A Hierarchy Navigation Framework: Supporting Scalable Interactive Exploration over Large Databases

Nishant K. Mehta, Elke A. Rundensteiner and Matthew O. Ward
Computer Science Department,
Worcester Polytechnic Institute, Worcester, MA 01609
{nishantm | rundenst | matt}@cs.wpi.edu *

Abstract

Modern computer applications from business decision support to scientific data analysis use visualization techniques. However, visual exploration tools do not scale well for large data sets due to screen clutter. Visualization tools have thus been extended to support hierarchical views of the data, with support for focusing and drilling-down using interactive brushes. We now investigate how best to couple such a near real-time responsive visualization tool with database support. For this, we have developed a tree labeling method, called MinMax tree, that allows the movement of the on-line recursive processing of visual user interactions on hierarchical data sets into an off-line precomputation step. Using MinMax tree we map the recursive processing at the interface level to two dimensional range queries that can be answered efficiently using spatial indexes. We also employ caching and prefetching at the client side to cope with the real-time response requirements. The techniques have been incorporated into XmdvTool, a free software package for multi-variate data visualization and exploration. Our experimental results show 70% to 80% reduction in response time latency even with limited system resources.

1 Introduction

Whether the domain is stock data, scientific data, or the distribution of sales, visualization is becoming a popular technique for data exploration. Visualization tools exploit the fact that humans can detect patterns and trends in the underlying data by just looking at it. However, most existing techniques do not scale well with respect to the size of the data. [10] proposed an approach for displaying and visually exploring large datasets. The idea was to present data at different levels of detail based on clustering the initial data points into a hierarchy called the *cluster tree*. The problem of clutter at the interface level is solved by displaying only one level of detail at a time. Such hierarchical summarizations (*cluster tree*) in fact increase the size of the input by at least one order of magnitude. Hence management of data remains an even more critical issue. While storing the data in main memory and flat files is appropriate for moderate sized sets, this becomes unworkable when scaling to large data sets.

However, visual exploration operations such as the brush-driven navigation of hierarchies are not easily supported by traditional database systems. Furthermore, techniques used for main memory processing are typically not effective if implemented directly in a database system. In particular, the recursive processing involved when exploring hierarchies in main memory is no longer appropriate when storing those hierarchies on disk. Instead, we have developed a hierarchy encoding technique called *MinMax trees*, which allows us to map hierarchies to a 2 dimensional space called *2D Hierarchy Map*. This mapping in turn allows us to represent visual navigation operations as spatial searches. Thus the *2D Hierarchy Map* is stored in a database, where the searches are executed efficiently using spatial indexes.

Also interactive visual exploration tools exhibit a variety of characteristics that can be exploited to make the system scale to huge data sets. These include locality of exploration and data access, predictability of user's exploratory movements, and presence of idle time between user operations. To take advantage of the above characteristics we propose a caching strategy that buffers the recently used data items. The cache also uses a fast look up mechanism using a memory resident spatial index. Moreover, the idle time between user operations can be effectively utilized for predicting and prefetching the data for future user requests. Towards this end, we integrate prefetcher technology into our system.

We applied our proposed solution strategies to the hierarchical navigation tool (*structure-based brush*) in XmdvTool [10]; a freeware software package for visually exploring

*This work is supported under NSF grants IIS-0119276.

multi-variate data sets. However, the proposed techniques are general and can be used for visual exploration of arbitrary hierarchies, a common class of navigation operations in large scale visualization systems [9].

The main contributions of this paper are:

- A hierarchy encoding technique that maps a tree into a 2D space and visual navigation operations into spatial range queries.
- A framework that exploits the encoding technique and characteristics of the visual navigation environment for efficient retrieval of online data. This includes:
  1. A caching strategy to reduce fetch latencies.
  2. Index structures that exploit hierarchy encoding for efficient searches on cache and database.
  3. A direction based prefetching strategy that exploits properties of visual interactive tools to predict future user requests.
- Experimental evaluation quantifies the relative effectiveness of each technique towards latency reduction. Overall, we find that the approach scales to large data sets and even for moderate data sets reduces user response time by 70 to 80 percent.

This paper is organized as follows. Section 2 introduces multi-variate hierarchical visualization. The hierarchy encoding and *MinMax* query processing are presented in Section 3. Section 4 introduces the proposed framework. Section 5 presents the performance study. Section 6 discusses related work, while Section 7 discusses conclusions.

2 Visual Data Exploration

2.1 XmdvTool: The Motivating Application

XmdvTool is a visualization tool designed for exploration and analysis of multivariate data sets, offering four distinct yet interlinked visualization techniques [10, 11, 25].

2.2 Visual Brush-Based Exploration

Brushing is the process of interactively painting over a subregion of the data display using a mouse, stylus, or other input device that enables the specification of location attributes [1]. The location attribute values are then used to select subsets of the data. Brushing can be performed in screen, in data space or in structure space.

Here we focus on *structure space* techniques, i.e., selection based on structural relationships between data points [11]. By structure, we mean that we recursively partition data into related groups and identify suitable summarizations for each cluster. Then we can examine the data set at different levels of abstraction. We move down the hierarchy (*drill-down*) when interesting features appear in the summarizations and up the hierarchy (*roll-up*) after sufficient information has been gleaned from a particular subtree.

