Information Visualization (2004) 3, 1–18
© 2004 Palgrave Macmillan Ltd. All rights reserved 1473-8716 $25.00
www.palgrave-journals.com/ivs

Dynamic query tools for time series data sets: Timebox widgets for interactive exploration

Harry Hochheiser[1]
Ben Shneiderman[1,2]

[1]Human–Computer Interaction Lab & Department of Computer Science, University of Maryland, College Park, MD, U.S.A; [2]Institute for Systems Research and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, U.S.A

Correspondence:
Harry Hochheiser, Laboratory of Genetics, National Institute of Aging, 333 Casell Drive, Suite 3000, Baltimore, MD 21224, U.S.A. Tel: +1 410 558 8046, E-mail: hsh@nih.gov

Abstract
Timeboxes are rectangular widgets that can be used in direct-manipulation graphical user interfaces (GUIs) to specify query constraints on time series data sets. Timeboxes are used to specify simultaneously two sets of constraints: given a set of N time series profiles, a timebox covering time periods $x_1 \ldots x_2$ ($x_1 \leq x_2$) and values $y_1 \ldots y_2$ ($y_1 \leq y_2$) will retrieve only those $n \in N$ that have values $y_1 \leq y \leq y_2$ during all times $x_1 \leq x \leq x_2$. TimeSearcher is an information visualization tool that combines timebox queries with overview displays, query-by-example facilities, and support for queries over multiple time-varying attributes. Query manipulation tools including pattern inversion and 'leaders & laggards' graphical bookmarks provide additional support for interactive exploration of data sets. Extensions to the basic timebox model that provide additional expressivity include variable time timeboxes, which can be used to express queries with variability in the time interval, and angular queries, which search for ranges of differentials, rather than absolute values. Analysis of the algorithmic requirements for providing dynamic query performance for timebox queries showed that a sequential search outperformed searches based on geometric indices. Design studies helped identify the strengths and weaknesses of the query tools. Extended case studies involving the analysis of two different types of data from molecular biology experiments provided valuable feedback and validated the utility of both the timebox model and the TimeSearcher tool. Timesearcher is available at http://www.cs.umd.edu/hcil/timesearcher
Information Visualization (2004) **3,** 1–18. doi:10.1057/palgrave.ivs.9500061

Keywords: TimeSearcher; timeboxes; dynamic query; visual query; angular queries; time series; temporal data; bioinformatics; graphical user interfaces

Introduction

Analysts in many domains study measurable quantities that change over time. Financiers examining trends in economic indicators, meteorologists studying climate data, demographers quantifying trends in census data, and numerous others use time series graphs, statistical evaluations, and other tools to identify patterns and find trends in these time series data sets. Our motivation for this work stems from collaboration with molecular biologists who are studying gene expression data, where typical experiments involve thousands of gene over tens of time periods.

Algorithmic and statistical methods for identifying patterns have provided substantial functionality in a wide variety of situations,[1–4] but this research only addresses one aspect of the analysis problem. The question of query formulation – which questions are worth asking? – is often left unanswered.

Users need tools to support interactive exploration of the contents of time series data sets. By providing analysts with the power to construct

Received: 3 June 2003
Revised: 16 October 2003
Accepted: 16 October 2003

queries quickly, modify parameters, and examine result sets, these tools would encourage the development of understanding of these data sets.

Familiar graphic displays of time series presents an obvious starting point for the application of information visualization techniques to time series data. Timeboxes are rectangular regions that specify queries on these displays.[5] After introducing timeboxes, this paper discusses the TimeSearcher application, which supports timebox queries along with a number of extensions designed to provide increased expressivity. Algorithmic implications and evaluations through design studies and case studies are also discussed.

Timeboxes

Timeboxes are rectangular query regions drawn directly on a two-dimensional display of time series data. The extent of the timebox on the time (x) axis specifies the time period of interest, while the extent on the value (y) axis specifies a constraint on the range of values of interest in the given time period. Given a data set consisting of n items, each of which has a measurement at each of m time points, a timebox acts as a filter that accepts only those items that have values in the given range during the interval spanned by the box.

More specifically, assume that $n_i \in N$ is an item in a time series data set, $n_i(j)$ is the value of n_i at time j, and a timebox is a 4-tuple: $b = (t_{min}, t_{max}, v_{min}, v_{max})$ (Figure 1). An item n_i satisfies the timebox b if $\forall_{t_{\min} \le t \le t_{\max}} v_{\min} \le n_i(t) \le v_{\max}$ (assuming $v_{\max} \ge v_{\min}$ and $t_{\max} \ge t_{\min}$).

Creation of timeboxes is straightforward: users click on the desired starting point of the timebox and drag the pointer to the desired location of the opposite corner. As the box is drawn, it is constrained to occupy an integral number of time points. As this is identical to the mechanism used for creating rectangles in widely used drawing programs, this operation should be familiar to most users.

Once the timebox is created, it may be dragged to a new location or resized via appropriate resize handles on the corners, using similarly familiar interactions. In all cases, the query is re-processed with each mouse event. When user action leads to a modification of a timebox, the new position of the timebox is stored, the query is updated, and the new result set is displayed.

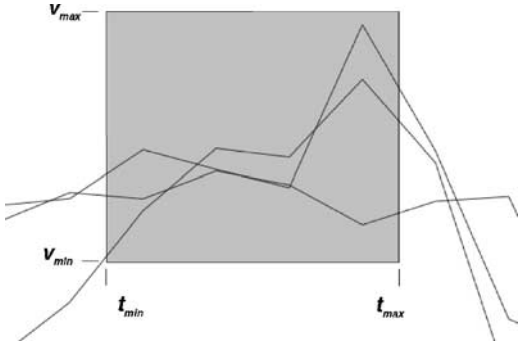

Figure 1 A Timebox query expresses constraints in time and value. An item n_i in the data set will satisfy a timebox query if and only if $\forall_{t_{\min} \le t \le t_{\max}} v_{\min} \le n_i(t) < v_{\max}$.

Multiple timeboxes can be drawn to specify conjunctive queries. Items in a data set must match all of the constraints implied by the active timeboxes in order to be included in the result set.

The example data set shown in Figure 2 contains 52 weekly stock prices for 1430 stocks and will be used in a brief scenario to illustrate the use of timeboxes. The *graph overview* display provided in Figure 2(a) shows all of the items in the data set, drawn directly on the query area. This overview provides insight into the density, distributions, and patterns of change found among items in the data set.

An analyst interested in finding stocks that rose and then fell within a 4-month period might start by drawing a timebox specifying stocks that traded between 70 and 190 during the first few weeks. When this query is executed, the graph overview is updated to show only those records that match these constraints. This query substantially limits the number of items under consideration, but many still remain (Figure 2(b)).

To find stocks in this restricted set that dropped in subsequent weeks, this query is extended by a second box, specifying items that traded between 12 and 80 during weeks 10–12 (Figure 2(c)). A third box, specifying a higher price range (60–120) during weeks 19–24 completes the query (Figure 2(d)).

As timeboxes are added to the query, the graph overview provides an ongoing display of the effects of each action and an overview of the result set. Once created, the timeboxes can be scaled or moved singly or together to modify the query constraints.

The use of simple, familiar idioms for creation and modification of timeboxes supports interactive use with minimal cognitive overhead. Rapid (<100 ms), automatic query processing on mouse events provides the virtually instantaneous response necessary for dynamic queries, thus supporting interactive data exploration. Users can easily and quickly try a wide range of queries, modifying these queries to see quickly the effects of changes in query parameters. This ability to explore easily the data is helpful in identifying specific patterns of interest, as well as in gaining understanding of the data set as a whole.

TimeSearcher

TimeSearcher is an information visualization tool based on the use of timeboxes to pose queries over a set of entities with one or more time-varying attributes. TimeSearcher is written in Java, using Piccolo for all graphics rendering and scenegraph management.[6] A screenshot of the main TimeSearcher window is given in Figure 3.

