Scalable Visual Hierarchy Exploration*

Ionel D. Stroe, Elke A. Rundensteiner and Matthew O. Ward

Worcester Polytechnic Institute, Worcester MA 01609, USA
daniel|rundenst|matt@cs.wpi.edu

Abstract. Modern computer applications, from business decision support to scientific data analysis, utilize visualization techniques to support exploratory activities. However, most existing visual exploration tools do not scale well for large data sets, i.e., the level of cluttering on the screen is typically unacceptable and the performance is poor. To solve the cluttered interface problem, visualization tools have recently been extended to support hierarchical views of the data, with support for focusing and drilling-down using interactive brushes. To solve the scalability problem, we now investigate how best to couple such a near real-time responsive visualization tool with a database management system. This integration must be done carefully, since the direct implementation of the visual user interactions on hierarchical datasets corresponds to recursive query processing and thus is highly inefficient. For this problem, we have developed a tree labeling method, called MinMax tree, that allows the movement of the on-line recursive processing into an off-line precomputation step. Thus at run time, the recursive processing operations translate into linear cost range queries. Secondly, we employ a main memory access strategy to support incremental loading of data into the main memory. The techniques have been incorporated into XmdvTool, a visual exploration tool, to achieve scalability. Lastly, we report experimental results that illustrate the impact of the proposed techniques on the system's overall performance.

Keywords: Hierarchical Structures, Visual Exploration, Recursive Queries, Memory Management, Database Back-end.

1 Introduction

Whether the domain is stock data, scientific data, or the distribution of sales, visualization plays an important role in the analysis. Visualization tools exploit the fact that humans can detect patterns and trends in the underlying data by just looking at it, *without* being aware in advance about what pattern they'll face. Human perception is greatly influenced by the way information is presented. Thus, various techniques for displaying data have been proposed, each of which emphasizes different of its characteristics. However, most of these techniques do not scale well with respect to the size of the data. As a generalization, [7]

* This work is supported under NSF grant IIS-9732897 and NSF CISE Instrumentation grant IRIS 97-29878.

postulates that any method that displays a single entity per data point invariably results in overlapped elements and a convoluted display that is not suited for the visualization of large datasets.

A new approach has been proposed recently for displaying and visual exploring large datasets [6]. The idea is to present data at different levels of detail based on hierarchical clustering the initial datapoints. The problem of cluttering at the interface level is solved by displaying one level of detail only at a time. However, such hierarchical summarizations result in increasing the size of the input by at least one order of magnitude, making the management of data an issue. While storing the data in main memory and flat files is appropriate for small and moderate sized sets, it becomes unworkable when scaling to large sets.

One solution to this is to integrate visualization tools with back-end DBMSs. The integration cannot be performed *blindly* though. Techniques used for main memory processing are typically not efficient if implemented directly in a database environment. A well known example is sorting. Internal sorting strategies differ significantly from external sorting ones. Another example is presented in this work. The recursive processing involved when navigating through hierarchies in main memory is no longer appropriate when storing those hierarchies on the disk. Instead, we propose using a technique called *MinMax trees* [16] that transforms the recursive processing into a set of fast range queries.

Besides efficiently implementing the visual exploration operations such as zooming and brushing in SQL, we also had to address the problem of caching and managing the results of previous database requests such that subsequent visual operations are as efficient as possible. Furthermore, we applied our proposed solution strategies when coupling the XmdvTool visualization tool [19] with an Oracle DBMS. We also report on results from our performance study of the system.

2 Hierarchical Multivariate Data Visualization

This work was triggered by our goal of adding database support to XmdvTool, a software package (http://davis.wpi.edu/~xmdv) designed for the exploration of multivariate data. The tool provides four distinct visualization techniques (scatterplot matrices, parallel coordinates, glyphs and dimensional stacking) with interactive selections and linked views. Our recent efforts [6, 7] have produced versions of these techniques that allows multi-resolution data presentation.

2.1 Visual Brush-Based Exploration

Selection is a process whereby a subset of entities on a display is isolated for further manipulation, such as highlighting, deleting, or analysis [20]. *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, 19].

Brushing can be performed in screen or data space to specify a *containment criterion*, i.e., whether a particular point is inside or outside the brush. In *screen space* techniques, a brush is specified by a 2-D contiguous subspace on the screen. In *data space* techniques, a specification consists of either an enumeration of the data elements contained within the brush or the N-D boundaries of a hyper-box that encapsulates the selection.

A third category, namely *structure space* techniques, that allows selection based on structural relationships between data points has recently been introduced in [7]. The *structure* of a data set specifies relationships between data points. This structure may be explicit (e.g., categorical groupings or time-based orderings) or implicit (e.g., resulting from analytic clustering or partitioning algorithms).

