in fact decrease density according to one metric while increasing it according to another metric.

As mentioned above, expert users may register new density functions. When they do so, they may also identify modification functions that will affect the corresponding metric. If they perform this task, VIDA will be able to suggest modifications based on the new density metric.

7.3 Complexity of transformation space

In Section 7.2, we described a system that suggests modifications to individual layers. In this section, we consider the generation of applications that are well-formed for all layers at all elevations. We call such a new application a *transformation.*

The space of potential transformations is very large. Consider that the user has provided n layers. Suppose that using the modification functions described above we can generate m versions of each layer (including the original version and the null version).

Consider the calculation of a transformation that is well-formed at a specific elevation. Assume we may choose only one version of each user-provided layer at a time. Then there are m^n possible groups of layers. Since ordering is significant, there are a total of $m^n!$ possible configurations at each elevation.

[2]The graphical representation manipulation functions represent a fairly comprehensive set in the context of our system. Note Bertin's observation that there are eight variables that provide information in two-dimensional graphics (x and y position, size, value, texture, color, orientation, and shape) [7]. Our system does not modify value, texture, or orientation as the former two do not pertain in VIDA, and the latter is not clearly desirable.

The configurations for each elevation must then be combined to form a single application. Obviously, there are a number of rules for this composition, *e.g.*, if a single layer appears at multiple elevations, these elevations should be contiguous.

Fortunately, we do not need to generate an optimal solution. Instead, we propose to allow the user to browse the search space. To support the user in this task, we propose one mechanism to limit the search space and another to organize it. We describe each of these methods in turn.

7.3.1 Rule system

In this subsection, we propose a rule system with which users can limit the space of possible transformations.

In general, rules take the form

```
If Condition, Then Action
```

In the design, the system provides a number of rules. Expert users may register additional rules.

Our primary conditions pertain to the Principle of Constant Information Density. In general, these conditions compare information density of a visualization (as measured by some criteria) to some value:

```
Density Operator Value
```

For example, a specific condition might be:

```
Number_of_objects < 50
```

As discussed above, the system provides a number of functions that can be used to assess density. Additionally, expert users may register such functions. Potential actions are those enumerated in Table 7.1.

Using these rules, applications are modified to adhere to specific resource constraints, *e.g.*, "there should no fewer than 20 objects on the screen at a given time" or "at least 20% of the pixels should be colored at a given time." The user is provided a simple forms interface in which they may tune the specific values assigned to these constraints.

7.3.2 Edit distance

We define *edit distance* to be the degree to which the system has modified the original application of the user (each action may have an associated edit distance). There are a number of ways edit distance might be defined. One such metric is the number of modification operations applied to the original visualization. Another possible metric is the number of pixels that change. Experimentation will be necessary to identify a good metric(s).

Recall that we wish to allow the user to browse the solution space. Edit distance is a natural mechanism with which to organize this space. For example, the initial set of transformations presented to the user could be chosen to represent disparate edit distance values. This would ensure that the user would have a broad range of alternatives from which to choose. The user could then incrementally refine their search. A possible interface is discussed further below.

7.4 Interface

We propose that the user interact with the system in the following three phases: edit (user); transform (system); and select (user). In this section, we describe each of these phases in turn.

7.4.1 Edit

The user begins by editing an application as in the current system. Specifically, they may graphically resize layers, changing the elevations at which they are visible. Additionally, they may add, delete, reorder, or modify the contents of layers. The result may be a visualization such as that presented in Figure 5.2, repeated here as Figure 7.1.

7.4.2 Transform

To request feedback on an application they have constructed, the user makes an explicit gesture in the interface. For example, they might press a "Transform" button. Alternately, recall that we have extended the layer manager so that the width of a layer bar represents its density. To invoke the semi-automated adjustment of a layer, users could graphically adjust the width of that layer.

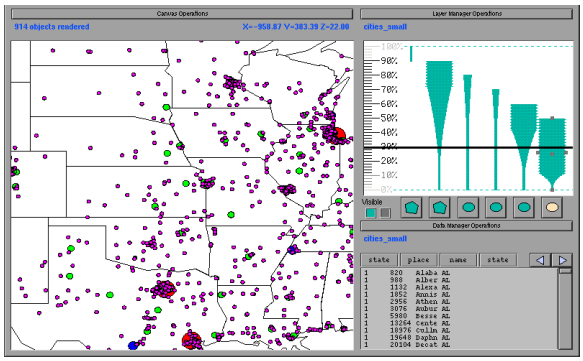


Figure 7.1: A cluttered application.