The top left corner of the TimeSearcher window is the query input space. TimeSearcher's bottom left window –

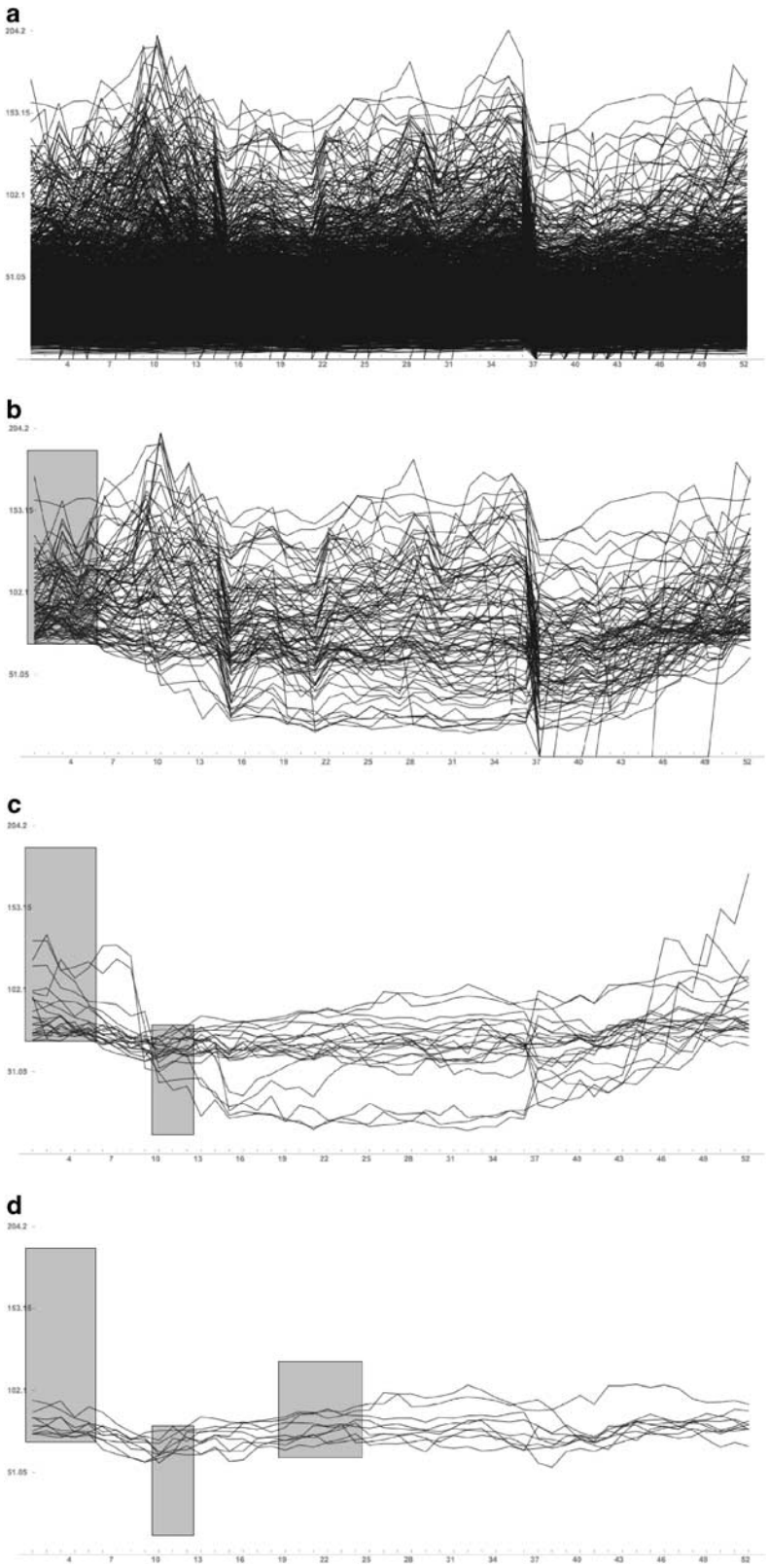

Figure 2 Timebox queries can be combined to form conjunctive queries of arbitrary complexity. The graph overview displayed, which is formed by superimposing the time series for all items in the data set on the query space, shows how the addition of new query constraints leads to a reduction in the set of items that satisfy the query. (a) Graph overview display for the entire data set. (b) Single timebox query, for items for items between 70 and 190 during weeks 1-5. (c) Query containing two timeboxes, refining the query in (b). (d) Three-timebox query: a further refinement of the query in (c).

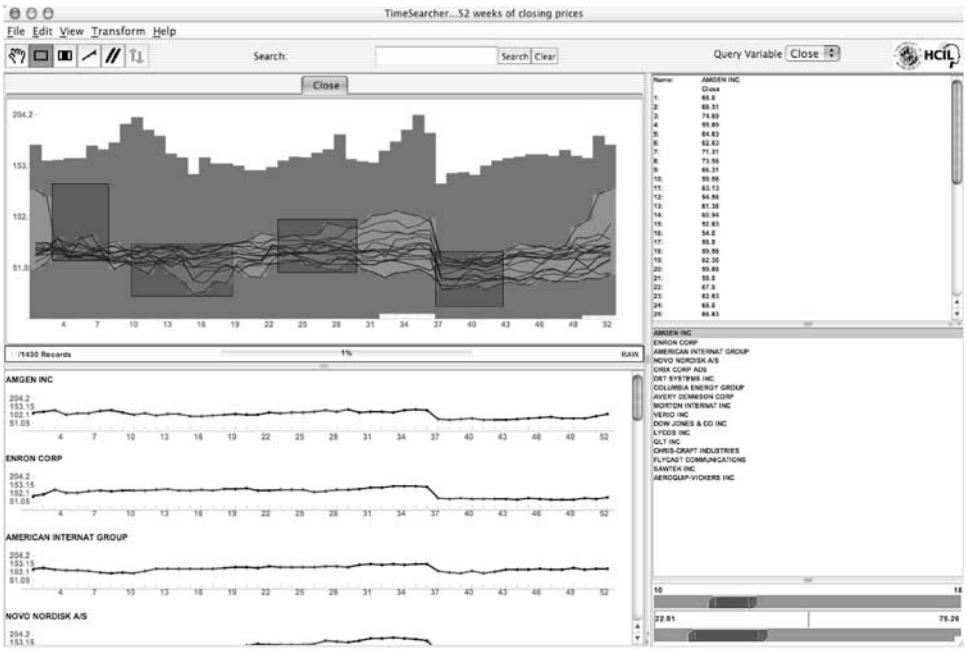

Figure 3 The TimeSearcher information visualization tool is based on the timebox query model. Clockwise from upper-left: query space where timeboxes are drawn (with data envelope, query envelope, and graph overview) details-on-demand for selected items, list of items by name, range sliders for query adjustment, and display list, containing the graph display for each item in the data set.

the display list – contains graphs for each of the items in the data set, in a scrollable list. The items in this list are tightly coupled with the upper right-hand window, which provides complete details-on-demand for an item, and the middle right hand window – the item list – which displays all of the items in the data set by name. These three windows are tightly coupled: selection of an item in the display list leads to the selection of the item in the item list (and *vice versa*), and the display of the item in the details-on-demand window. When query processing completes, the display list and the item list on the right are updated to show those entities that match the query constraints.

Once the initial query is created, the timeboxes can be moved and resized. The hand and box icons on the upper toolbar are used to switch between creating timeboxes and moving/resizing them. As is the case with initial timebox creation, the query is reprocessed with each mouse event. Timeboxes can also be adjusted via paired range sliders and text entry fields in the lower-right hand corner of the screen.

When multiple timeboxes are present, they can be modified individually or simultaneously in groups of two or more. This functionality is particularly useful for searches for complex patterns (Figure 2(d)). Users can select some or all of the timeboxes (using standard lasso and shift-click interactions) and simultaneously apply the same translation and/or scale along either or both axes to all selected timeboxes. This is useful for searching for instances of a pattern that vary slightly in scale or magnitudes, or for modifying queries based on example items.

TimeSearcher also provides support for querying by example. An individual item from the display list can be dragged and dropped onto the query space. This leads to the creation of a query consisting of one timebox for each time point, centered around the value of the given item at that time point. These timeboxes can be modified to specify for varying definitions of similarity. For example, the boxes could be enlarged to allow for a looser definition of similarity, or subsets of the query could be eliminated to focus on items that are similar only at specific time points. This tool is particularly useful in combination with the search box, which can be used to find items by name. Together, these tools support searches for items similar to an item known by name.

Overviews of the data set are provided by the graph overview display described above. The lines in the graph overview also support browsing: when users mouse over one of these lines, it is highlighted, thus displaying the individual item in the context of the larger data set. At the same time, the name of the item is displayed as a tooltip, along with the value of the item at the time point closest to the point where the mouse-over occurred. The item list, display list, and details-on-demand window are also updated to display the selected item. This tight coupling in response to lightweight mouse movement will encourage exploration based on visual examination of the graph overview.

Overdrawing and visual clutter might cause the graph overview display to become less useful for large data sets. Furthermore, the computational overhead of drawing the graph overviews and processing the mouse-over handling

can lead to substantial performance degradation when graph overviews are used with these data sets.

To avoid these difficulties, TimeSearcher provides a lower-resolution overview that produces less screen clutter and has lower computational requirements. Known as a *data envelope*, this overview is shown in the background of the query window as a contour that follows the extreme values of the query attribute at each point in time, thus displaying the range of values that may be queried. When users execute a query, the data envelope is extended by a *query envelope* – an overlay that outlines extreme values of the entities in the result set (Figure 4).

TimeSearcher supports the possibility of graceful degradation between overviews. For large result sets, the data and query envelopes will be shown. When user queries reduce the size of the data set below a user-specified threshold (set to 100 items by default), the graph overviews will be displayed. The use of graph overviews for smaller result sets and data/query envelopes for larger result sets thus provides an example of a dynamic decision regarding the tradeoff between high-resolution overviews and performance.