Brushing in structure space involves two containment criteria. The leaves of the tree are chained together, i.e., have a total order imposed. Given this order, nodes that fall into a user defined interval satisfy the containment criteria, called 'horizontal selection'. We augment each node in the hierarchy, i.e., each cluster, with a monotonic value that controls the *level-of-detail*. The nodes at the desired *level-of-detail* are selected using so called 'vertical selection'. The *level-of-detail* value can have different semantics; e.g., *width* of the cluster i.e., the number of leaf nodes the cluster encompasses, or the *distance* of cluster from root. In summary, a structure-based brush is defined by a subrange of the *structure extents* and *level-of-detail* values.

2.3 Structure-Based Brushing in XmdvTool

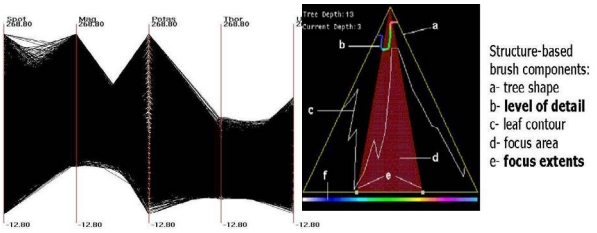

Figure 1. Cluttered Parallel Coordinates.

Figure 2. Structure-based brush in XmdvTool.

Figure 1 shows a parallel coordinates display of a five dimensional data set having 16,384 records. In this display each of the N dimensions is represented by a vertical axis. A data point in N-dimensional space is mapped to a polyline that traverses across all N axes; crossing each axis, at a position proportional to its value for that dimension. As seen from Figure 1, displaying all the data to the user at the same time results in display clutter.

To support visual navigation of cluster trees for large data sets, XmdvTool contains a structure-based brush (Figure 2). The triangular frame depicts the hierarchical tree. The contour near the bottom of the tree delineates the approximate shape formed by chaining together the leaf nodes. To navigate the hierarchy the tool provides two main "sliders". The *level-of-detail* slider denoted by '*b*' allows users to navigate the tree vertically and view clusters at different levels of detail. The *focus extents* slider denoted by

'e' allows users to move horizontally and focus on a subset of clusters within the same level. The left and right extents of the 'e' slider can also be adjusted individually to modify the width of the focus area. Figure 3 displays the same data set as Figure 1 but focused on a specific cluster of data points. This is after the user narrows the width of the focus area using 'e' and performs a drill-down operation using 'b' as reflected in Figure 3. Figure 4 displays the same data set as Figure 3 but showing mean and the range of the data points in that cluster. This is after the user performs a roll-up operation using 'b' as seen in Figure 4.

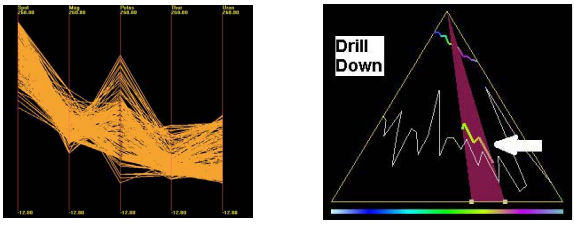

Figure 3. After Focused Area Drilled-Down: (a) Display, (B) Brush.

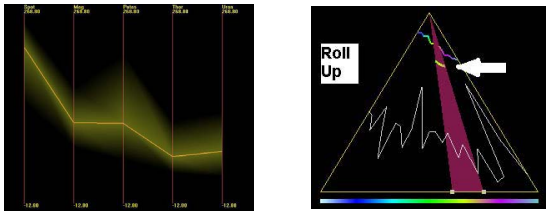

Figure 4. After Focused Area Roll-Up: (a) Display, (B) Brush.

2.4 Brush Semantics

A structure-based brush is defined as the intersection of two independent selections, the horizontal extents of the brush e_1 and e_2 and $level - of - detail$. In horizontal selection, first a set of leaf nodes is selected based on the order property "selecting all leaves between two values e_1 and e_2". This selection is propagated up towards the root based on either ANY or ALL semantic, i.e., to select nodes that have ANY (or ALL) of its children already selected.

For vertical selection we use the *level-of-detail* value that has been associated with each node in the hierarchy. This can be any montonically increasing or decreasing value from the root towards the leaves. The Algorithm 1 explains the process of vertical selection. We assume that *level-of-detail* values are monotonically decreasing from the root towards the leaves. The function $lod(x)$ returns *level-of-detail* of node x. At end, W contains the nodes that satisfy the vertical selection criteria. The set of nodes that satisfy both the selection criteria forms the final set of nodes in the brush.

Algorithm 1 Vertical Selection

```
1:  Let S and W be two sets of nodes.
2:  Let S contain only root node and W be empty.
3:  while S is not empty do
4:     Remove node n from S
5:     if lod(n) ≤ lod(brush) then
6:        Insert n into W
7:     else
8:        Insert descendants of n into S
9:     end if
10: end while
```

The brush operations, as described above, are inherently recursive. Thus, in Section 3 we develop equivalent but non-recursive computation methods for setting structure-based brushes based on assigning precomputed values to the nodes that recast retrievals as range queries.

3 MinMax Trees: Translating Navigation Operations

3.1 Labeling the Nodes

To map the recursive process of selection to a non-recursive one we augment each node in the cluster tree with horizontal and vertical extents. Each of these extents forms an interval. We call this tree a MinMax tree. A MinMax tree is a n-ary tree. The horizontal extents of the nodes correspond to open intervals defined over a totally ordered set, called an *initial set*. The horizontal extents of the leaf nodes in the tree form a sequence of non-overlapping intervals. The non-leaf nodes are unions of intervals corresponding to their children.