In response to the user's gesture, the system generates transformations using the approach detailed in Section 7.3. Recall that a transformation is a well-formed application derived by modifying a user-created application.

7.4.3 Select

The system presents a small number of transformations to the user in a *transformation canvas*. Each transformation in the canvas appears as a portal. (The user may configure the number of transformations displayed; four is the default, since that is roughly the number of portals that fits conveniently on a VIDA canvas when they contain a large amount of detail.)

The user may go through the portal of any of the proposed transformations and explore it. If the user wishes to view other transformations they may press the "Transform" button while in the new canvas; the system creates a new transformation canvas in response. Alternatively, they may go back to the previous transformation canvas and enter a different transformation. In this way, the user can incrementally refine their search for a desirable transformation. When the user finds an option they like, they indicate that they accept it.

Because the portal representation of each transformation is used to direct the search, it is desirable for this representation to provide as much information about the transformation as possible. A particularly dense representation is a picture of the layer manager for the application. Therefore, each portal contains a visualization of the layer manager of a transformation. When the user zooms closer to the transformation canvas, the visualization displayed for each transformation appears.

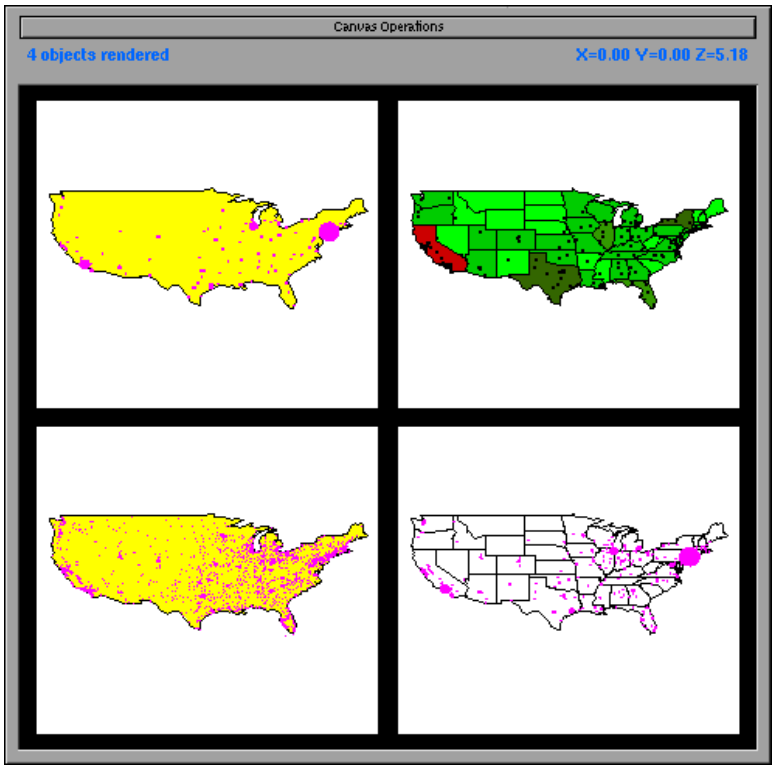


Figure 7.2: The transformation canvas.

Figure 7.2 shows a sample transformation canvas. Each visualization shows population data for the United States. A variety of disparate choices are available to the user. In the options on the left, the state outlines have been removed to reduce density. The option in the upper-right is particularly interesting. Circles (cities) in the original application were associated with population. In this transformation, the association between circle radius and population has been removed. However, an association between state color and population has been added.

7.5 Conclusions

In Chapters 5 and 6, we discussed using the Principle of Constant Information Density to choose when to display given layers. In this chapter, we presented a method for using the same principle to determine the contents of layers, including a taxonomy for modifying density using data manipulation and graphical representation operators and mechanisms

and interfaces for modifying density of individual layers and entire visualizations.

Chapter 8

Goal-directed zoom

8.1 Introduction

In Chapters 5, 6, and 7, we considered ways to construct visualizations with constant information density.[1] In this chapter, we discuss a new navigation method that is generally useful and is particularly appropriate for constant information density visualizations.