A *tree* is one convenient mechanism for organizing large data sets. By recursively partitioning data into related groups and identifying suitable summarizations for each cluster, we can examine the data set methodically at different levels of abstraction, moving 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.

As described earlier, brushing requires some containment criteria. For our first containment criterion, we augment each node in the hierarchy, that is each cluster, with a monotonic value that controls the level-of-detail. The second is based on the fact that each node in a tree has extents, denoted by the left- and right-most leaf originating from the node. A structure-based brush is thus defined by a subrange of the structure extents and level-of-detail values. Intuitively, if looking at a tree structure from the point-of-view of its root node (Fig. 1), the extent subrange appears as a *focus region* (with the focus point at its center), while the level-of-detail subrange corresponds to a sampling rate factor or a *density*. In a 2-D tree representation, the subranges correspond to a horizontal and vertical selection, respectively (Fig. 2).

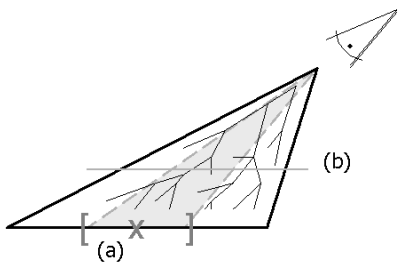

Fig. 1. Structure-based brush: *focus region* (a) and *density factor* (b).

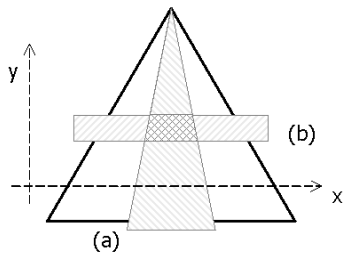

Fig. 2. Structure-based brush: horizontal (a) and vertical (b) selection.

2.2 Brush Semantics

Since a structure-based brush is defined as the intersection of two independent selections, setting such a brush requires two computational phases as well. The first one, the horizontal selection, is accomplished in two steps. In the first step a set of leaf nodes is initially selected based on the order property. Basically, this step corresponds to "select all leaves between the two extreme values e_1 and e_2". In the second phase, the initial selection is propagated up towards the root based on what we term an ALL semantic: "select nodes that have *all* their children already selected" (other semantics like ANY or $MOST$ are also possible [16]). The second computation phase, the vertical selection, consists of refining the set of nodes generated in phase one. Nodes on the desired level-of-detail are only retrieved out of the whole phase one selection.

The brush operations, as described above, are inherently recursive. Recursive processing in relational database systems can be extremely time consuming and thus unsuitable for interactive applications. In Section 3 we develop equivalent but non-recursive computation methods for setting structure-based brushes based on assigning some precomputed values to the nodes that recast retrievals as range queries.

3 MinMax Trees: Translating the Navigation Operations

The question addressed in this section is "how do we translate the visualization operations into database operations". For this purpose we have developed a technique called a *MinMax tree* [16]. The method places the recursive processing in a *precomputation* stage, when labels are assigned to all nodes. The labels provide a containment criterion. Thereafter, simply by looking at a node's label, we are able to determine whether that node belongs to the active selection or not.

3.1 MinMax Tree Definition

A MinMax tree is a n-ary tree in which nodes correspond to open intervals defined over a totally ordered set, called an *initial set*. The leaf nodes in such a tree form a sequence of non-overlapping intervals. The interior nodes are unions of intervals corresponding to their children. The initial set can be continuous (such as an interval of real numbers) or discrete (such as a sequence of integers). In either case, the nodes are labeled as pairs of values: the extents of their interval. As the intervals are unions of child intervals, it follows that a 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 $c_1 = (\alpha, \beta)$ and $c_2 = (\gamma, \delta)$ will be labeled as $n = (\alpha, \delta)$. Essentially, the process of labeling the nodes is recursive. The intervals are computed and assigned off-line at the time the hierarchy is created and their value and distribution (as well as the tree structure itself) depend on the technique used to create the hierarchy.

Given a MinMax tree T and two nodes x and y of T whose extent values are (x_1, x_2) and (y_1, y_2) respectively, node x is an ancestor of node y if and only if $x_1 \leq y_1$ and $y_2 \leq x_2$. The property is based on the intuition that each node in the tree is included in its parent as an interval.