It is always possible to draw the tree such that all leaf nodes are horizontally ordered. Leaf nodes are then labeled with pairs of values corresponding to the extents of their interval. The intervals of non-leaf nodes are unions of their children intervals. A non-leaf node will be labeled with the minimum extent of its first interval and the maximum extent of its last interval. A node n having two children with intervals $c_1 = (\alpha, \beta)$ and $c_2 = (\gamma, \delta)$ such that $\alpha < \gamma$, will be labeled as $n = (\alpha, \delta)$. Figure 5 gives an example of a labeled cluster tree. For the tree in Figure 5 the process of assigning the horizontal extents started at the leaf nodes. The interval between 0 to 1 was divided equally between all leaf nodes. These intervals were propagated up towards the root. (See Figure 5).

Given a MinMax tree T and two nodes x and y of T whose horizontal extent values are (x_1, x_2) and (y_1, y_2) re-

spectively, node x is an ancestor of node y if and only if its horizontal extents $x_1 \leq y_1$ and $x_2 \geq y_2$. The containment property is based on the intuition that each node in the tree is included in its parent's interval.

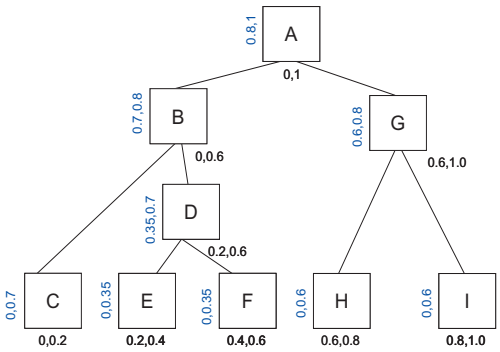

Figure 5. Labeled Minmax Tree

The horizontal extents of the brush and each node in the MinMax tree both define a horizontal interval. Using MinMax trees we can reduce this process of selecting nodes in the ANY brush to searching for nodes in the MinMax tree whose interval intersects with the interval of the brush. Similarly, the process for the ALL brush maps to selecting nodes whose intervals are fully contained in the brush interval.

For selection of the *level-of-detail* (vertical selection), the MinMax tree augments each node in the cluster tree with a vertical extent value. Given the brush semantics in Section 2.4, the vertical extent of a node A is the interval (v_1, v_2) where $(v_1 = lod(A), v_2 = lod(parent(A))$ where the function $parent(n)$ returns the parent node of node n. The node n with vertical extents (v_1, v_2) lies in the brush if $v_1 \leq lod(brush) < v_2$ is true. The show the vertical extents for each node in cluster tree.

Essentially, the process of labeling the nodes is a recursive one. The intervals are computed and assigned off-line at the time the hierarchy is created.

3.2 2-D Hierarchy Maps

Note that the labels assigned by the MinMax procedure can be viewed as giving each node a spatial representation. The complete cluster tree can thus be mapped to a 2 dimensional space, what we call the *2-D hierarchy map*. Figure 6 shows a *2-D hierarchy map* for the MinMax tree in Figure 5. All the leaf nodes are shaded in grey. A node n with horizontal extents (h_{min}, h_{max}) and vertical extents (v_{min}, v_{max}) maps to a rectangular region in the *2-D hierarchy map* with the bottom left corner at (h_{min}, v_{min}) and the upper right corner at (h_{max}, v_{max}).

The 2-D hierarchy map exhibits the following:

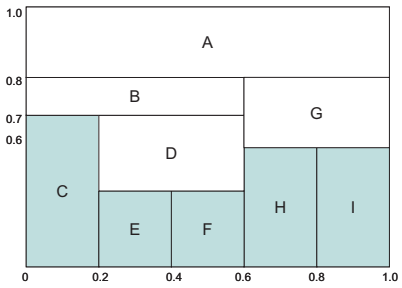

Figure 6. 2-D Hierarchy Map of MinMax Tree in Figure 5.

- The space between $(0, 0)$ to $(1, 1)$ is **completely filled**, i.e., given any point $(0, 0)$ and $(1, 1)$ there exists a node that contains the point.
- Interiors of **no** two nodes **overlap** in 2-D hierarchy map.

3.3 Using 2-D Hierarchy Maps to Implement Structure-Based Brushes

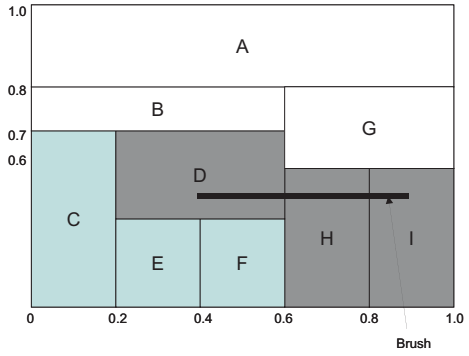

Figure 7. 2-D Brush Selection with b_{min}=0.4, b_{max}=0.9 and lod=0.35

From the 2-D hierarchy map, we can implement ANY and ALL structure-based brushes as non-recursive operations. The containment criteria for the ANY structure-based brush can be defined as follows. Given the brush's horizontal extents (b_{min},b_{max}) and the *level-of-detail*=lod, any node n having horizontal extents (h_{min}, h_{max}) and vertical extents (v_{min}, v_{max}) lies in the brush iff:

- The extents (h_{min}, h_{max}) intersect the brush interval (b_{min}, b_{max}), and

- $v_{min} < lod \leq v_{max}$.