As we have discussed previously, visualization systems commonly represent objects in a two-dimensional canvas over which the user may pan and zoom. In such systems, zooming changes the user's distance from the canvas, also known as the perceived *elevation*. Because the amount of display space available to an object varies with elevation, a graphical representation of an object that has appropriate visual complexity at one elevation may have inappropriate visual complexity at other elevations. Many zooming systems address this issue by supporting multiple graphical representations of objects.

Existing systems reflect this balance between elevation and object representation in one of two ways. In naïve systems, elevation and representation are decoupled; the user chooses a representation and then zooms until the display "looks right." More sophisticated

[1] Much of the material in this chapter appears in [52].

systems couple elevation and representation using semantic zoom, so that when the end user zooms to a given elevation, the system displays each object using the representation valid at that elevation.

Consider the relationship between zooming and choice of representation in such systems. In naïve systems, elevation and choice of representation are controlled independently by the user. In semantic zoom systems, the elevation determines the choice of representation. An alternative is a system in which the choice of representation determines the elevation. We call the functionality provided by such a system *goal-directed zoom.*

In the next section, we describe the desirable characteristics of goal-directed zoom. We then present our design and implementation of a system to support the end-user construction of goal-directed zoom visualizations. Finally, we present an extended example and conclude.

8.2 Characteristics of goal-directed zoom

We propose that a goal-directed zoom system ideally possesses at least the following three properties:

1. **Menus**. The system presents the user with a selection of possible object representations. (The user should be able to invoke this selection using some lightweight mechanism, *e.g.*, a simple mouse click.)

2. **Previews**. The selection mechanism graphically depicts the possible representations. For example, suppose a city can be displayed at several levels of detail ranging from a dot to a display of all the buildings in the city. An icon of a house could indicate the most detailed representation.

3. **Automatic zoom**. When the user selects a representation, the system zooms to an elevation "appropriate" to that representation. (This elevation may be determined by ranges specified in a semantic zoom system, or it may be calculated in some other manner. One such method is described below.)

Note that each of these properties can apply either to a set of objects present in a visualization or to individual objects in a visualization. Ideally, these properties apply to an individual object of interest, yielding the following additional guidelines:

4. The menu is appropriate for the specific type of object selected.

5. The graphical options presented to the user are based on the specific object selected.

6. The elevation to which the system zooms depends on the specific object selected.

A number of systems provide some subset of the functionality listed above, but to our knowledge, no system meets all of these criteria. For example, BigBook [22] an online directory, provides maps with iconic zoom buttons. However, the buttons zoom to fixed elevations and the icons are abstract scale indicators. Specifically, they are pine trees of different sizes, which indicate different scales but fail to provide any information about the contents of each layer. As another example, our work has some similarities to Magic Lenses [8] or portal filters [32]. However, while these mechanisms show different representations of objects, they do not zoom automatically to display appropriate detail for a given object. Finally, Pad provides primitives that could be used to support goal-directed zooming, but does not support it directly. The Pad++ web browser, for example, partially supports property 3, automatic zoom [6].

8.3 A goal-directed zoom system

We have designed and implemented goal-directed zoom in VIDA. When users click on an object in a visualization, VIDA presents a menu of the possible graphical representations of that object. When the user selects a representation, the system automatically pans so the selected object appears in the center of the visualization and then zooms to the elevation at which the selected representation has appropriate visual density. (Appropriate visual density is defined using the metrics described in Chapter 5.) The current implementation supports all of the listed properties but 5.

Figure 8.1 shows an example user interaction. The visualization is an interactive scatterplot of selected Fortune 500 companies; on the x and y axes are %profit growth and number of employees, respectively. (This visualization also appears in Figures 6.4 and 6.5, without the navigational functionality provided by goal-directed zoom.) Each company has three potential representations: (1) a dot; (2) an icon of the general category of industry to which the company belongs; and (3) an icon of the specific type of industry to which the company belongs.

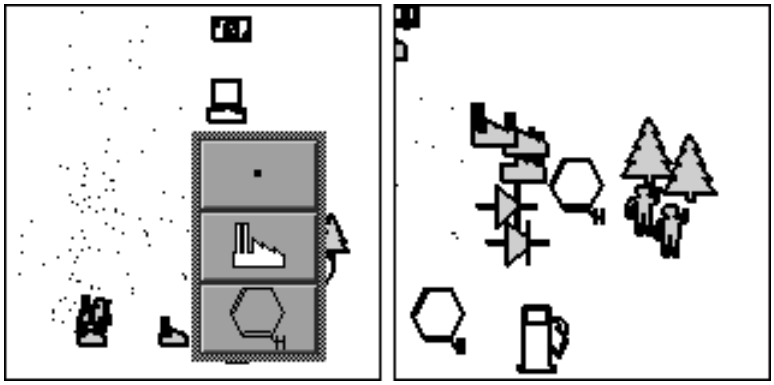


Figure 8.1: Sample goal-directed zoom interaction.