Leaders & laggards

The analysis of time series data sets frequently includes a search for items with behavior trends that somehow anticipate changes that will eventually be seen in other items in the data set. For example, stock market analysts might look for a given stock that dropped sharply shortly before other stocks in the same sector experienced a similar decline in price. Similarly, biologists looking at microarray experiments might be interested in finding a gene that has a sharp increase in expression levels immediately before a group of genes has a similar increase. Such a finding might form the basis for the hypothesis that the first gene is a regulatory gene that plays a role in stimulating the expression of the other genes.

TimeSearcher provides a graphical bookmark mechanism that supports this search for ''Leaders & Laggards''. After creating a timebox query that identifies the set of items with a trend of interest, users press a toolbar button to invoke this ''leaders'' mode. The query window will then be split into two sub-windows:

- The top, ''leader'' window contains the originally specified query, along with the graph overview lines for items that match that query.
- The lower, ''laggard'' window contains one new query box for each timebox in the original query. These new query boxes are offset by one time period from their original counterparts. This display also includes outlines of the timeboxes found in the original query.

These two windows are vertically aligned, in order to support direct visual comparison between the two queries and result sets. An example query, and its use as a ''leader'', are shown in Figure 5.

Once the leader and laggard windows have been created, users can use the standard mechanisms to modify the query in the laggard window as desired. For example, users might lasso all of the timeboxes in the laggard query and move them further in the time direction, in order to find items that lag the original query by an arbitrary number of time points. Alternatively, users might scale the laggard boxes to find items that have a different range of values than the original query.

The ''leaders & laggards'' facilities grew out of design discussions with molecular biologists who were interested in using TimeSearcher to identify genes that exhibited certain expression behaviors before or after other genes. Identification of these relationships can be very useful in identifying potential regulatory interactions.

The implementation of leaders & laggards as a graphical bookmark represents a compromise between functionality and performance. As implemented, leaders

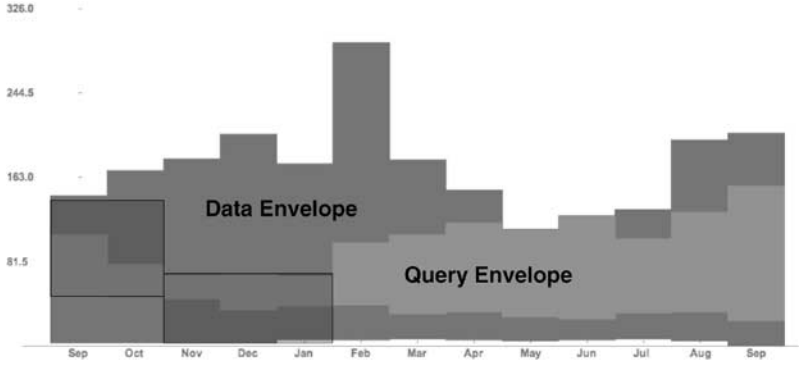

Figure 4 Query display with data and query envelopes. Graph overviews (Figure 2) can become cluttered when the result set is large. When this happens, lower-fidelity overviews that show the ''shape'' of the result set without displaying each item individually might be more useful. The data envelope (dark gray) is a contour formed by the largest and smallest values of any item in the data set at each point in time. The query envelope (light gray) includes only those items that match the current query.

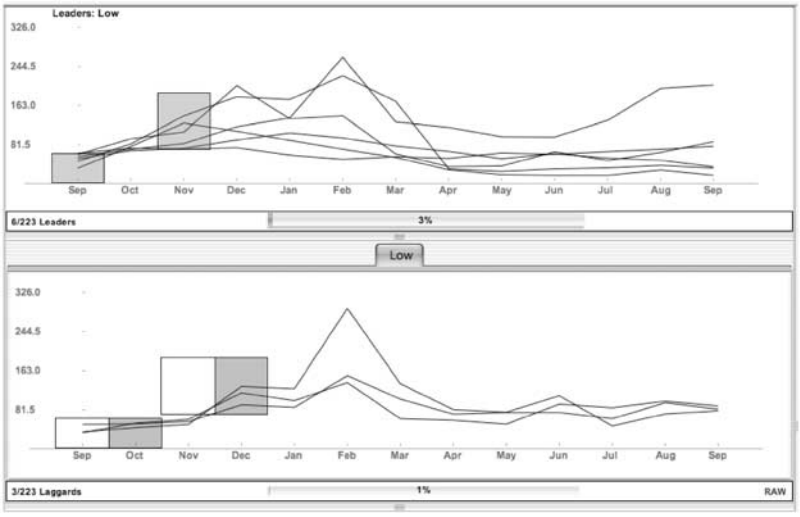

Figure 5 TimeSearcher's ''leaders & laggards'' tool can help users identify items that have similar transitions at different times. In this mode, the query space is split into two new windows. The top window shows the original query. Below this window, the laggards display has new timeboxes representing the new query, which is defined by shifting the old query one time period to the right. The laggards display also contains outlines of the timeboxes found in the original query. By comparing the results of the laggard query in the lower window to the original leader query, the user can identify items that undergo transitions similar to those experienced by results from the leader query, but at a later time. The laggard query can be arbitrarily modified to specify the desired similarity between leaders and laggards.

& laggards supports direct identification of patterns that are offset by one time period, with user modification of the query required for further comparison. A more powerful variant might have supported a fully open-ended query, identifying all items in the data set that undergo a given transition at any points in time. However, evaluating this query within the 100 ms response time desired for dynamic query performance would be very difficult for even moderately-sized data sets.

The leaders & laggards functionality might be extended with a more generalized bookmark facility, which would provide similar functionality for multiple stored queries. These stored queries would serve as a library of templates that might be used to identify patterns of interest.

Query inversion

Having found items in a data set that match a specific query, users might like to find items that have opposite behavior patterns. For example, a stock analyst might like to see stocks that fell at the same time as others were rising. TimeSearcher's ''query inversion'' facility supports this task.

Queries containing one or more timeboxes can be inverted by selecting the desired timeboxes and pressing a toolbar button. The inverse query is derived by calculating a pivot point for the group, and rotating each of the selected boxes around this pivot. The pivot is used by finding the average of the upper-most upper bound and lower-most lower bound for all of the selected timeboxes (Figure 6).

As the original queries all must fit within the parameters of the results set, this approach to inversion has the desirable feature that the resulting inverse query is guaranteed to be a legal query. Other definitions of reciprocal queries – for example, taking the first timebox as a given constant and rotating other boxes relative to this first box – might lead to queries that fall outside the range of values that are found in the data set.

Multiple time-varying attributes

Although the basic definition of timeboxes assumes a data set containing items having a single measurement that varies over time, there is no particular reason for restricting consideration to data sets involving only one attribute. In fact, many meaningful data sets include multiple simultaneous measurements.

TimeSearcher uses a tabbed display to support data sets with multiple variables. When a data set with multiple variables is loaded into TimeSearcher, the first variable in the data set is initially shown as the default, in a single pane of a tabbed pane window. To examine and query the values of any other variable, users select the desired variable name from the pull-down menu marked ''Query Variable'' in the toolbar. This leads to creation of a new frame in the tabbed pane.

When multiple attributes are present, users can switch between them by clicking on the tab at the top of the pane. An attribute can be removed by clicking the close icon (the '' × '') in the appropriate tab, and reinstated by making the appropriate selection in the pull-down menu. The pane for each attribute acts as a query space for that

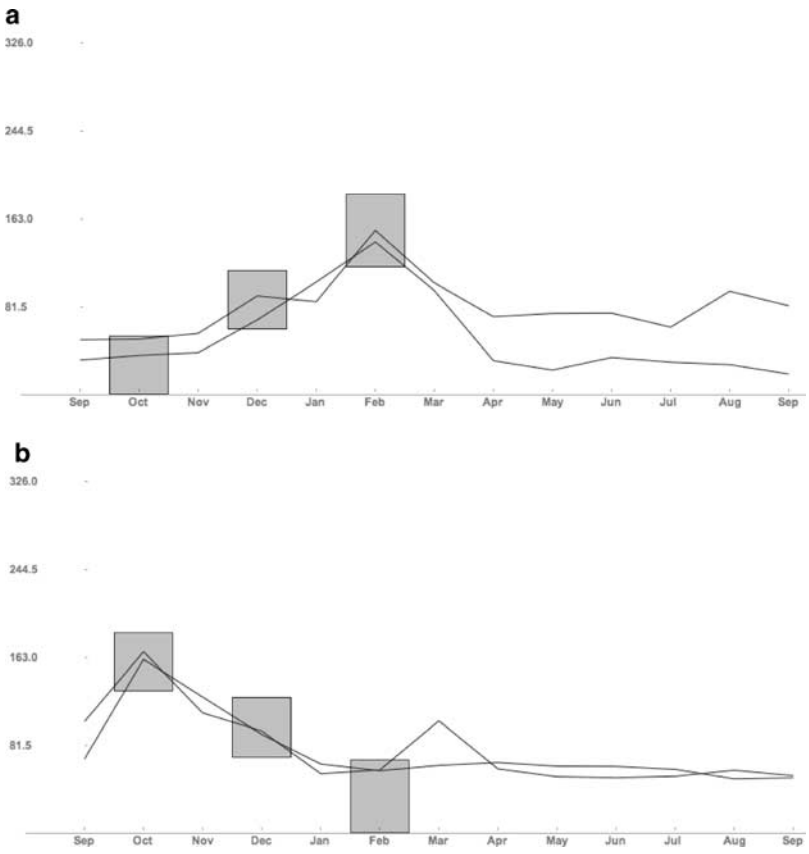

Figure 6 Query inversion enables users to find patterns that are opposite to each other by horizontally flipping the timeboxes around the center of their range. (a) A query containing three timeboxes finds two items that have an upward trend. (b) The inverse of this query finds two items with a downward trend.

attribute. Queries can be created independently for each attribute, and only items that match all queries – even those for variables in panes other than that which is currently selected – will be included in the result set. When a query is created or modified, the query envelopes and graph overviews for each active variable will be updated to display the appropriate subset of the results. Figure 7(a) shows the query window to a data set containing two variables. There are no timeboxes on the selected query space (''high''), but the query envelope has been updated to indicate the filtering of the data set by queries that have been created in the query space for the other variable (''low'').