A node n lies in the ALL structure-based brush iff:

- $(h_{min}, h_{max}) \cap (b_{min}, b_{max}) = (h_{min}, h_{max})$, and
- $v_{min} < lod \leq v_{max}$.

This definition can also be stated differently. If we map the brush to a line segment with end points at (b_{min}, lod) and (b_{max}, lod) in the 2-D hierarchy map, a node n lies in the ANY (ALL) structure-based brush if its representation in the 2-D hierarchy map intersects with that of the brush (brush segment intersects both the right and left edge of the node). Our reformulation maps the process of searching for nodes in ALL and ANY brush into spatial queries.

Figure 7 gives an example of the selection for ANY brush where b_{min}=0.4, b_{max}=0.9 and lod=0.35. Figure 7 shows the brush in black and all selected nodes (i.e., active set) in dark grey.

3.4 Translating Structure-Based Brushes into SQL

The 2-D hierarchy map technique reduces the containment criterion from initially recursive semantics to an inclusion test in the horizontal and vertical direction. One scan of the hierarchy is hence sufficient to form the selection.

Let H be the relational table that stores the nodes in the hierarchy. Each tuple in H models one node in the cluster tree and has horizontal and vertical extents of the node, besides the node information. We have:

H (e_{min},e_{max},v_{min},v_{max}, ...)

An ANY **structure-based brush** having horizontal extents (b_{min}, b_{max}) and *level-of-detail* (lod) can be expressed as a range query as follows.

select * from H
where $e_{min} \leq b_{max}$ and $e_{max} \geq b_{min}$
and $v_{min} \leq lod$ and $v_{max} > lod$

An ALL structure-based brush query for the same parameters is specified by:

select * from H
where $e_{min} \geq b_{min}$ and $e_{max} \leq b_{max}$
and $v_{min} \leq lod$ and $v_{max} > lod$

This range query requires only a linear processing time for computing brushing results.

4 XmdvTool Backend Framework: Indexing, Caching and Prefetching

We now describe the components in our framework (Figure 8). The cache is used to buffer the recently used data items. The prefetcher predicts user requests and fetches data into the cache. For each user request the cache is searched to find the requested objects. The cache may contain all the requested nodes, or only a subset. In the latter case the delta calculator computes a remainder query to fetch the subset of nodes not in the cache. The loader fetches the result of the remainder query into the cache. Once all the requested nodes are in the cache they are delivered to the front-end.

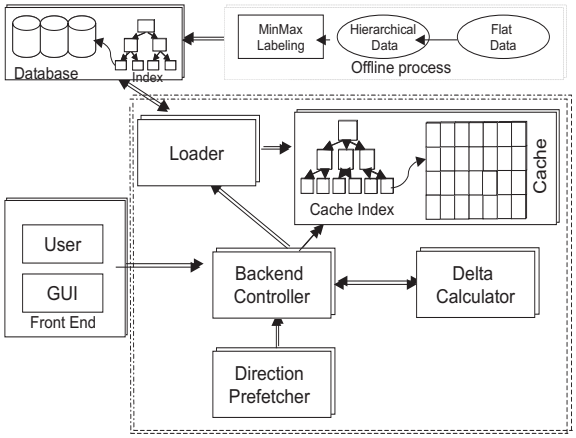

Figure 8. XmdvTool Backend.

4.1 Spatial Index

For each request from the front end we need to quickly search the contents of the cache, compute the difference query and fetch the data from the database. We thus need a fast search mechanism both for the cache and the database. The front-end passes a query q to the back-end to search for objects that lie within the brush. This brush maps to a segment in the *2-D hierarchy map* (Section 3.2). The answer is a set of clusters that intersect this segment. Therefore the query q is a 2 dimensional spatial range query (Section 3.4). To execute this query efficiently we thus propose to use a spatial index.

A spatial index utilizes spatial relationships to organize data entries with each key value seen as a point or a region in a k-dimensional space. Many spatial index structures have been proposed, each of which has its pros and cons. For our purpose we require a spatial index that works for spatial range queries and supports high update rates because the contents of the cache are continuously changing.

In our current implementation we use an R-Tree index [12]. It is a simple multi-dimensional index structure, while its performance is comparable to the more complex index structures available. To support fast insertions we use the *linear spit* method [12] when splitting nodes.

4.2 Delta Calculator

Each time the front end submits a query q, the backend searches the cache to find all the objects that lie in the brush. Given this list the backend computes the remainder query (q_Δ) to fetch the objects not in the cache.