Suppose the user is browsing the visualization and wants more information about companies with high profit growth and a small number of employees. When the user selects such a company in the visualization, a menu of the possible graphical representations of that company appears above it (Figure 8.1, left side). In this case, the factory icon in the middle of the menu indicates that the selected company belongs to the heavy industry sector and the benzene ring at the bottom of the menu indicates more specifically that the company is a chemical company.

Now suppose the user selects the most detailed representation (the benzene ring). The system automatically pans and zooms so that the selected company is centered in the screen and zooms until the benzene ring becomes visible (Figure 8.1, right side). From this vantage point, the user learns that, unexpectedly, only a small number of the high growth companies with few employees are high-technology companies (many of them are timber, service, or heavy industry companies). Note that because VIDA chooses representations based on local density, the objects are presented at varying levels of graphical detail. In this case, some of the companies surrounding the benzene ring are represented as a general category, while others are represented as a specific industry. Companies in slightly denser regions are represented as dots.

8.4 Conclusions

We have presented a novel zoom method. This method, goal-directed zoom, allows users to directly control the choice of graphical representation of an object. We have

implemented goal-directed zoom in VIDA.

Part III

Future work and conclusions

Chapter 9

Future work

We have identified a number of issues that would be interesting to study in the future. In this chapter, we present future directions for buffering, weak inversion/verification, and constant information density.

9.1 Buffering

There are many potential directions for further research. Certainly a variety of other heuristics could be examined. Additionally, extending our model and simulator to consider modification in addition to search queries would be straightforward. Further, we could consider optimal buffer allocation for a multiuser buffer pool, *i.e.*, when the amount of buffer space for a recipe can vary over the course of the query path.

A more complex extension would consider buffering of partial results. First, modifications may affect only part of a box result. Second, since browsers can display a subset of a box result, it may not be necessary to calculate an entire box result. In the context of buffering of partial results, slaved browsers [55] raise an additional complication. When two browsers are slaved together, examining a partial result in one browser spawns a process that generates a corresponding partial result in another browser. The model could be extended to consider this type of dependency in the access pattern.

9.2 Weak inversion and verification

We are currently exploring several ways in which weak inversion and verification can be applied to optimization problems such as efficient rematerialization of intermediate results, efficient materialization of partial results, and reuse of common subexpressions.

9.2.1 Efficient rematerialization of intermediate results

In Chapter 3, we made the simplifying assumption that all intermediate results in a given chain of dataflow operators had been materialized. However, many dataflow systems cache their intermediate results (*e.g.*, Data Explorer [25]). Suppose the user of a caching system finds an anomaly and attempts to view an intermediate datum that contributed to it. In the worst case, the system must rematerialize all intermediate results simply to recreate one datum in one intermediate step.

Weak inversion enables us to recreate lost intermediate results efficiently. Previously, we discussed invoking f^{-w}s on either user-specified attribute values of interest (*i.e.*, the original image) or attribute values that contributed to the computation of those interesting values (*i.e.*, the verified inverse images in the intermediate results). However, the system could also invoke f^{-w}s on the constant values found in the σ^{-w} filters (which were generated by f^{-w}s operating on images to their right in the dataflow diagram). Using this fact, the system can generate a sequence of σ^{-w} filters all the way to the base table of our dataflow chain. It can then apply these σ^{-w} filters as additional selections at each intermediate step in the chain, reducing the amount of data that must be processed.

9.2.2 Efficient materialization of partial results

The technique described above extends trivially to the efficient materialization of partial results. For example, if the system allows the user to specify bounds on the (end result) regions to be visualized, the coordinates of that bounded region become another set of constants to which the system can apply f^{-w}s. This implies that it can generate a chain of weak-inverse selections similar to that described in Section 9.2.1, again reducing the amount of data that must be processed.