The use of the tabbed pane for multiple time-varying attributes represents an efficient use of screen space at the potential expense of lesser clarity and greater cognitive load. As only one query space is visible at any given time, users must remember the queries that have been created on other variables in order to interpret search results. TimeSearcher provides an optional ''summary'' overview that can help alleviate this problem. This window contains miniature views of all of the active query spaces is opened. Each summary view is labeled with the name of the appropriate attribute, and the summary view corresponding to the currently active query window is highlighted (Figure 7(b)).

These windows contain active linked views that are dynamically updated with each event that updates the query space. Although the miniaturized views do not provide enough detail to fully interpret the queries, they provide reminders of the content of the occluded query spaces without taking large amounts of screen space from the currently selected query.

Future work might address alternative solutions to this problem of occlusion. For example, the tabbed pane might be replaced by a series of individual windows, one for each attribute. These windows would be coordinated, providing multiple perspectives similar to those found in Snap-Together Visualizations.[7]

Variable-time timeboxes

The basic timebox is limited to expressing queries with fully defined time and value constraints. Additional expressive power might be gained by extending the model in a manner that relaxes these constraints. One possibility is to support searches for items that fall within a given value range during some interval of a given duration that falls within some longer window of time.

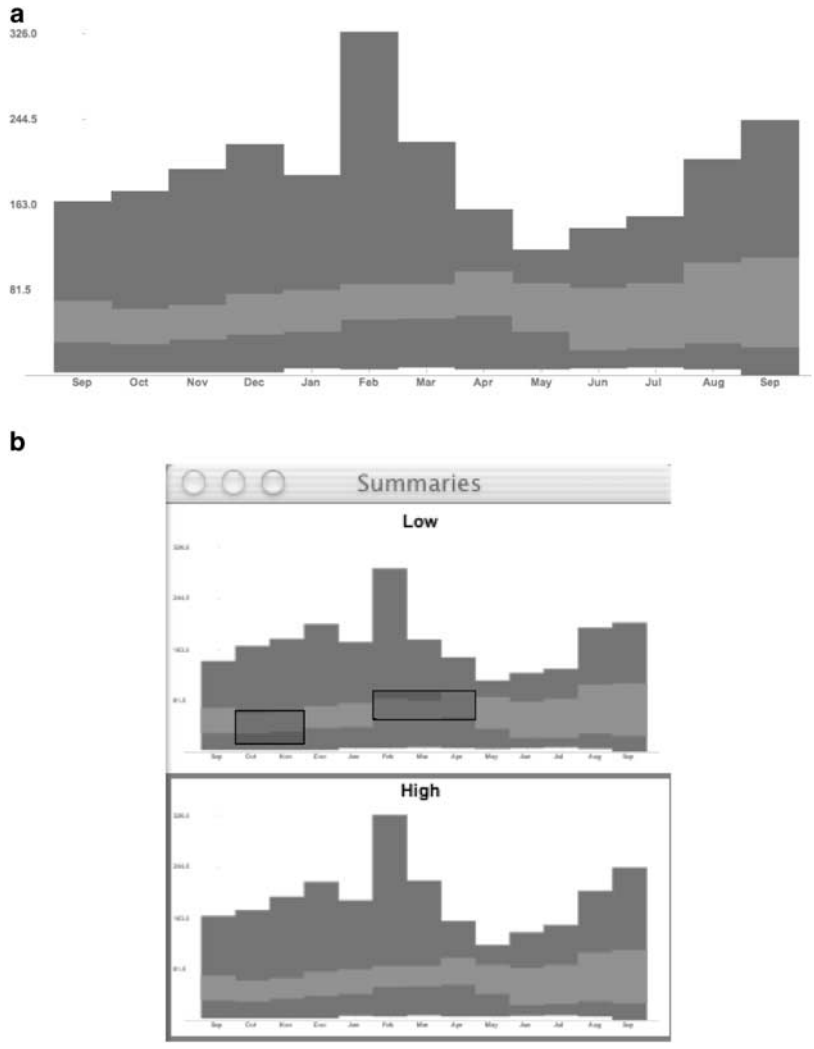

Figure 7 Multiple-attribute queries can be used to express simultaneously queries involving multiple time-varying values. A tabbed-pane interface supports switching between multiple query spaces, one for each time-varying value. As queries are created in each space, the conjunction of all active queries provides the overall result. Overviews in each query space are updated to reflect queries constrained over any of the variables. A summary window provides a smaller-scale view of all active query spaces. (a) Updated query envelopes for one of two attributes that are currently active. Note that even though there are no queries in this window, queries in the inactive window (for ''Low'' measurements) have constrained the data set, as shown by the query envelope. (b) Summary window for the query in (a).

For example, stock analysts might want to identify stocks that traded between 30 and 60 for some 3-month period anytime between January and August (inclusive). These queries are known as variable time timeboxes (VTTs).

Formally, a VTT is defined as two points (x_1,y_1) and (x_2,y_2) and a single integer R. The VTT provides a constraint on a time series such that for the time range $x_1 \le x \le x_2$, the dynamic variable must have a value in the range $y_1 \le y \le y_2$ for at least R consecutive time units (assuming $y_2 \ge y_1$ and $x_2 \ge x_1$). A VTT with a value of $R = x_2\text{-}x_1$ is simply a standard timebox.[8]

Graphically, VTTs are represented as outline boxes that surround a traditional time box (Figure 8). A VTT is created by drawing a box, just as a standard timebox is created. Initially, the interval of interest covers the entire width of the box: $R = x_2\text{-}x_1$. By clicking and dragging the sides of the internal rectangle, users can adjust the value of R.

A preliminary evaluation of variable time timeboxes has shown that they can be useful for creating and interpreting queries that attempt to separate items in a time series data set into disjoint classes.[8]

Angular queries

Timeboxes are straightforward and easy to interpret. This simplicity has helped illustrate their utility in a number of contexts. However, timeboxes are limited to searches based on changes between specific, fixed values. This may

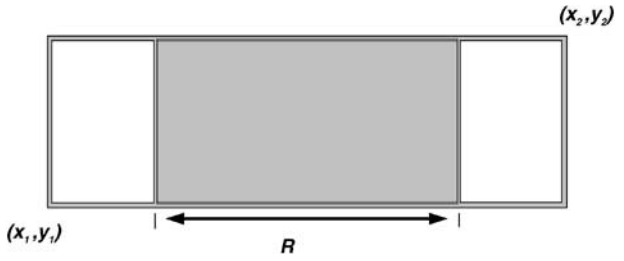

Figure 8 Variable time timeboxes relax some of the constraints of timeboxes, allowing searches for items that fall within a given range for some number of time periods that fall within a longer interval. For at least R consecutive time periods between x_1 and x_2, items must have values in the range $y_1 \leq y \leq y_2$.

limit their utility for analyses involving relative rates of change. For example, timeboxes can be used to find items that rise from a value of 80 to a value of 120 four time periods later, but they cannot be used to identify all items that rose by 40 in value – regardless of the starting value – over that same time period.

The display of individual data items on a two-dimensional query space – as used in TimeSearcher's graph overviews – provides a natural basis for this sort of query. As an item in a data set changes over time, its graph will form an angle with the horizontal (Figure 9(a)). Query widgets that allow users to specify minimum and maximum desired values for this angle over some period of time can be used to identify relative changes in value.

TimeSearcher's *angular query* widget supports this relative query. An angular query is a four-tuple: $b = (t_{min}, t_{max}, \theta_{min}, \theta_{max})$. As with standard timeboxes, t_{min} and t_{max} specify the starting and ending points for the query. The angles θ_{min} and θ_{max} ($-\pi/2 \leq \theta_{min} \leq \pi/2$ and $-\pi/2 \leq \theta_{max} \leq \pi/2$) present upper and lower bounds on the angle that the item's profile must form with the horizontal during the interval in question. These constraints must be met by each transition during the given interval: if $\phi(n_i(t), n_i(t+1))$ is the angle formed by the segment connecting $n_i(t)$ with $n_i(t+1)$, an item n_i satisfies the angular query b if $\forall_{t_{\min} \leq t \leq t_{\max}} \ \theta_{min} \leq \phi(n_i(t), n_i(t+1)) \leq \theta_{max}$.