3.2 Using MinMax Trees to Set Structure-Based Brushes

Having a hierarchy labeled as a MinMax tree, we can implement an ALL structure-based brush as a non-recursive operation based on the property that given the brush values v_{min} and v_{max}, the brush represents the union of all nodes $n = (n_1, n_2)$ whose extents are fully contained in the brush interval (v_{min}, v_{max}), i.e., $(n_1, n_2) \subseteq (v_{min}, v_{max})$ [16]. The horizontal selection defined by an ALL structure-based brush in Fig. 3 and the brush values 0.25 and 0.75 is visually depicted in Fig. 4. The selected nodes are underlined.

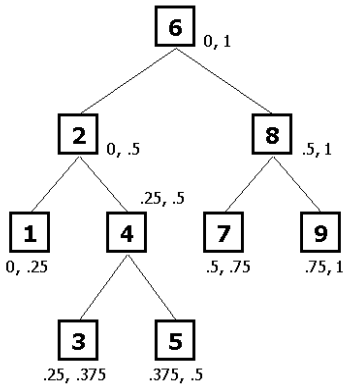

Fig. 3. MinMax tree.

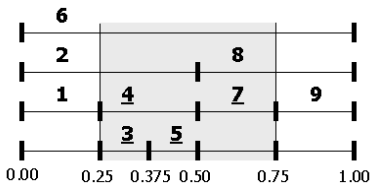

Fig. 4. ALL structure-based horizontal selection.

The value corresponding to the level-of-detail of an aggregation (node) can also be precomputed and stored in the node as part of the label. A vertical selection corresponds further to identifying all nodes at a desired level-of-detail that satisfy the extent selection criterion.

3.3 Translating MinMax Tree Brushes into SQL

The MinMax tree technique reduces the containment criterion from an initially recursive semantic to a simple inclusion test (horizontal selection) and an equality test (vertical selection). We can thus decide whether a node should belong

to the selection or not *independently* from the information stored in the other nodes. One scan of the data is hence sufficient to completely form the selection.

Let H be the relational table that stores the datapoints. Besides the node information, H contains extent and level attributes:

```
H (e_min, e_max, level, ... )
```

An ALL structure-based brush for a hierarchy labeled as a MinMax tree and the brush values (min, max, lev) is now a range query, expressed in SQL as:

```
select  *
from    H
where   e_min >= :min
and     e_max <= :max
and     level  = :lev;
```

While a recursive processing would require an exponential processing time, this range query requires only a linear (logarithmic if using indices) processing time for computing brushing results. Tests to confirm the important improvement achieved by our method are presented in Section 5.

4 Memory Management

The question addressed in this section is "how do we organize the data into memory once it arrives from the database". The memory organization is critical in interactive applications since it influences the performance of the subsequent operations. When a new request is issued by the front-end, the difference between the new requested objects and the current content of the buffer has to be quickly computed. Thus, we need to know in each moment what data resides in the memory buffer without fully traversing it.

A characteristic of the objects that are placed in the buffer is that they are not referenced by their IDs when accessed by the front-end. In other words, the front-end doesn't ask for the object x or y. Instead, it passes a query q to the back-end: "are the objects with these characteristics (within this brush) available". Thus, although there are object attributes (the extents) that uniquely identify each entry, a classical lookup by a cache key is not possible when testing whether an object is in the buffer or not. Instead, a query Q is associated with the buffer, similar to *semantic caching* [4]. The two queries q and Q are then compared to determine what objects from q are not in Q, and those objects are retrieved next. The difference results in a new query q_Δ that corresponds to those *to be selected next* objects.

Computing q_Δ is always possible. Since the selected objects are always on the same level it follows that they can be ordered and their extents correspond to disjoint intervals that are consecutive. A selection becomes an interval. Since both q and Q are selections, and thus intervals, and since the difference between two intervals can always be computed, it follows that the difference q_Δ can also be always computed.

The buffer is implemented as a double linked list in which the objects are stored in the increasing order of their extent value. Thus, at any time the buffer content is fully described by the first and the last element in the list, i.e., (e_1, e_2, L) and (e_{n-1}, e_n, L) respectively. An object on level L is already contained in the buffer if its extent interval is included in (e_1, e_n). The look-up uses exactly two buffer accesses.

A significant difference in the buffer management is made by whether the buffer is large enough to store all the objects in the active set or not. Both cases have been analyzed in detail and shown to be efficient and consistent with the buffer access operations in [17].

5 Experimental Results

Experimental Setup. All the experiments were conducted on an Alpha v4.0 878 DEC station, running Oracle 8.1.5. and having no concurrent clients during the tests. We used C as the host language and embedded SQL statements for accessing the the database. The datasets we used had D1=2^{12} (4,096), D2=2^{16} (65,536), D3=2^{17} (131,072) and D4=2^{18} (262,144) tuples in the hierarchical table. All all them represented complete trees with a constant fan-out (2). The number of dimensions was 8.