9.2.3 Reuse of common subexpressions

The process of breaking dataflow graphs into linear components may produce components with shared steps. This is undesirable because I^{-w}s and I^{-v}s may be computed redundantly. However, the problem is not as simple as finding overlapping steps because the system can only share inverse images between two components if, for example, the inverse images would result from applying the same f^{-w}. The problem is further complicated if components require different properties. If the system allows the user to specify that one dataflow graph source must be pure and another must be complete, then overlapping components may require I^{-w}s and I^{-v}s that have been produced using different properties. Sharing then becomes more difficult, if not impossible.

9.3 Constant information density

We have identified several interesting areas for further research on constant information density visualizations. We summarize these ideas in this section.

9.3.1 Movement optimization

As users pan and zoom, they may require less detail on the screen than when they are not moving. The user interface should allow the user to differentiate between appropriate detail for movement versus still conditions. For example, the user may require fewer objects while panning and zooming than when viewing a still image. This information can be used to reduce rendering time.

9.3.2 User studies

Further studies of user response to applications with constant information density are plainly warranted. Additionally, although we have focused in this work on preserving constant information density for a given metric rather than comparing density metrics and studying appropriate values for such metrics, such studies would plainly be useful. A formal taxonomy of density metrics would also be of significant interest.

9.3.3 Display and constraint mechanisms

Potential improvements of the mechanism described in Chapter 6 include replacing the grid-based model with an object-based model, adding an allocation mechanism, and developing a graphical mechanism with which users can specify constraints.

Chapter 10

Conclusions

We have presented novel solutions for two key problems in data visualization: data lineage and information density. We have shown how our solutions can be applied in a database visualization environment, significantly improving its usability.

In the first part of the dissertation, we defined and addressed the data lineage problem. In Chapter 2, we discussed a efficient method to support data lineage queries. In Chapter 3, we introduced the general principles of weak inversion and verification and showed how they can be used to reconstruct the (approximate) lineage of derived data without explicit metadata. These techniques eliminate irrelevant source data, thereby reducing clutter in the display.

In the second part of the dissertation, we used the Principle of Constant Information Density to reduce the visual complexity of graphical representations of data. In Chapter 4, we presented a pilot study that indicates that users navigate differently in visualizations with and without constant information density. In Chapter 5, we presented a number of techniques for creating visualizations with constant information density. These techniques are most appropriate to uniformly-distributed data sets. In Chapter 6, we extended our work to non-uniformly-distributed data sets. In Chapter 7, we discussed techniques for semi-automatically modifying the graphical representations of data. Finally, in Chapter 8, we introduced a novel navigation method appropriate to constant information density visualizations.

Bibliography

[1] C. Ahlberg and B. Shneiderman. Visual information seeking: tight coupling of dynamic query filters with starfield displays. In *Proc. ACM SIGCHI '94*, pages 313–317, Boston, April 1994.

[2] A. Aiken, J. Chen, M. Stonebraker, and A. Woodruff. Tioga-2: a direct manipulation database visualization environment. In *Proc. 12th Int. Conf. on Data Engineering*, pages 208–17, New Orleans, Louisiana, February 1996.

[3] R. Avnur, J.M. Hellerstein, B. Lo, C. Olston, B. Raman, V. Raman, T. Roth, and K. Wylie. CONTROL: Continuous Output and Navigation Technology with Refinement On-Line. In *Proc. ACM SIGMOD '98*, pages 567–569, Seattle, Washington, June 1998.

[4] E. Bainto, J. Dozier, J. Frew, J. Gray, R. Mechoso, and S. Miley. Requirements for Sequoia database system. Sequoia 2000 technical memorandum, University of California, Berkeley, California, September 1993.

[5] B.B. Bederson and J.D. Hollan. Pad++: a zooming graphical interface for exploring alternate interface physics. In *Proc. ACM UIST 94*, pages 17–26, Marina del Rey, CA, November 1994.

[6] B.B. Bederson, J.D. Hollan, J. Stewart, D. Rogers, A. Druin, and D. Vick. A zooming web browser. In *SPIE Multimedia Computing and Networking (Proceedings of the SPIE, San Jose, CA, January, 1996)*, volume 2667, pages 260–271, 1996.

[7] J. Bertin. *Semiology of Graphics*. The University of Wisconsin Press, Madison, Wisconsin, 1983. J. Berg, translator.

[8] E. Bier, M. Stone, K. Pier, W. Buxton, and T. DeRose. Toolglass and magic lenses: the see-through interface. In *Proc. ACM SIGGRAPH '93*, pages 73–80, Anaheim, California, August 1993.

