CS-TR-3730
ISR-TR-97-5

Incremental Data Structures and Algorithms for Dynamic Query Interfaces*

Egemen Tanin† U. Maryland
Richard Beigel‡ U. Maryland and Yale
Ben Shneiderman§ U. Maryland

Abstract *Dynamic query interfaces* (DQIs) are a recently developed form of database access that provides continuous realtime feedback to the user during the query formulation process. Previous work shows that DQIs are an elegant and powerful interface to small databases. Unfortunately, when applied to large databases, previous DQI algorithms slow to a crawl. We present a new approach to DQI algorithms that works well with large databases.

Keywords Data Structure, Algorithm, Database, User Interface, Information Visualization, Direct Manipulation, Dynamic Query.

1 Our Innovations

We propose a new approach to dynamic query interface (DQI) algorithms that can handle much larger databases than previous implementations. Our approach gains its efficiency from the following innovations:

Active subset We define an "active" subset of the database, of limited size, which we store in main memory. (While in principle the size of main memory may seem like a severe limitation, in practice DQI algorithms seem to be limited more by time than by space. Our current implementation handles an active subset that is an order of magnitude larger than the databases that previous implementations could handle.)

Auxiliary data structures We augment the active subset with data structures that facilitate continuous querying (users can tolerate a response time of about 0.1 seconds for continuous operations).

Reprocessing The auxiliary data structures change only when the user clicks on a widget. After such an action the user can tolerate a delay of approximately 1 second or less, during which we reconstruct the auxiliary data structures.

Incremental display Slight changes in the query tend to cause only slight changes in the output. By computing and displaying the difference, we can update the display continuously.

2 Background on Dynamic Queries

DQIs were developed recently by [1, 7, 8] as a mechanism for visualizing multidimensional data. DQIs are graphical (as opposed to textual languages such as SQL) and provide continuous feedback to the user as the query is formulated. Experiments show that querying with DQIs is faster, easier, more pleasant, and less error-prone than with other database interfaces [8]. (See Figure 1 for a sample DQI by [1]. Some demos are available from [3, 4].)

Queries are made using widgets, such as range sliders (for continuous data attributes), alphanumeric sliders (for textual attributes), toggles (for binary attributes), and check boxes (for discrete multi-valued attributes), to specify each attribute (dimension) of the data. Output is provided via a starfield display (a 2-dimensional projection of the set of hits), bars (such as a preview bar that displays the number of hits), and charts (which provide other cumulative information). We perform tight coupling among range sliders: as the hit set varies all the range sliders are updated to show the hit set's bounding rectangle, so the range sliders provide a limited form of output as well. If desired, we can even display a histogram on each range slider

*This paper will be presented at the Workshop on New Paradigms in Information Visualization and Manipulation, Nov. '96. Address for authors: Human–Computer Interaction Laboratory, Dept. of Computer Science, University of Maryland, College Park, MD 20742-3251. Research supported by NASA grant NAG 52895.

†egemen@cs.umd.edu, partially supported by NASA grant NAG 52895.

‡beigel@cs.umd.edu, partially supported by NSF grants CCR-8958528 and CCR-9415410 and by NASA grant NAG 52895, on sabbatical from Yale University until 8/1/97.

§ben@cs.umd.edu, partially supported by NSF grant EEC-9402384, also affiliated with the Center for Automation Research and the Institute for Systems Research.