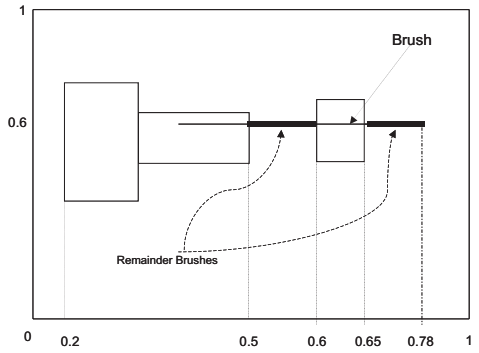

Figure 9. Computing Δ queries. The line segment represents the brush. ($b_{min} = 0.3, b_{max} = 0.78, lod = 0.6$)

Let $\{b_1\}$ represent the set of nodes contained in brush b_1 having horizontal extents (b_{min}, b_{max}) and $level - of - detail = lod$. A node n with horizontal extents (α, β) in the brush b_1 can be used to divide the brush into two disjoint brushes b_2 and b_3 such that the $\{b_2\} \cup n \cup \{b_3\} = \{b_1\}$, where b_2 has horizontal extents (b_{min}, α), $level - of - detail = lod$ and b_3 has horizontal extents (β, b_{max}), $level - of - detail = lod$. This property is based on the intuition that the horizontal extents of the brush and the node n both define an interval. Therefore we can divide the interval of brush b_1 into two disjoint intervals such that the result of the union of these two intervals b_2 and b_3 with interval of n gives us the interval of brush b_1. Thus, to compute the remainder query we need to find the nodes in the cache that belong to the current brush. We can then check to see what parts of the brush interval are not occupied by the nodes in the cache. Each of these unoccupied intervals forms a remainder brush and a part of q_Δ.

Figure 9 gives an example of above for the brush $(b_{min} = 0.3, b_{max} = 0.78, lod = 0.6)$. The figure shows the 2D hierarchy map of the contents of the cache. The bold part of the brush illustrates the remainder brushes.

4.3 Cache

Using main memory (cache) to store frequently used data items to reduce fetch latencies from secondary storage devices is a proven technique. We employ caching to reduce latency and to avoid database fetches. The cache in XmdvTool is a contiguous chunk of main memory. Each cache entry contains a cluster (node) from the cluster tree and a descriptor that describes the position of the node in the 2-D hierarchy map (Section 3.2). Analysis of real user traces of our visualization environment done in [8] has shown that user traces exhibit characteristics like:

1. Locality of Exploration: Users doing data exploration explore one area of display at a time before moving to another area.

2. Contiguous queries have similar answers: Exploration using visual navigation tools such as sliders and knobs translate to consecutive queries and the answers to these queries have a significant number of objects that are common.

3. Presence of idle time: Users usually pause to understand the display and look for patterns in the data, so there is idle time between queries to the database.

Note the properties (1) and (2) correspond to the concept of spatial and temporal locality respectively. For program reference streams that exhibit similar characteristics the LRU [23] replacement policy performs very close to the optimal replacement policy. We thus implement the LRU replacement policy.

4.4 Direction-based Prefetching

To further improve the performance of subsequent user operations, XmdvTool incorporates prefetching technology [19, 20]. The prefetcher exploits the third user trace characteristic given in Section 4.3, namely, the idle time in between user operations, to predict and fetch future user requests. In particular, our current study focusses on direction-based style prefetchers; which are analogous to sequential prefetching proposed in other prefetching works [5, 18, 8]. The strategy predicts the most likely direction of the next brush movement. For interactive navigation tools like sliders and knobs it is intuitive that the user will continue to use the same navigation tool for a while before changing to another one. Based on the user's past explorations, the predictor assigns probabilities to the four directions. The strategy then is to prefetch data in the direction with highest probability.

5 Experimental Results

5.1 Experimental Setup and Metrics

All of our experiments were run on a Pentium 3 windows XP machine with 128 MB of memory. The complete system was implemented in C++. We used the OTL oracle-odbc template library to access data on an oracle server running Oracle 9i. We used the oracle spatial extension to construct the R-Tree index at the database. For the main memory R-Tree index we used the spatial index library developed at University of California Riverside.

To test the scalability of the system we used two real data sets obtained from online repositories. Data set D1 had 20,000 data points we call it the out5d data set. It is a five dimensional remote sensed data (SPOT, magnetics, and three radiometrics channels - potassium, thorium, and uranium). Data set D2 had 195,000 data points and 6 dimensions. It contains flow simulation data. We ran experiments over data set D1 with a set of real user traces. We used a set of four real user traces each of half hour duration, collected as a part of the study performed in [8]. For experiments over data set D2 we used a set of four synthetic user traces. However, these traces were modeled based on the characteristics exhibited by the real user traces. [8] presents details of modeling user traces. All the results reported in this section are an average taken over four runs.

The main metric used to evaluate the performance of the cache is *latency*. The *latency* for a single user request is the time taken for the backend to serve the data once the request is submitted. To compute the latency for a complete user trace, we use the following formula:

$$latency = \frac{\sum_{i=1}^{N} L_i}{\sum_{i=1}^{N} T_i} \quad (1)$$

Where N is the total number of requests, T_i is the number of objects (tuples) fetched in request i and L_i is the latency for request i. Equation 1 gives us the latency per object fetched. It gives us a common ground to compare and combine the latency measures for different user traces. A measure derived from latency is the *Latency Reduction Ratio* (*lrr*). The *latency reduction ratio* for a particular system configuration is the ratio of the decrease in latency to the latency obtained when running the same experiment using the base configuration. In the base configuration the cache, prefetcher and the secondary index structure all are turned off so the user requests are sent directly to the database.

$$lrr = \frac{Latency_{base} - Latency_{output}}{Latency_{base}} \quad (2)$$

Equation 2 gives us the fraction of the latency reduced by a particular system configuration. This helps to evaluate the relative usefulness of each configuration. In addition we also use *object hit ratio* evaluated using Equation 3 to measure the usefulness of the prefetecher.