Experiment 1: MinMax vs. Recursive. First, we measured the system's performance when implementing the structure-based brushes using both the MinMax tree technique and a recursive technique. As SQL does not support recursive views, the recursive technique used an additional attribute to mark the selected tuples along the recursion steps. In both cases we created simple indices on the join attributes and compound indices on extent and level attributes.

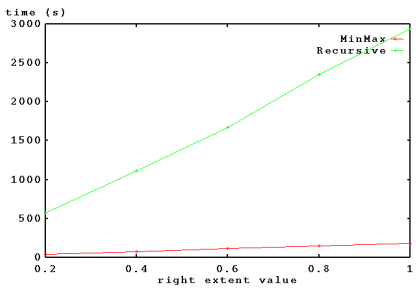

Fig. 5. MinMax vs. Recursive. Structure-based brushes for dataset D3.

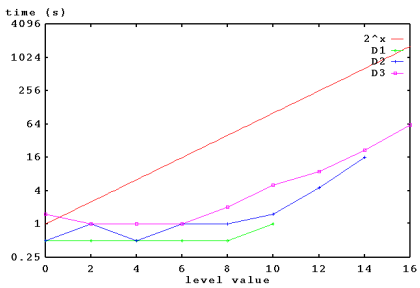

Fig. 6. Varying the level value. Functions compared against 2^x. Logarithmic y scale.

As depicted in Fig. 5), the MinMax method is substantially faster than the recursive one in all cases. These results show clearly that the application of the

MinMax technique in our system made visual exploration over large datasets practically feasible, thus accomplishing our ultimate goal.

Experiment 2: Varying brush parameters. In the next three experiments we analyze the behaviour of the system when the size of the brushes changes. Specifically we vary the level value, the extent values and the size of the dataset. For these experiments we use a compound index on $(e1, e2, L)$ and a $NOCACHE$ hint for the Oracle's optimizer (to keep the impact of the Oracle's caching small). Since the brushes are implemented as range queries, we expect that the response time will vary linearly with respect to all these parameters. Indeed, the results show that the time is at most linear with respect to the size of the input (Fig. 6–8).

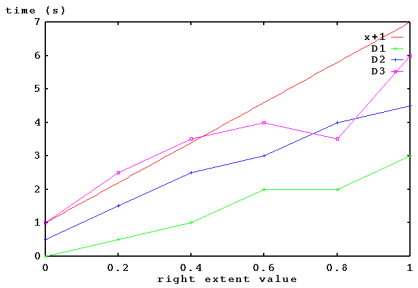

Fig. 7. Varying extent value. Functions compared against $x + 1$.

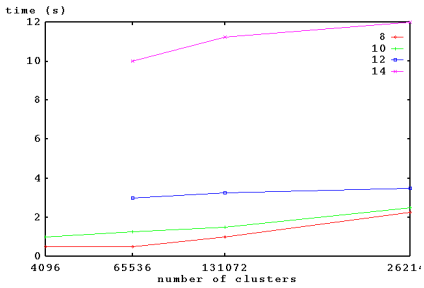

Fig. 8. Varying dataset size. Levels 12 and 14 not defined for D1.

Discussion. The experiments have demonstrated that the processing time for a structure-based brush is only proportional to the number of objects being selected and independent from what brush parameter changed. Thus, the time required for computing a sequence of brushes is proportional to the sum of the objects in all these selections. Due to the incremental computation of the brushes, there is a high probability that small brushes are actually computed rather than large ones. Thus, the amount of the data being read from the database, and implicitly the overall processing, is significantly reduced.

6 Related Work

Visualization-database integration. Integrated visualization-database systems such as Tioga [15], IDEA [14], DEVise [12] represent the work closest related to ours in terms of problem area. The approaches are however different. Tioga [15] implements a multiple browser architecture for what they call a *recipe*, a visual query. However, the problem of query translation is not present since database queries are explicitly specified by the graphical interface. IDEA [14] is

an integrated set of tools to support interactive data analysis and exploration. Some constraints on the data model are imposed by the application domain, but on-line query translation and memory management are not addressed. In DEVise [12], a set of query and visualization primitives to support data analysis is provided. However, caching data is being done at the database level using default mechanisms only.

Other work includes dynamic query interfaces [9], dynamic query histograms [5] and direct manipulation query interfaces [10, 8]. They all have a visual interface and a database back-end. However, the operations translate differently: to dynamic range queries in [5], to temporal queries in [8] and to 2-D spatial queries in [10]. These works do not deal with hierarchy exploration support.