The angular query widget consists of two lines. An angled line from the starting point to the ending point indicates the time interval associated with the query. This line meets a vertical line at the ending point. The segment between the left-hand end of the query and the bottom of this line is used to calculate the minimum angle of the query, while the segment between the left-hand end and the top of the vertical line defines the maximum angle, as shown in Figure 9(b). The width of the angular query widget is used to indicate fixed starting and ending points. Handles on the angled line and the vertical line can be dragged to modify the query's length in time and angles.

An example query is shown in Figure 9(c). Note that the query widget is above the item that matches the query. As the angular query is based on relative changes between values, the vertical position of the query widget is unimportant: as long as the relative positions of the end points do not change, the widget can be translated vertically without changing the result of the query.

Angular queries are conceptually distinct from timeboxes. A timebox is constructed around the notion of requiring that items fall inside the box during a given interval. Angular queries do not have any ''insides'' – they simply specify a range of slopes. Preliminary observations with users indicates that this does not pose any difficulties for comprehension, but further investigation will be needed to understand the usability tradeoffs involved in the designs, as well as to understand the relative utility of these query widgets.

Angular queries are conceptually similar to *angular brushes* in parallel coordinates. Angular brushes can be used to find trends of a certain direction and magnitude in parallel coordinates displays, without regard for initial comparison point.[9] CASSATT uses a dialog box to provide similar functionality.[10]

There are two important differences between angular queries and angular brushes. Angular brushes are limited to comparisons between two adjacent axes, while the comparisons specified by angular queries may involve comparison across time points separated by an arbitrary interval–adjacency is not required.

Angular brush widgets are also simpler than angular queries, involving only the selection of a maximum angle that an item can make, with an implied minimum of zero. This interface provides greater simplicity, at the expense of reduced expressivity: angular brushes cannot be used to restrict changes to falling within two non-zero angular values. Comparison of the relative strengths and weaknesses of these two approaches for specifying queries on relative values might be an interesting area for further investigation.

Performance issues

A series of measurements of display time relative to total processing time indicated that over 75% of query processing time was spent in query evaluation, as opposed to the display of query results. Thus, optimization efforts focused on improving search performance are likely to be most fruitful. Side-by-side tests of query evaluation algorithms were conducted as a means of identifying efficient algorithms and understanding the reasons for those efficiencies.

Query evaluation algorithms

The naive approach to timebox query evaluation follows directly from the definition of the problem: for each of the item in the data set, examine all of the time points in all of the timeboxes in a query.

The conjunctive nature of timeboxes leads directly to the some improvements on this scheme. If some item fails to meet the constraint for a timebox at some time,

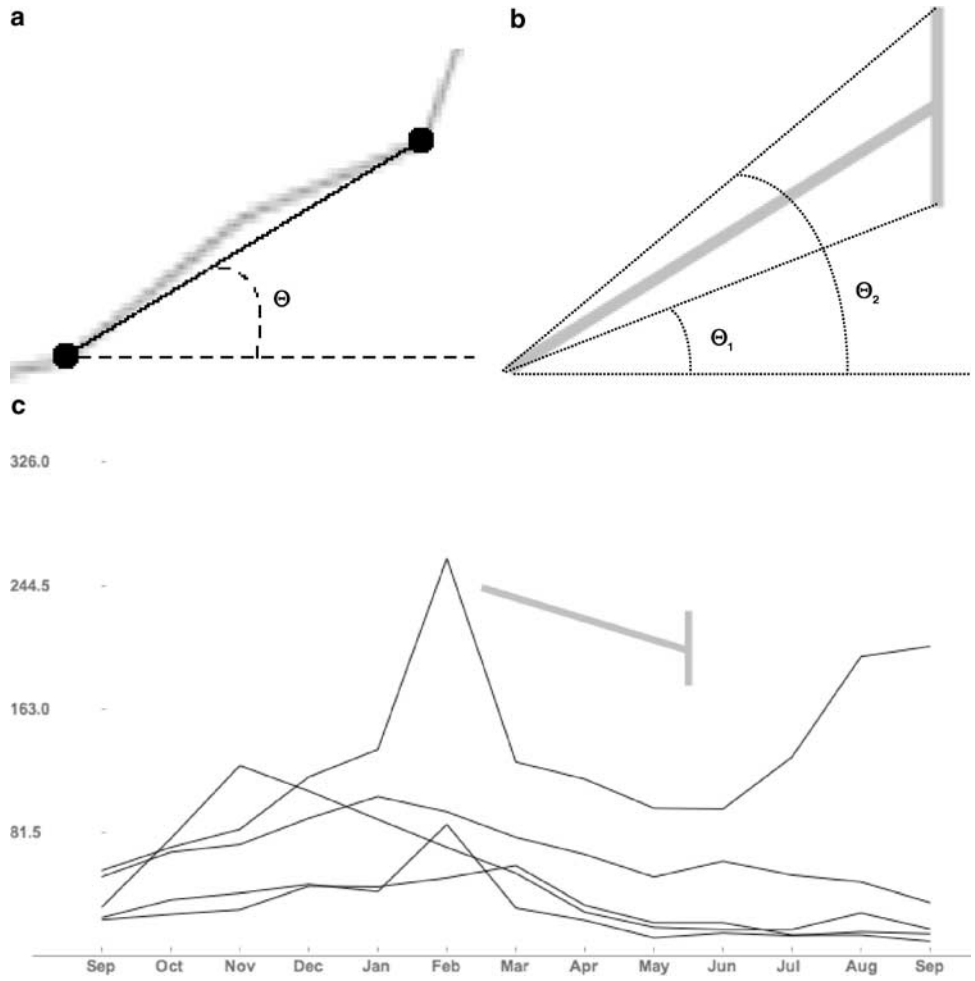

Figure 9 Angular queries enable users to find items with similar slopes over several time periods. (a) The motivation for angular queries. As a time series changes, its slope creates an angle with the horizontal. Queries that specify angles can be used to find items with desired rates of change. (b) The angular query widget. The endpoints of the vertical bar on the right allow specification of a minimum and maximum desired angle, relative to the start point on the left. (c) In this angular query, the user finds items that decreased steadily over three time periods.

we can immediately conclude that the item does not match the query, even if we have not completely processed the range of times covered by the query. Thus, processing of any (item, timebox) pair can stop as soon as one value outside of the given range is encountered. This is equivalent to the familiar programming language shortcut used in evaluation of conjunctive conditionals.

Additional optimizations can be applied to conjunctive queries. When a new timebox is added to an existing query, the only items that must be checked are those that met the constraints of the original query. Analogous logic can be applied when deleting a timebox.

Techniques from computational geometry provide an alternative approach to timebox query evaluation. A set of n time series profiles, each containing measurements for each of m time points can also be interpreted as a set of mn points in a two-dimensional space. Each of these mn points is associated with one of the n profiles, such that each profile has exactly one associated point for each time point. Under this interpretation, a timebox can be seen as a two-dimensional orthogonal range query – a query aimed at identifying the points that fit inside the rectangular region covered by the query.

Thus, to process a timebox query, we start by identifying the points that fall within this query. For each of these points, we increment a counter associated with the timebox and the time series to which that point belongs. When all of the points that are processed, the items that have a count for the timebox that is equal to the width of the timebox are the matches. This approach can be extended to conjunctive queries containing multiple timeboxes by simply maintaining a separate counter for each (item, timebox) pair.

Implementation of geometric methods requires an appropriate index for efficient handling of the range queries. The two possibilities considered in the current

evaluation were orthogonal range trees and a grid approach, which places items in a regularly sized buckets.[11]

Comparison

Comparison on simulated data can be used to build a better understanding of the merits of the various approach. To test for the effects of the number of items in the data set, data sets containing between 100 and 50,000 items with 100 time points each were created. To test the effects of the number of time points in the data set, data sets with 100 items and between 100 and 10,000 time points were used. All data sets were created pseudo-randomly. These data sets were subjected to suites of similarly pseudo-random queries, involving timebox creation, scaling in either or both directions, resizing in either or both directions, and deletions.

Four algorithms were tested: optimized sequential search (''Seq''), geometric search with an orthogonal range tree (''Orth''), and two variants of geometric search with a grid index. The ''Grid-20'' data set had approximately 20 items/bucket, while the ''Grid-100'' data set had approximately 100 items/bucket.

Sequential scan outperformed the geometric alternatives in all but one of the test cases. For the test involving 100 items and 100 time points, the orthogonal search outperformed the other alternatives. For each test case, single-factor ANOVAs comparing the four algorithms revealed strongly significant differences ($P<0.01$) (Figure 10).