$$hitratio = \frac{TotalNumberOfObjectHits}{TotalNumberOfObjectsRequested.} \quad (3)$$

5.2 Database Index

To show the usefulness of the R-Tree index structure in our context, we ran two experiments. In the first experiment we ran four user traces over data sets D1 and D2. The cache was turned off, so each user request was sent directly to the database. We record the latency for each user trace with database index on and off. Database index off implies that system configuration is same as base configuration. Figures 10 and 11 show the latency and also *lrr* for data sets D1 and D2 respectively. For data set D1 the latency reduction ratio on average is approximately 33%, while for D2 it is 70%. Search time for sequential scan increases linearly relative to the size of the data set, whereas with an index the search time increases almost logarithmically with the data set size. As the size of the data set increases, the benefits of the database index become more significant.

	No-Index	Index	lrr
User1	2.0253	1.5	0.25
User2	0.6444	0.35	0.45
User3	0.751336	0.5	0.33
User4	0.8855	0.6	0.32

Figure 10. Latency in msec, Data Set D1

	No-Index	Index	lrr
User1	3.90	0.94	0.75
User2	6.09	1.75	0.71
User3	2.05	0.71	0.65
User4	5.42	1.09	0.80

Figure 11. Latency in msec, Data Set D2

In the second experiment we measure the effectiveness of the index with the cache turned on. The cache size set to 10% of the size of the data set for D1 and to 2% for D2. Note that D2 is around 10 times larger than D1. Figures 12 and 13 show the *l*atency reduction ratio (llr) for four user traces using data sets D1 and D2 respectively. Δ *lrr* is significant, illustrating that the database index is beneficial. On average for D1 we gain approximately 14% and for D2 we gain approximately 17.25%. As the size of the data set increases Δ lrr increases. difference in the amount of time it takes to search for remainder queries for the two approaches

will increase as the size of the data set increases. This shows that the database index makes our system scalable.

	lrr no-index	lrr index	Δlrr
User1	0.50	0.68	0.18
User2	0.52	0.58	0.05
User3	0.25	0.49	0.23
User4	0.36	0.48	0.11

Figure 12. Latency Reduction Ratio (llr) with Relative Cache Size 10 % for Data Set D1

	lrr no-index	lrr index	Δlrr
User1	0.73	0.91	0.18
User2	0.74	0.92	0.18
User3	0.70	0.90	0.20
User4	0.79	0.92	0.13

Figure 13. Latency Reduction Ratio with Relative Cache Size 2% for Data Set D2

5.3 Cache Size

Here we show the effect of the cache size on the *lrr* for data sets D1 and D2. For this experiment we turn on the cache and the index structure in the database.

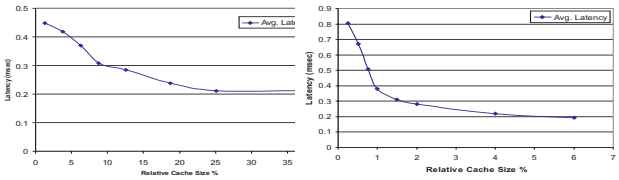

Figure 14. Comparison of Cache size vs. Latency. (a) Data D1, (b) Data D2.

We ran the same 4 user traces. We plot the *a*verage latency vs. relative cache size in Figure 14. As seen in the results, the *l*atency decreases at a high rate for smaller cache sizes. However, the curve flattens out and we get less gains for bigger cache sizes.

Latency is inversely correlated with hit ratio. For the user traces used in this experiment it appears that for big cache sizes the only misses we get are *compulsory misses*. *Compulsory misses* occur when the cluster is accessed for the first time by the user trace. *Compulsory misses* are independent of the cache size, at least until the prefetcher is inactive. Thus we can see that increasing the relative cache size from 20 percent to 40 percent for data set D1 results only in little improvement in latency. The curve flattens out much earlier for data set D2 when compared to data set D1. This demonstrates the size of the cache does not have to scale with the size of the data set. This property makes the system scalable for large data sets.

5.4 R-Tree Cache Index

To show the effectiveness of the R-Tree main memory index over the cache, we ran experiments with the R-Tree main memory index structure turned on and also off. When the R-Tree main memory index is turned off we use sequential scanning to find the requested objects.

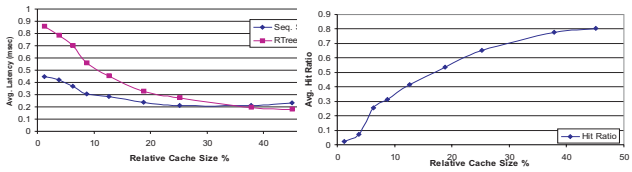

Figure 15. Comparison of Cache Size for Data set D1. With (a) Latency and (b) Hit Ratio.

Figure 15 shows the average latency for the 4 user traces for Data Set D1. The cache index performs worse than the sequential scan for small cache sizes. The main reason being, for small cache sizes the hit ratio is low. This means that cache contents are changing frequently. Thus, the R-Tree index has to be updated very frequently. However, for bigger cache sizes the R-Tree curve does crossover and gives lower latency values. Figure 15 also shows the hit ratio for the same experiment. Note there is only one curve because the hit ratio for both the R-Tree index and sequential scan is the same, the only difference really is in the cache look up latency. If we compare the charts in Figure 14 we see that around the 80% hit ratio mark the R-Tree starts performing better than sequential scan. We also get similar results for data set D2. Thus, we see that the index structure can be helpful in certain situations. However, to make the index work for smaller cache sizes and lower hit ratios we may develop an index like the LR-Tree [2] that supports not only high query rates but also high update rates.