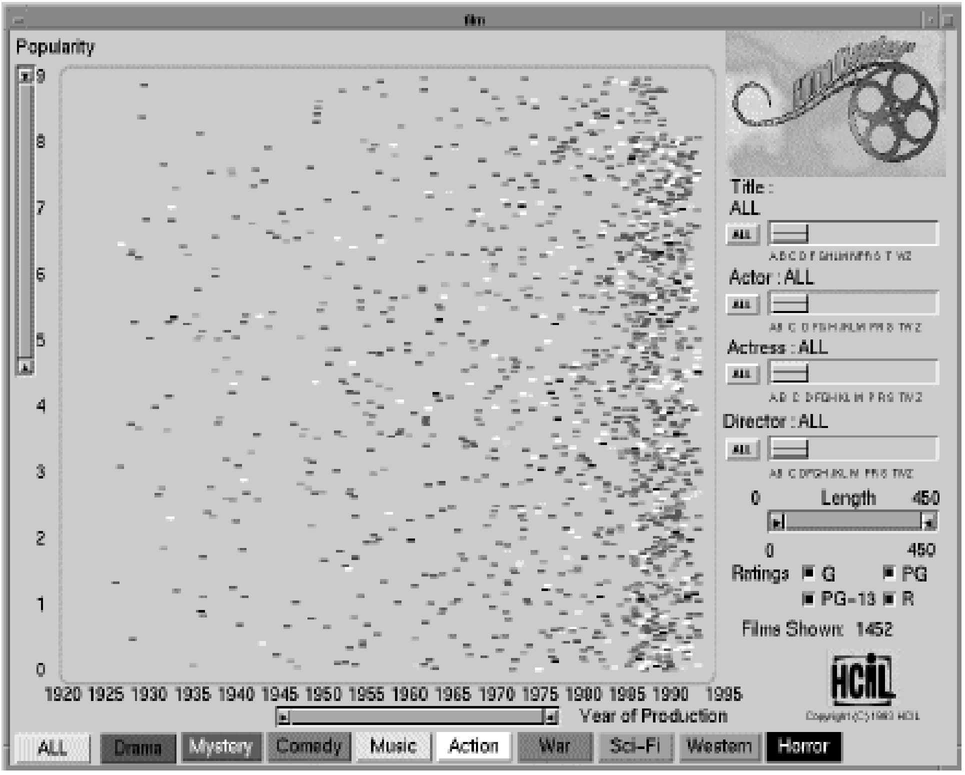

Figure 1: Film Finder [1]: a sample DQI. The large square is the starfield display. The widgets on the left, bottom, and right are for querying. This example does not contain an explicit preview bar or chart.

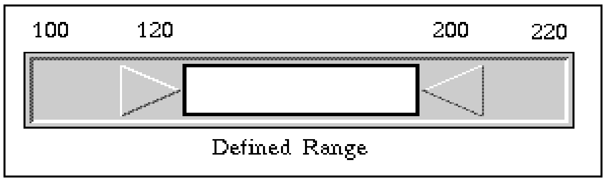

Figure 2: A sample range slider. By moving the arrows, the user specifies the range, which is represented by the white rectangle. The numbers above the arrows give the current range. The numbers on the far ends of the range slider are the extreme values that the attribute can take.

to show the distribution of data in its dimension. Furthermore, the user may click on an individual point in the starfield for details on demand. (See Figure 2 for a sample range slider.)

A range slider contains a pair of arrows, one at each end. The user selects a range slider by clicking on it, and the user adjusts the range by dragging either arrow with the mouse. As the range is adjusted, the starfield display, bars, and charts are updated continuously. Ranges or histograms for other range sliders are updated as well. Thus, for each tiny increment of the range slider, much information must be computed rapidly.

Toggles allow the user to specify a binary attribute of the data. On the screen they look like boxes. Internally they can be implemented directly without much trouble or treated as a nearly trivial special case of range sliders. List boxes and radio buttons can be handled with similar ease. There is no need to discuss them further in this paper.

Alphanumeric sliders allow the users to specify a range of strings. Although our auxiliary data structures apply to them as well, the fine granularity of alphanumeric data seems to necessitate additional implementation ideas that are best described in a separate paper.

3 Description of the Algorithm

We envision using the DQI algorithm in tandem with a query previewer [2] that will allow the user to browse a huge database and select a manageably small subset to scan. Once the user selects such a subset, the query previewer will pass its bounding rectangle to the DQI, which will then take control. The bounding rectangle for the active subset determines the extreme values for each attribute.