[9] P. Brown and M. Stonebraker. BigSur: a system for the management of Earth science data. In *Proc. 21st Int. Conf. on Very Large Data Bases*, pages 720–728, Zürich, Switzerland, September 1995.

[10] B.P. Buttenfield and R.B. McMaster, editors. *Map Generalization: Making Rules for Knowledge Representation*. Longman, London, England, 1991.

[11] H.-T. Chou and D. DeWitt. An evaluation of buffer management strategies for relational database systems. In *Proc. 11th Int. Conf. on Very Large Data Bases*, pages 127–141, Stockholm, Sweden, August 1985.

[12] C. Cleverdon and M. Keen. *Factors Determining the Performance of Indexing Systems*. ASLIB Cranfield Research Project, Cranfield, Bedsford, England, 1966.

[13] P. Denning. Virtual memory. *ACM Computing Surveys*, 2(3):153–189, September 1970.

[14] Federal Geographic Data Committee. Content standards for digital spatial metadata (final draft), June 1994.

[15] K. Fishkin and M.C. Stone. Enhanced dynamic queries via movable filters. In *Proc. ACM SIGCHI '95*, pages 415–420, Denver, May 1995.

[16] A.U. Frank and S. Timpf. Multiple representations for cartographic objects in a multiscale tree - an intelligent graphical zoom. *Computers & Graphics*, 18(6):823–829, November-December 1994.

[17] G.W. Furnas. Generalized fisheye views. In *Proc. ACM SIGCHI '86*, pages 16–23, Boston, 1986.

[18] G.W. Furnas and B.B. Bederson. Space-scale diagrams: understanding multiscale interfaces. In *Proc. ACM SIGCHI '95*, pages 234–241, Denver, May 1995.

[19] W.O. Galitz. *User-Interface Screen Design*. QED Pub. Group, Boston, 1993.

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

[21] M.M. Gorlick and R.R. Razouk. Using weaves for software construction and analysis. In *Proc. 13th International Conference on Software Engineering*, pages 23–34, Austin, Texas, May 1991. IEEE Computer Society Press.

[22] GTE New Media Services. BigBook, 1995-1998. http://bigbook.com.

[23] Y.E. Ioannidis, M. Livny, J. Bao, and E.M. Haber. User-oriented visual layout at multiple granularities. In *Proc. 3rd International Workshop on Advanced Visual Interfaces*, pages 184–193, Gubbio, Italy, May 1996.

[24] D. P. Lanter. Design of a lineage-based meta-data base for GIS. *Cartography and Geographic Information Systems*, 18(4):255–261, 1991.

[25] B. Lucas, G. Abram, N. Collins, D. Epstein, et al. An architecture for a scientific visualization system. In *Proc. 1992 IEEE Visualization Conference*, pages 107–114, Boston, October 1992.

[26] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, April 1986.

[27] E. Mesrobian, R.R. Muntz, J.R. Santos, E.C. Shek, C.R. Mechoso, J.D. Farrara, and P. Stolorz. Extracting spatio-temporal patterns from geoscience datasets. In *Proc. IEEE Workshop on Visualization and Machine Vision*, pages 92–103, Seattle, Washington, June 1994.

[28] National Institute of Standards and Technology. Federal Information Processing Standard (FIPS PUB) 173-1: Spatial Data Transfer Standard (SDTS), 1994.

[29] J.V. Nickerson. *Visual programming.* Ph.D. dissertation, New York University, New York, 1994.

[30] C. Olston, A. Aiken, J. Hellerstein, and M. Stonebraker. VIQING: Visual Interactive QueryING. In *Proc. 14th IEEE Symp. on Visual Languages*, Halifax, Nova Scotia, September 1998. To appear.

[31] B. Paul. Mesa, 1995-1996. ftp://iris.ssec.wisc.edu/pub/Mesa/.

[32] K. Perlin and D. Fox. Pad: an alternative approach to the computer interface. In *Proc. SIGGRAPH '93*, pages 57–64, Anaheim, California, August 1993.

[33] R.J. Phillips and L. Noyes. An investigation of visual clutter in the topographic base of a geological map. *Cartographic Journal*, 19(2):122–132, 1982.

[34] J. Rasure and M. Young. An open environment for image processing software development. In *Proc. 1992 SPIE Symposium on Electronic Image Processing*, pages 300–310, San Jose, California, February 1992.