5.5 Prefetcher

Now we set out to show the effectiveness of the directional prefetcher. Our experiments are based on the 4 user traces over D1 and D2, with and without the prefetcher.

Figure 16 shows the *lrr* with and without the prefetcher for user3. Here we can see that the prefetch curve is above

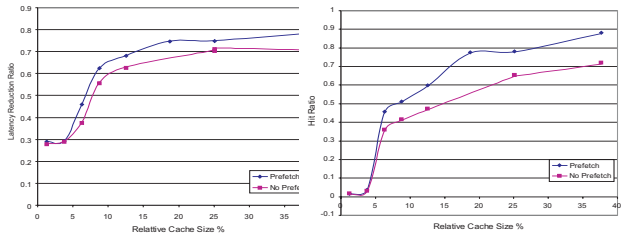

Figure 16. Prefetch vs. Not, User3, Data set D1: (a) *lrr* and (b) Hit Ratio.

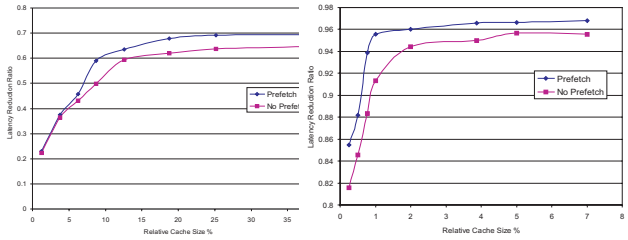

Figure 17. Average *lrr* for 4 user traces, Prefetch vs. Not; Data Sets (a) D1; (b) D2.

the no prefetch curve for practically all cache sizes. This shows that the prefetcher generally improves the performance of the cache (in our case, in particular, by around 8% on average). This can be directly attributed to the improvement in hit ratio.

Figure 16 shows the hit ratio for the same user trace with and without the prefetcher. The curves for the remaining user traces also show the same trend. Figure 17 shows the average *lrr* for the four user traces for data sets D1 and D2. We can see that enabling the prefetcher increases the average *lrr* significantly also as the input data set gets larger the increase in the *lrr* gets larger.

5.6 Discussion

The experiments have demonstrated that each component of the framework plays a significant part in reducing the response time of the system. Also each of the methods scales, i.e., performs relatively better with bigger data sets. Thus our proposed framework has indeed the critical components for serving visualization applications. Due to this success, our framework is scheduled to be released with the next version of the freeware XmdvTool.

6 Related Work

Visualization-database integration. Database support systems for visualization systems such as Tioga [22], USD [14], GODVIA [17] and IDEA [21] represent work closely related to ours in the sense that all of them work towards making visualization systems run over large persistent data. However, many of the detailed techniques used are rather different in each of these systems. USD [14] proposes a semantic net model to store and retrieve unstructured data. I Tioga [22] implements a multiple browser architecture for visual queries. The problem of query translation is not studied. IDEA [21] is an integrated set of tools to support interactive data analysis and exploration. The issues of online query translation and memory management are not addressed.

Other systems that have a visual interface and a database back-end include dynamic query histograms [6] and direct manipulation histograms [13]. However, the operations translate differently: to dynamic range queries in [6] and to temporal queries in [13]. Neither deals with hierarchy exploration support, nor with caching or prefetching.

Special techniques for hierarchy encoding. Our approach towards hierarchy labeling is related to the nested interval method [3, 26]. The labels assigned to the node by the nested interval approach are similar to the horizontal extents. We augment this labeling scheme with labels for vertical extents to incorporate the vertical selection brush semantics. Our work goes beyond hierarchy encoding. Ciaccia et al. [4] used the mathematical properties of *simple continued fractions* for encoding tree hierarchies. However, given a node n, this method cannot efficiently provide the list of descendants of n. Teuhola [24] used a so called *signature* for encoding the ancestor path. Given a node n, the code of n is obtained by applying a hash function to it and by concatenating the resulting value with the code of its parent. The non-unique code can make the quantity of data retrieved be much larger than needed. Moreover, the code obtained by the concatenation of all ancestor codes could exceed the available precision for deep trees.

Recently, there has been considerable work in the area of hierarchy representation for XML. [16] extends the idea proposed in Dietzs numbering scheme [7] to support dynamic insertions with recomputation of labels. In our case the data is assumed to be static. The main idea in [7] was to use tree traversal order to determine the ancestor-descendant relationship between any pair of tree nodes. Each node is labeled with a pair of preorder and postorder numbers. This scheme does not incorporate the vertical selection semantics of the ALL and the ANY brush. Moreover, unlike the labels assigned by our scheme, the labels assigned to the nodes do not easily map to a 2D space. Thus the structure-based brush selection cannot be expressed as simple spatial intersection operations as in our framework. Similarly, [15] compares two main approaches for hierarchy encoding namely nested interval method and the prefix based encoding scheme.