If, at some later time, the user wants to look outside the active subset, then the simplest solution is for the DQI to return control to the query previewer. (However, a more sophisticated DQI might allow the user to update the active subset without using the query previewer. The DQI could query the database about the size of the user's proposed new active subset. If it is not too large, the DQI could repeat the setup operation. This will be considered in a future paper on the interaction between the DQI and the query previewer.)

Our DQI algorithm performs three major operations: setup, selection, and querying.

Setup occurs when the query previewer passes control to the DQI. During setup, the widgets, starfield display, bars, and charts are initially drawn on the screen. The DQI reads the active subset. In addition, it makes a copy of the active subset and re-scales each attribute to the range $[1, p]$ where p is the number of pixels in the attribute's range slider. Because setup occurs infrequently, we can allow several seconds for it.

Selection occurs when the user clicks on a range slider. During selection, the algorithm computes the auxiliary data structures, which depend on the currently selected attribute and the current ranges for the other attributes. Various experiments with user interfaces show that we should respond to the mouse click in about 1 second in order not to annoy the user. At the cost of a factor of 2 in memory, we could precompute the auxiliary data structures whenever the mouse is moved close to a new range slider. Using somewhat more memory, we could steal cycles from the query operation in order to precompute the auxiliary data structures for several sliders (chosen according to some heuristic). Thus 1 second may be an overly conservative bound on the time for selection.

Querying occurs continuously as the user drags the mouse to update the selected range slider. During querying, the algorithm recomputes the hit set and updates the output on the screen. For the purpose of timing, we say that a single query occurs each time the DQI detects a change in the position of the mouse on the range slider. Experiments have shown that DQIs should process each query in about 0.1 seconds or less in order to give the appearance of a continuous response [1].

4 Data Structures

Because of the resolution of the screen we can assume that a range slider is no more than 250 pixels long. During the selection operation, the DQI computes the *maximum hit set*, which is the hit set determined by the extreme values for the selected attribute and the current ranges for the other attributes. Then it partitions the maximum hit set into p buckets, one for each user-specifiable value for the current attribute. This is essentially a linear time counting sort of the maximum hit set. We store the size of each bucket and all left-to-right partial sums of these sizes. We also store information that facilitates computation of histograms and tight coupling of range sliders, i.e., the redisplay of all slider ranges when any range is changed. Tight coupling will be discussed in the full version of this paper.

Thus, any time the range slider is updated, even by a large number of pixels (as might happen if the user moves the mouse extremely fast), we can determine the number of hits in constant time. We can also determine ranges for the other sliders in constant time per slider, and histograms for the other sliders in constant time per point. Changes to the display are determined by scanning the buckets between the older attribute value and the new one. The display is updated in time that is linear in the number of changes.

If the user changes the axis parameters for the starfield display, then the hits are redisplayed (projected on the new pair of coordinates) but none of the internal data structures is changed.

5 Complexity

Let r denote the number of records in the active subset, a the number of attributes, and b the number of bytes needed to store the value of a single attribute. Let p denote the length in pixels of each range slider, f the area in pixels of the starfield, and u the average number of pixels that need to be updated in the starfield display per query (this number will depend in a nontrivial way on the size of the starfield, the velocity of the range slider, and the clustering of data in the active subset.) Let m denote the number of records in the maximum hit set.

The active subset occupies $r \cdot a \cdot b$ bytes. The re-scaled active subset occupies $O(r \cdot a)$ bytes. The bucket partition also occupies $O(r \cdot a)$ bytes. The data structures for tight coupling occupy $O(a \cdot p)$ bytes. The data structures for range histograms occupy $O(a \cdot p^2)$ bytes. The starfield occupies f bytes.

Setup takes time $O(r \cdot a \cdot b)$.