For the tests that varied the number of time points, search time was almost completely insensitive to the different test conditions. This might be explained by an advantage in the number of points that must be processed for a given timebox query. Figure 11 shows a timebox that spans eight time points, along with a time series that falls within the timebox for seven of those eight points. To determine whether this item satisfies the constraints of the timebox, the sequential search only needs to examine the first two values in the given time range: once it is determined that the second value falls outside of the timebox, there is no need to examine any of the remaining values. The geometric approach must examine all of the seven points that falls inside of the timebox, before concluding that the item fails to match the constraints. As a result, the sequential search uses a significantly smaller number of checks to determine whether or not an item in the data set matches the timebox.[12] This advantage is crucial to the success of the sequential search.

The superior performance of the sequential search algorithm may be a result of the ''dimensionality curse'' – the inherent difficulty of searching in high-dimensional cases. As a time series data set with n time points can be viewed as a set of n dimensional vectors, a timebox query over that data set can be considered to be a query in n-dimensional space. Recent analyses of index structures for searches in high-dimensional space have shown that sequential scans outperform indexed searches, even for moderate (<20) dimensionalities.[13–15] Although additional optimizations to reduce constant factors are certainly possible, these results suggest that sub-linear search performance for timebox queries may be difficult, if not impossible, to achieve.

Evaluation

Design studies

Two design studies were conducted to verify that users understood timeboxes and to refine interface features. In

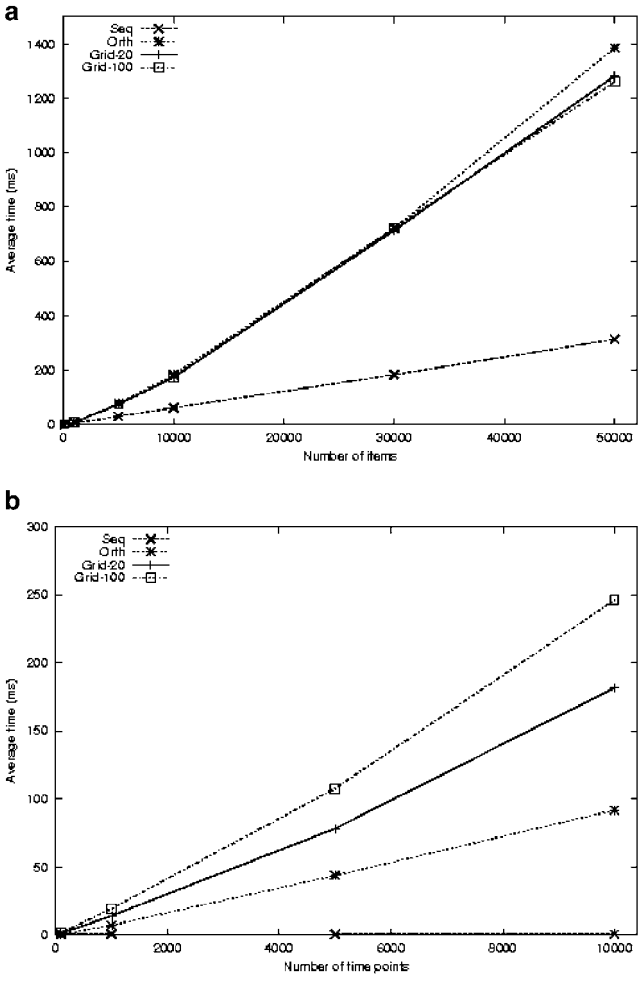

Figure 10 Empirical comparisons of query evaluation algorithms: (a) Average query completion times (ms) for data sets with 100 time points and 100, 1000, 5000, 10,000, 30,000, and 50,000 items. (b) Average query completion times (ms) for data sets with 100 items and 100, 1000, 5000 and 10,000 time points. In all but one of the cases, the sequential search ''seq'' outperformed both the orthogonal range tree ''Orth'' and the bucketed (''Grid-20'' and ''Grid-100'') geometric searches. For the data set composed of 100 items and 100 time points, orthogonal queries outperformed sequential search. In all cases, single-factor ANOVAs for each test case revealed statistically significant ($P<0.01$) differences between the algorithms.

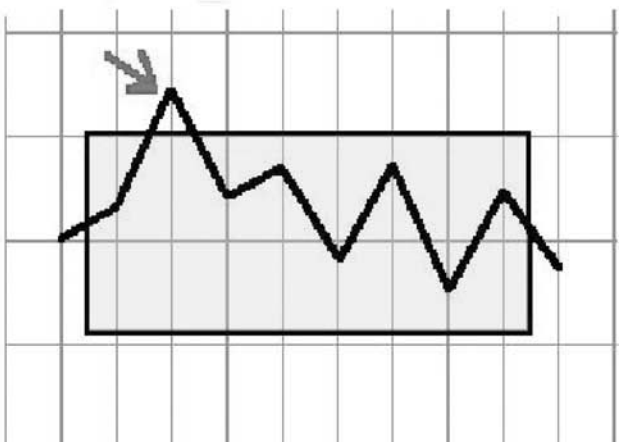

Figure 11 A timebox query demonstrating the advantage that sequential processing has over geometric methods. For this timebox that spans eight time points, sequential processing can stop after the second time value is identified as falling outside of the timebox. However, the geometric approaches must examine every point that falls within the timebox.

all, 24 Computer Science students (undergraduate and graduate) completed tasks using timebox queries and semantically equivalent input mechanisms: fill-in and paired range slider (one for time and one for value) interfaces. The first study involved comparison of all three input modes, with a graph overview display (Figure 2) for output results. Tasks in this study were fully specified (''During days 22–23, are there more stocks between 69–119, 59–109, or 49–99?''). The second study involved comparison between timeboxes and form fill-in queries – both with graphical output – and form fill-in queries with output presented in a tabular spreadsheet format. Tasks in this study were more open-ended (''Find stocks that traded in a 20 range for at least three consecutive time periods.''). Both studies used a data set of mock stock price data, with one time-varying quantity (price). Task completion time was the dependent variable.

In the first study, the form fill-in interface was fastest, followed by range sliders and finally timeboxes. All differences were statistically significant. There were no statistically significant differences between the three interfaces in the second study.

These studies demonstrated that users could understand and work with the timebox input and the graph display. They yielded insights into the strengths and weaknesses of the timebox interface. For example, when precise values were to be entered the form fill-in boxes were significantly easier than sizing and positioning a two-dimensional rectangular widget. These results provided support for design decisions that had been made prior to the study, such as allowing typed in values to adjust timeboxes and a mode that fixes *y*-values when timeboxes are dragged horizontally.

Feedback from study participants, casual users, lab visitors, and long-term collaborators helped refine our TimeSearcher interface design. In particular, our colleagues in molecular biology have successfully used TimeSearcher to examine time series and linear sequence data sets.

DNA microarray analysis: programmed cell death

Recent advances in DNA microarray technology have provided geneticists with the ability to examine expression levels of thousands of genes under varying circumstances.[16] Numerous published reports of microarray data have used the examination of changes in gene expression levels over time to examine the effects of various stimuli on genetic expression.

Analyses of the microarray data generally are conducted via some sort of mathematical grouping of genes with similar expression profiles.[17–19] Clustered expression profiles are often displayed with two-dimensional layouts that use coloring to display the expression levels of each sample, with bright-green indicating relatively underexpressed genes and bright-red indicating genes with relatively high levels. These displays are very useful for condensing significant amounts of information in a display that helps highlight gross trends and similarities between clusters. However, they generally suffer from the drawbacks of other static displays: interactive querying and exploration are not supported. The Hierarchical Clustering Explorer[19] addresses many of these problems by combining filters for minimum similarity and detail display with alternative displays showing pairwise similarities between expression profiles and the ability to compare clusters computed from different algorithms.

TimeSearcher's dynamic query tools are well-suited for expressing queries aimed at identifying genes with particular expression profiles. Researchers have been using TimeSearcher to analyze their microarray data sets. This collaborative effort has led to a number of suggestions and design ideas that have informed the work described in this paper.

The process of controlled destruction and elimination of cells – known as programmed cell death (PCD) – is of interest to biologists for a variety of reasons. As a genetically controlled process, PCD involves complex interactions between many genes. Furthermore, the absence of cell death may be related to the uncontrolled proliferation of cells associated with cancerous tumors. Studies of cell death in flies, worms, humans, and other organisms have identified a variety of genes that are involved in the control of PCD. Furthermore, many of the genes involved in this process appear to be similar in these organisms–in other words, the relevant genes have been conserved.[20]

Previous studies have identified the gene *E93* as a pivotal factor in programmed cell death in *Drosophila melanogaster* – the common fruit fly.[21] Subsequent microarray experiments examined changes in gene expression level at 6 and 12 h after a rise in hormone

level that triggers cell death. These analyses identified significant changes in expression levels for genes known to be involved in cell death.[22]

TimeSearcher has been used to analyze a microarray data set that provides finer-grain insight into the processes behind PCD. Specifically, this data set contains expression profiles for 6,8, 10, 12, and 14 h after the increase in hormone level associated with the onset of PCD. This analysis is based on the hypothesis that genes that have profiles similar to E93's profile might also be involved in cell death. Similarly, genes that show increases in expression level after *E93's* expression increases might be regulated by *E93*–in other words, *E93* might be a factor that contributes to the expression of these genes.