7 Conclusions

With the increasing amount of data being accumulated nowadays, the need for visually exploring large datasets becomes imperative. A viable way to achieve scalability for visualization tools is to integrate them with database systems. Such integrations raise two problems: query translation (i.e., how do visualization operations map to queries in the database?) and memory management (i.e., how can we manage the memory to achieve fast response times?). This paper presents a solution that addresses both these aspects. The approach involves mapping the cluster hierarchy into a spatial representation namely *2-D Hierarchy Maps*. This allows us to translate visual brushing operations into spatial queries. These spatial queries can then be efficiently executed using spatial index structures such as R-Trees. The solution also develops a caching strategy and uses a prefetching policy to improve upon the response time of the system. To show the effectiveness of the methodology we implemented our idea in a freeware visualization tool for data exploration called Xmdv. In the process we coupled Xmdv with an Oracle 9i database management system. Experiments for assessing the method quantify the effectiveness of each component in our system.

References

[1] A. Becker and S. Cleveland. Brushing scatterplots. *Technometrics, Vol 29(2), p. 127-142*, 1987.

[2] P. Bozanis, A. Nanopoulos, and Y. Manolopoulos. Lr-tree: a logarithmic decomposable spatial index method. *British Computer Society*, pages 319–331, May 2003.

[3] J. Celko. *Joe Celko's Trees and Hierarchies in SQL for Smarties*. Morgan Kaufmann, May 2004.

[4] P. Ciaccia, D. Maio, and P. Tiberio. A method for hierarchy processing in relational systems. *Info. Systems*, 14(2):93–105, 1989.

[5] F. Dahlgren, M. Dubois, and P. Stenström. Fixed and Adaptive Sequential Prefetching in Shared Memory Multiprocessors. In *1993 Int. Conf. on Parallel Processing, 1993*, volume 1, pages 56–63, 1993.

[6] M. Derthick, J. Harrison, A. Moore, and S. Roth. Efficient multi-object dynamic query histograms. *Proc. of Info. Visualization*, pages 58–64, Oct. 1999.

[7] P. F. Dietz. Maintaining order in a linked list. In *Proc. of ACM Symposium on Theory of Computing*, pages 122–127. ACM Press, 1982.

[8] P. R. Doshi, E. A. Rundensteiner, and M. O. Ward. A strategy selection framework for adaptive prefetching in data visualization. *15th Int. Conf. on Scientific and Statistical Database Management*, pages 107–116, 2003.

[9] Y. Fua, M. Ward, and E. Rundensteiner. Structure-based brushes: A mechanism for navigating hierarchically organized data and information spaces. *IEEE Trans. on Visualization and Computer Graphics*, 6(2): 150 -159.

[10] Y. H. Fua, M. O. Ward, and E. A. Rundensteiner. Hierarchical parallel coordinates for exploration of large datasets. *IEEE Proc. of Visualization*, pages 43–50, Oct. 1999.

[11] Y. H. Fua, M. O. Ward, and E. A. Rundensteiner. Navigating hierarchies with structure-based brushes. *Proc. of Information Visualization*, pages 58–64, Oct. 1999.

[12] A. Guttman. R-trees: a dynamic index structure for spatial searching. *ACM Int. Conf. on Management of data*, pages 47–57, 1984.

[13] S. Hibino and E. A. Rundensteiner. Processing incremental multidimensional range queries in a direct manipulation visual query. In *Proc of Int. Conf. on Data Engineering, Orlando, Florida, USA*, pages 458–465, 1998.

[14] R. R. Johnson, M. Goldner, M. Lee, K. McKay, R. Shectman, and J. Woodruff. Usd - a database management system for scientific research. *ACM SIGMOD Conf*, pages 4–4, 1992.

[15] H. Kaplan, T. Milo, and R. Shabo. A comparison of labeling schemes for ancestor queries. In *Proc. of ACM-SIAM Symposium on Discrete Algorithms*, pages 954–963. 2002.

[16] Q. Li and B. Moon. Indexing and querying xml data for regular path expressions. In *Proc. of Int. Conf. on Very Large Data Bases*, pages 361–370. 2001.

[17] X. Ma, M. Winslett, J. Norris, X. Jiao, and R. Fiedler. Godiva: Lightweight data management for scientific visualization. *20th Int. Conf. on Data Eng.*, pages 732–744, 2004.

[18] S. Manoharan and C. R. Yavasani. Experiments with sequential prefetching. *Lecture Notes in Computer Science*, 2110:322–331, 2001.

[19] M. Palmer and S. B. Zdonik. Fido: A cache that learns to fetch. In *Proc. of Int. Conf. on Very Large Data Bases*, pages 255–264. 1991.

[20] N. Polyzotis and Y. Ioannidis. Speculative query processing. *Conf. on Innovative Data Systems Research*, 2003.

[21] P. G. Selfridge, D. Srivastava, and L. O. Wilson. Idea: Interactive data exploration and analysis. In *Proc of ACM Int. Conf. on Management of Data, 1996*, pages 24–34. 1996.

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

[23] A. S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, 1992.

[24] J. Teuhola. Path signatures: A way to speed up recursion in relational databases. *IEEE Trans. on Knowledge and Data Engineering*, 8(3):446–454, June 1996.

[25] J. Yang, M. Ward, and E. Rundensteiner. Interactive hierarchical displays: A general framework for visualization and exploration of large multivariate data sets. *Computers & Graphics*, 27(2):265–283, 2003.

[26] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, and G. M. Lohman. On supporting containment queries in relational database management systems. In *SIGMOD Conference*, 2001.