There are four components to the time for selection. Determining the maximum hit set takes time $O(r \cdot a)$. Sorting the maximum hit set takes time $O(m)$ (really, there is no log factor because we discretize the data.) Computing the auxiliary data structures for tight coupling takes time $O(a \cdot p + m \cdot a)$. Computing the auxiliary data structures for histograms takes time $O(a \cdot p^2 + m \cdot a)$. Thus, the total time for selection is $O(a \cdot (r + m + p^2)) = O(a \cdot (r + p^2))$.

There are three components to the time for querying. Tight coupling takes time $O(a)$. Computing histograms takes time $O(a \cdot p)$. Updating the starfield takes time $O(u)$. Thus the total time for querying is $O(a \cdot p + u)$.

6 Our Results and Comparison to Prior Work

Our preliminary experiments were successful with an active subset consisting of 100,000 records with 10 attribute each. On average, selection took about 1 second, and querying took about 0.1 second (on a SUN Microsystems SPARC 5 with 32 megabytes of main memory). It is notable that the querying time is dominated by the starfield update time. When only summary information is displayed, querying takes about 0.02 seconds.

In comparison, the pioneering work in the area, the Film Finder program [1], could handle a database of 10,000 records with 10 attributes, and some of the standard data structures analyzed in [5] and tested in [6] demonstrated scalability up to 20,000 records with 10 attributes.

7 Future Directions

We plan to:

- perform comprehensive experiments to determine the running time for various parts of our operation as functions of the database size, screen size, range slider velocity and other parameters.

- implement alphanumeric sliders.

- try spatial data structures like k-D trees to see how they effect the times for selection and querying. (As a general non-worst-case rule of thumb, spatial data structures answer range queries in time $O(|H|^{1-1/a})$ where H is the set of hits. This could be good for selection, because it is sublinear. But it could be bad for querying, because it is close to linear, and prior work seems to confirm this doubt [5, 6]. Instead, we will use an incremental approach where we compute the difference ΔH between consecutive hit sets, which in practice should take time only $O(|\Delta H|^{1-1/a})$.)

- extend our work to active subsets consisting of about 10^6 records. (Querying (including updating the starfield display), tight coupling, and histograms will scale up without difficulty, because their complexity does not depend on the number of records in the active subset. Selection, however, seems to be a problem, because its complexity is proportional to the size of the active subset. By stealing cycles and precomputing, we plan to hide some of the time for selection; however, we will have to experiment with actual users to see how much this helps. Spatial auxiliary data structures may help as well.)

- ultimately combine our DQI with a query previewer [2] developed at the Human-Computer Interaction Laboratory in order to produce a new state of the art in interactive dynamic database access.

References

[1] Ahlberg, C. and Shneiderman, B., Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays, *Proc. ACM Human Factors in Comput. Systems '94*, 1994, pp. 313–317.

[2] Doan, K., Plaisant, C., and Shneiderman, B., Query Previews in Networked Information Systems, *CfAR Tech. Report*, University of Maryland, College Park, CAR-TR-788, Oct. 1995.

[3] HCIL, ftp://ftp.cs.umd.edu/pub/hcil/Demos/DQ/dq-home.zip. Downloadable PC demo.

[4] IVEE Development AB, http://www.ivee.com/. Online Java demo and downloadable demos for various platforms.

[5] Jain, V. and Shneiderman, B., Data Structures for Dynamic Queries: An Analytical and Experimental Evaluation, *CfAR Tech. Report*, University of Maryland, College Park, CAR-TR-685, Sep. 1993.

[6] Pointek, J., personal communication, 1995.

[7] Shneiderman, B., Dynamic Queries for Visual Information Seeking, *IEEE Software*, Vol. 11, No. 6, 1994, pp. 70–77.

[8] Williamson, C. and Shneiderman, B., The Dynamic HomeFinder: Evaluating Dynamic Queries in a Real-Estate Information Exploration System, *Proc. ACM SIGIR '92*, 1992, pp. 339–346.