Special techniques for hierarchy encoding. There has been also work in the area of hierarchy representation and querying. Ciaccia et al. [2] 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. This limitation reduces the number of operations that can be supported and makes updates difficult to handle.

Teuhola [18] 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.

Caching and buffer management. High level caching in which objects are not individually identified, but rather a set of objects together is identified with the query that generated it is called *semantic caching* [4] or *predicate caching* [11]. Our buffer management requirements are similar to those present in semantic cashing. Other work in the area of object level caching for database applications have been addressed in [3, 13].

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 in visualization is to integrate visualization applications with database management systems. Such integrations raise two kind of problems: query translation and memory management. This paper presents a solution that addresses both aspects. The approach is being used in coupling XmdvTool, a visualization application for interactive exploration of multivariate data, with an Oracle8i database management system. Experiments for assessing the method showed that, despite the recursive nature of the operations at the interface level, the processing time in our integrated system is only proportional to the number of objects in the active selection. Moreover, the system scales linearly with respect to the size of the dataset.

References

1. A. Becker and S. Cleveland. Brushing scatterplots. *Technometrics, Vol 29(2), p. 127-142*, 1987.
2. P. Ciaccia, D. Maio, and P. Tiberio. A method for hierarchy processing in relational systems. *Information Systems*, 14(2):93–105, 1989.
3. G. P. Copeland, S. Khoshafian, M. G. Smith, and P. Valduriez. Buffering schemes for permanent data. In *Intl Conf on Data Engineering*, pages 214–221, 1986.
4. S. Dar, M. J. Franklin, B. T. Jónsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Intl Conf on Very Large Data Bases*, pages 330–341, 1996.
5. M. Derthick, J. Harrison, A. Moore, and S. Roth. Efficient multi-object dynamic query histograms. *Proc. of Information Visualization*, pages 58–64, Oct. 1999.
6. Y. H. Fua, M. O. Ward, and E. A. Rundensteiner. Hierarchical parallel coordinates for exploration of large datasets. *IEEE Visualization*, pages 43–50, Oct. 1999.
7. Y. H. Fua, M. O. Ward, and E. A. Rundensteiner. Navigating hierarchies with structure-based brushes. *Information Visualization*, pages 58–64, Oct. 1999.
8. S. Hibino and E. A. Rundensteiner. Processing incremental multidimensional range queries in a direct manipulation visual query. In *Intl Conf on Data Engineering*, pages 458–465, 1998.
9. Y. Ioannidis. Dynamic information visualization. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(4):16–16, Dec. 1996.
10. S. Kaushik and E. A. Rundensteiner. SVIQUEL: A spatial visual query and exploration language. *Lecture Notes in Computer Science*, 1460, 1998.
11. A. M. Keller and J. Basu. A predicate-based caching scheme for client-server database architectures. *VLDB Journal*, 5(1):35–47, 1996.
12. M. Livny, R. Ramakrishnan, K. S. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and R. K. Wenger. DEVise: Integrated querying and visualization of large datasets. In *ACM SIGMOD Intl Conf on Management of Data*, pages 301–312, 1997.
13. N. Roussopoulos, C.-M. Chen, S. Kelley, A. Delis, and Y. Papakonstantinou. The ADMS project: View R us. *Data Engineering Bulletin*, 18(2):19–28, 1995.
14. P. G. Selfridge, D. Srivastava, and L. O. Wilson. Idea: Interactive data exploration and analysis. In *ACM SIGMOD Intl Conf on Management of Data*, pages 24–34, 1996.
15. M. Stonebraker, J. Chen, N. Nathan, C. Paxson, and J. Wu. Tioga: Providing data management support for scientific visualization applications. In *Intl Conf on Very Large Data Bases*, pages 25–38, 1993.
16. I. D. Stroe, E. A. Rundensteiner, and M. O. Ward. Minmax trees: Efficient relational operation support for hierarchy data exploration. Technical Report TR-99-37, Worcester Polytechnic Institute, Computer Science Department, 1999.
17. I. D. Stroe, E. A. Rundensteiner, and M. O. Ward. Scalable visual hierarchy exploration. Technical Report TR-00-06, Worcester Polytechnic Institute, Computer Science Department, 2000.
18. J. Teuhola. Path signatures: A way to speed up recursion in relational databases. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):446–454, June 1996.
19. M. O. Ward. Xmdvtool: Integrating multiple methods for visualizing multivariate data. *Proc. of Visualization*, pages 326–333, 1994.
20. G. Wills. Selection:524,288 ways to say this is interesting. *Proc. of Information Visualization '96, p. 54-9*, 1996.