The use of TimeSearcher to explore this line of investigation begins with the use of the text search box to find the *E93* sample in the database. The drag-and-drop query-by-example tool is then used to create a query identifying those items that are similar to E93's profile. As the 10 and 12-h measurements – corresponding to the interval before the second peak in ecdysone levels – are most interesting,[23] the boxes for the 6, 8, and 14 h samples are eliminated. The timebox for the 12-h time point is adjusted to include only those genes with a more pronounced increase in expression level – the box is moved up to remove lower values, and expanded to include a higher range of values (Figure 12).

The leaders & laggards facility was then used to identify genes that have this same increase in expression level at a later time point – specifically, between 12 and 14 h. This leads to a set of under 100 laggards – genes that might be regulated by *E93*.

Alternative approaches included shifting the paired timeboxes to look at genes with increases in transcription levels at earlier time points, and using the query inversion facility to identify genes with expression levels that decrease when *E93*'s expression is increasing.

The use of TimeSearcher for the analysis of the cell death data played an important role in the identification of novel results.[24] Collaboration with the biologists involved numerous design discussions, which generated several ideas for TimeSearcher functionality. Some of these features have been implemented, while others present interesting possibilities for future work.

Currently implemented features that resulted at least partially from these discussions include leaders & laggards, support for multiple time-varying attributes, and query inversion. Leaders & laggards was suggested early in the discussions as potentially useful for identifying regulatory relationships, as described above. Support for multiple time-varying attributes was proposed as useful for simultaneous display and querying of data collected under two different conditions: naturally occurring (''wild-type'') flies and mutated flies. Query inversion was proposed as a tool for identifying transitions that were contrary to previously identified trends of interest.

Analysis of the cell death data set highlighted the potential utility of integrating TimeSearcher with other visualization tools. As coordination of multiple visualizations has been shown to decrease task performance time and increase user satisfaction,[25] the use of TimeSearcher in conjunction with other visualization tools might improve comprehension and utility of the visualizations. For example, an ongoing effort has investigated the possibility of displaying Gene Ontology information in a hierarchical treemap display.[26,27] A coordinated visualization might use a treemap display to highlight the genes that matched a particular query. This would provide an immediate graphical perspective on the similarities between genes in the result set: the presence of multiple

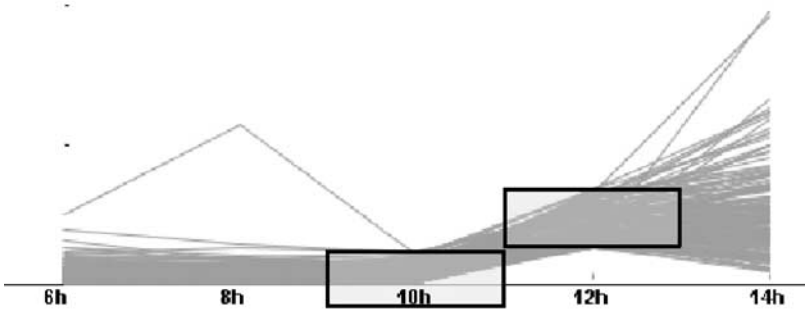

Figure 12 TimeSearcher has been used to analyze gene expression patterns during programmed cell death (PCD) in *Drosophila melanogaster*. A microarray dataset involving 3225 genes at five time points was analyzed to find genes that have expression patterns similar to those of PCD gene E93. This query identifies genes with that are roughly similar to E93 at 10 and 12 h, key points during PCD. Starting with timeboxes centered around the values of E93 at these time points, the 12 h timebox has been shifted up, to eliminate smaller increases in expression levels. This timebox has also been increased in height, in order to include some very sharp increases in expression level that might not have been included in the original timebox.

similar genes would lead to a tight cluster of highlighted genes. However, if the result set of a given query contained highly dissimilar genes, the highlights would be scattered throughout the treemap.

Nucleotide sequence data

The interpretation of timeboxes and TimeSearcher as tools for querying time series data sets places an unnecessary limitation on the applicability of these ideas. In fact, there is nothing in the timebox model, or in the TimeSearcher application that is restricted specifically to time series data. The only requirement is that data sets involve measurements taken at discrete intervals along some linearly ordered dimension.

Nucleotide sequences provide a particularly interesting example of the application of TimeSearcher to linear dimensions other than time. Specifically, short sequences of nucleotides (A,G,C, and T) can be aligned and statistics regarding the frequencies at which different patterns appear in different positions in these sequences can be calculated. TimeSearcher can then be used to find patterns that have desired frequency profiles. The use of TimeSearcher for this purpose complements existing statistical approaches towards identification of these subsequences.[28,29]

TimeSearcher has been used to identify consensus branch site splicing signals in the plant *Arabidopsis thaliana*. These secondary signals in the RNA transcripts of genes help to determine which sequences (introns) are removed from RNA. The segments that remain are known as exons. The data set being used for this purpose was generated from the genomic sequences surrounding 8550 internal exons that were internally truncated and aligned with respect to their boundaries. This data set contains the normalized frequencies of each of the 1024 possible pentamers – sequences of five nucleotides – at each of 192 possible positions.

Figure 13(a) shows a data envelope overview of the whole data set. Two peaks, indicating the boundaries between the exon in the middle and the introns on the ends, are immediately apparent. These peaks represent well-known conservation of sequences at splice sites – the boundaries between exons and introns.

As branch points are found approximately 25–30 positions upstream (before) the end of an intron, sequences that might include branch points can be found by searching for pentamers that are frequently found 25–30 positions before the end of an intron and infrequently found elsewhere in the intron. To identify candidate splicing signals, a query using two timeboxes is used. One component of the query will identify those pentamers that are frequently found before the exon–intron boundary. The second identifies pentamers that are infrequently found elsewhere with the intron (Figure 13(b)).

Taken together with domain knowledge of the expert user, these results can be used to extend known consensus sequences. In this case, the analysis was used to extend the previously identified consensus branch point sequence from five to six bases.[30]

Related work

Much of the recent research involving interactive tools for time series data has been focused on display issues, with relatively little attention paid to creation and manipulation of queries over these data sets. Query facilities in these tools are generally limited to selecting and zooming in on time periods of interest. Examples include DiskTrees and TimeTubes,[31] which provides an interesting model for the use of circles to display multiple attributes from a hierarchical data set.

Other efforts have addressed the challenges of supporting multi-scale and periodic views. Recursive patterns, an early visualization technique, provide dense displays of data divided hierarchically into finer-grained time periods (year, month, week, etc.).[32] Spiral visualizations uses a circular metaphor to display the periodicity of some data sets.[33] Extending timeboxes and TimeSearcher to handle these data sets is an interesting area for future work.

Several tools have addressed various aspects of the challenge of specifying queries over time series data. MIMSY uses traditional GUI widgets including text entry fields and pull-down menus,[34] to search for trends of interest in stock data. MIMSY supports a number of aggregate operators, and support for relative changes. Query processing is handled in a traditional batch mode.

QuerySketch is an innovative query-by-example tool that uses an easily-drawn sketch of a time series profile to retrieve similar profiles, with similarity defined by Euclidean distance.[35] Designed for simplicity and ease-of-use, QuerySketch does not support editing of existing queries. Spotfire's Array Explorer 3 supports graphically editable queries of temporal patterns in microarray data.[36] Queries are dynamically modified by moving discrete value markers at each time point. The limitation of each query point to a single time instance complicates the expression of queries involving values that remain relatively unchanged for a period of time.

Patterns uses a set of graphic primitives and operators to specify patterns of interest in time series data.[37] Query primitives can be used to search for intervals during which values are rising, falling, flat, contained within a given threshold, straight (constant slope), concave, or convex. Although performance details are not provided, the Patterns query language is powerful and flexible.