[35] Regents of the University of California. DataSplash, 1997. http://datasplash.cs.berkeley.edu.

[36] S.F. Roth, J. Kolojejchick, J. Mattis, and J. Goldstein. Interactive graphic design using automatic presentation knowledge. In *Proc. ACM SIGCHI '94*, pages 112–117, Boston, April 1994.

[37] M. Schiff. *Designing graphic presentations from first principles.* Ph.D. dissertation, University of California, Berkeley, Berkeley, CA, 1998.

[38] A. Smith. Cache memories. *ACM Computing Surveys*, 14(3):473–530, September 1982.

[39] C.J. Springer. Retrieval of information from complex alphanumeric displays: screen formatting variables' effects on target identification time. In G. Salvendy, editor, *Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems*, pages 375–382. Elsevier, Amsterdam, the Netherlands, 1987. Published as Proceedings of the Second International Conference on Human-Computer Interaction (August 1987, Honolulu, Hawaii), volume 2.

[40] P. Steenkiste. Advanced register allocation. In P. Lee, editor, *Advanced Language Implementation.* MIT Press, Cambridge, Massachusetts, 1991.

[41] M. Stonebraker, J. Chen, N. Nathan, C. Paxson, and J. Wu. Tioga: providing data management for scientific visualization applications. In *Proc. 19th Int. Conf. on Very Large Data Bases*, pages 25–38, Dublin, Ireland, August 1993.

[42] M. Stonebraker and J. Dozier. Sequoia 2000: large capacity object servers to support global change research. Sequoia 2000 Technical Report 91/1, University of California, Berkeley, CA, March 1992.

[43] M. Stonebraker and G. Kemnitz. The POSTGRES next-generation database management system. *Comm. of the ACM*, 34(10):78–92, October 1991.

[44] Surveys and Resource Mapping Branch, Ministry of Environment, Lands and Parks. Spatial Archive and Interchange Format (SAIF), release 3.1, 1994.

[45] F. Töpfer and W. Pillewizer. The principles of selection, a means of cartographic generalization. *Cartographic Journal*, 3(1):10–16, 1966.

[46] K. Tsui, P. Fletcher, and M. Hutchins. PISTON: a scalable software platform for implementing parallel visualization algorithms. In *Proc. Computer Graphics International*, Melbourne, Australia, June 1994.

[47] E.R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, Connecticut, 1983.

[48] T.S. Tullis. Screen design. In Martin Helander, editor, *Handbook of Human-Computer Interaction*, pages 377–411. Elsevier Science Publishers (North-Holland), Amsterdam, 1988.

[49] C. Upson, T. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. VanDam. The Application Visualization System: a computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):32–40, July 1989.

[50] A. Woodruff, J. Landay, and M. Stonebraker. Constant density visualizations of non-uniform distributions of data. In *Proc. UIST '98*, pages 19–28, November 1998.

[51] A. Woodruff, J. Landay, and M. Stonebraker. Constant information density in zoomable interfaces. In *Proc. Advanced Visual Interfaces '98*, pages 57–65, L'Aquila, Italy, May 1998.

[52] A. Woodruff, J. Landay, and M. Stonebraker. Goal-directed zoom. *SIGCHI '98 Summary*, pages 305–6, April 1998.

[53] A. Woodruff and M. Stonebraker. Buffering of intermediate results in dataflow diagrams. In *Proc. 1995 IEEE Symposium on Visual Languages*, pages 187–94, Darmstadt, Germany, 5-9 Oct. 1995.

[54] A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Proc. 13th Int. Conf. Data Engineering*, pages 91–102, Birmingham, England, April 1997.

[55] A. Woodruff, P. Wisnovsky, C. Taylor, M. Stonebraker, C. Paxson, J. Chen, and A. Aiken. Zooming and tunneling in Tioga: supporting navigation in multidimensional space. In *Proc. 10th IEEE Symposium on Visual Languages*, pages 191–3, St. Louis, Missouri, October 1994.

[56] H. Yamana, J. Kohdate, T. Tasue, and Y. Muraoka. An environment for dataflow program development of parallel processing system-Harray. *Systems and Computers in Japan*, 22(8):26–38, 1991.

[57] T.C. Zhao and Mark Overmars. XForms, 1996. http://bragg.phys.uwm.edu/xforms.