The ancestry of timeboxes can be traced back to one-dimensional range sliders, which extended traditional GUI sliders. Range sliders allowed users to adjust values from both ends (instead of only one end), and to move the entire range of interest by dragging the middle of the slider.[38] Similar techniques have been used for selection and filtering of items in parallel coordinates displays, both with implicit and explicit sliders.[39,40]

These approaches share the common limitation of controlling only one or two dimensions at any given

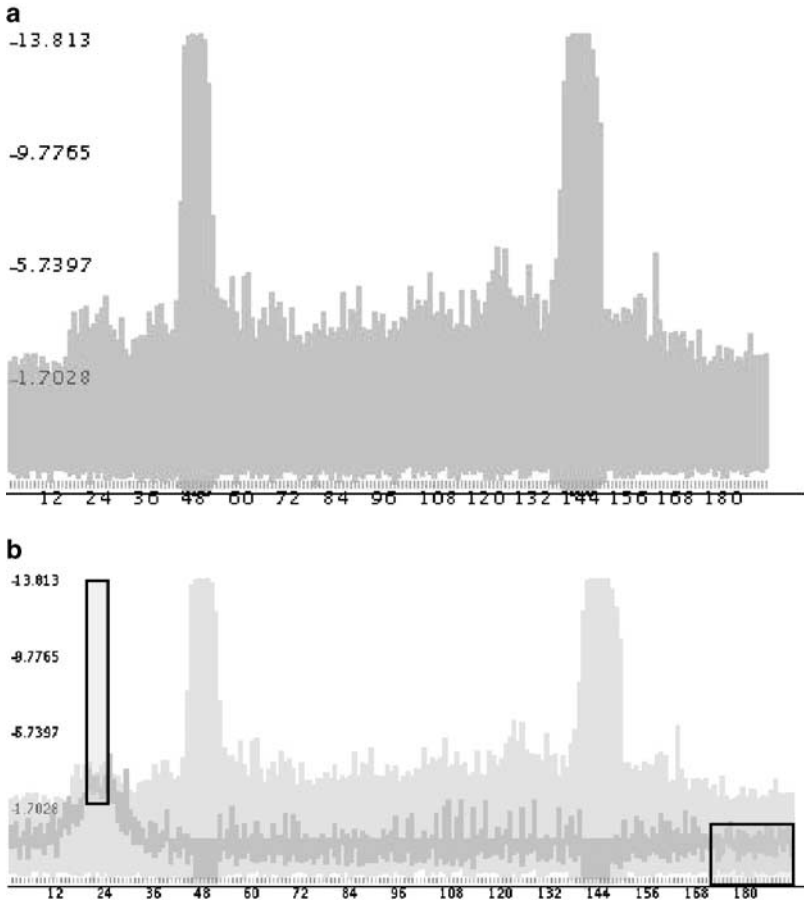

Figure 13 Application of TimeSearcher to the problem of identifying branch sites in DNA sequences. In this application, the horizontal access is position in the DNA sequence, rather than time. A data set of 8550 aligned sequences contains frequencies for each possible pentamer (5-base string) at each of the 192 positions in each of these sequences. (a) Data envelope overview of pentamer frequency distributions in all 8550 aligned sequences from Arabidopsis thaliana. The peaks are the highly conserved (and thus extremely frequent) boundaries between the introns on the ends and the exons in the middle. (b) Timebox query for identification of branch points. This query identifies pentamers that have high frequencies frequencies at a specific region within introns (the branch site) and lower frequencies elsewhere within introns. Items that match this query are potential branch points.

time. To adjust constraints on multiple dimensions, users must adjust multiple brushes controls individually. For high-dimensional data sets, this can get tedious. Two-dimensional widgets have been suggested as an approach to improve this situation. These widgets might be used to specify single points for two variables or to select a range in 2D space.[38]

TimeSearcher's ''graph overview'' is visually similar to Inselberg's parallel coordinates,[41] and the angular query tool is similar to Hauser's ''angular brushes''.[9] Timeboxes and TimeSearcher differ from parallel coordinates in their interpretation of the ordering of the dimension axis. For linearly ordered or time series data, adjacency of dimensions indicates adjacency in the given order. This is not the case with parallel coordinates, which have arbitrarily ordered dimensions. Thus, although timebox queries might be useful for parallel coordinates, the patterns that they reveal might be arbitrary functions of the ordering of dimensions: transitions between measurements can easily be artifactual. Timeboxes are also based on the assumption that each measurement is made on a common scale, and that all values will fall between some global minimum and maximum value. This assumption may not hold for parallel coordinates.

A vast body of research in the data mining and algorithmic literature has focused on the identification of time series data. Much of this work has involved finding subsequences of a data set that are similar to a query sequence As the query is simply another time series sequence, these approaches generally do not address questions of query specification and user interfaces.

One exceptions is the Shape Definition Language (SDL),[2] which provides a syntax and grammar for specifying a series of desired transitions in a time series profile. SDL's use of a simple syntax leads to easily

understood queries of limited power. Thus, users can ask for an ''upwards'' transition, but they cannot specify a magnitude of the transition. TimeSearcher's angular queries can be used to specify magnitudes, but at the potential cost of additional manipulation. Evaluation of the relative merits of these approaches might be an interesting topic for a future study.

Another interesting user interface suggestion for identifying interesting time series involved the use of relevance feedback to help users refine the notion of ''similarity'' to best meet their needs.[42]

Future work

Extension and formalization of the timebox query model

Although angular queries, variable-time timeboxes, and support for queries over multiple time-varying attributes add significant expressive power, there are numerous other possible extensions that might be explored and implemented.

For example, VTTs are based in a model of placing a timebox within a larger region that provides additional constraints. This model can easily easily be generalized to support other extensions that increase the expressivity of the timebox model. Queries with variability in value instead of time – VVTs – might be formed by providing vertical variability, instead of horizontal. Vertical and horizontal ranges might be combined to provide queries that support variability in both time and value.

Some queries might involve times and values that are specified relative to each other, rather than in terms of any absolute values. For example, a stock analyst might be interested in a query that stated: ''Find items that traded within a 20 range for 3 days, and then had a rise in price of at least 50% that was maintained for 1 week''. The starting and ending values of the item and the exact start points of these intervals, remain unspecified: any times or values that meet these constraints would be acceptable.

Other queries that might be interesting in some domains include queries involving maximal periods (''Find the longest interval consisting only of increases in value, for a given item''), aggregate functions (''Find items that have average prices in a given range during a given time period''), similarity queries (''Find items that are similar to X during a certain interval''), and other, more general queries: (''Find items that have periods lasting five intervals long that contain at least two upwards changes and no more than one downward change''). Numerous other extensions are possible.[12]

Design and implementation of these extended queries involves several challenges. Appropriate query widgets and interactions must be supported, along with output displays that can be used to determine why a given item matched a given query.

For many of the more open-ended extensions, query processing may be a concern: if queries cannot be evaluated within the 100 ms constraint usually associated with dynamic query interfaces, other strategies might be needed. For example, long-running queries that are deemed significantly useful might be evaluated upon demand through an explicit ''execute query'' button. Identification of algorithms and data structures that increase the speed of query execution might also help with this challenge.

The timebox query model and the extensions described here provide the beginnings of a special-purpose visual query language. Formal models that describe the semantics of these query tools would ease reasoning about the expressivity of the language and provide a framework for consistent definition of new query tools.

Extensions to the query model should be driven by an understanding of user needs and abilities that would be addressed by the new queries. Queries that provide increased power at the expense of user confusion in execution or interpretation may not be worthwhile.

Extension of the TimeSearcher tool

Many interesting time series data sets are very large, both in terms of the number of items, and the number of time points in each item. Scaling the timebox/TimeSearcher model to accommodate larger numbers (perhaps $O(10^6)$ items or time points) would require improvements to search algorithms and the rendering portion of the system. Alternatively, larger data sets might be randomly sampled or mathematically clustered into smaller sets of manageable size.

Long time series present particular problems for query specification and display. Screen space limitations of approximately 1000 horizontal pixels limit displays to time series of a few hundred time points. Scrolling displays that can be used to pan through long time series would likely suffer from the lack of context and slow execution often associated with scrolling.

Zooming facilities that could be used to display long time series at varying levels of scale and detail might be more promising. These views might be presented in an''overview+detail'' fashion, with a compressed overview presented alongside raw data. Alternatively, distortion techniques might be used to present areas of interest in full detail and peripheral areas in a compressed display.

Appropriate display and handling of missing values will be necessary to accommodate many data sets. The easiest approach to handling missing data may be to use some sort of average of surrounding data points, but this might lack mathematical validity. Alternatively, missing data points might be displayed differently - perhaps as a dashed line connecting the two valid points surrounding a missing point. Query processing might be adjusted to handles these data points correctly. For example, a missing value might be interpreted as matching any query.

As currently implemented, TimeSearcher is a generic, domain-independent tool. For some domains, additional

facilities for data manipulation and management would provide substantial additional value. This was a recurring theme in our work with microarray biologists, who are regularly faced with the challenges of coalescing data from multiple repetitions of a single experiment. This processes involves normalization, averaging of results, elimination of items that are not reliably present, and performance of statistical tests that characterize the results. Additional features that support these or other domain-specific tasks are likely to present interesting research challenges.

Conclusions

Despite the wide range of data sets and domains that make extensive use of time series data, there has been relatively little work to date involving dynamic queries for specifying constraints on time series data sets. The timebox query model provides the basis for an exploration of issues associated with interactive queries on time series and other linearly sequence data sets. TimeSearcher is an information visualization tool that takes timeboxes as a starting point and extends them with additional queries that provide expressive power.

Ongoing use of TimeSearcher by participants in our case studies and other users has led to encouraging feedback. Additional work on increasing the range of queries that can be expressed and generally increasing the utility of the tool will present further research challenges and possibilities for effective use by motivated users with real data analysis problems.

The success of the case studies stands in contrast to the relatively ambiguous results from the design studies. This discrepancy appears to have been caused by differences between the well-defined tasks used in the design studies and the more open-ended explorations conducted during the case studies. Thus, TimeSearcher presents a demonstration of the challenges of evaluating interfaces that support exploration of large data sets. As such, TimeSearcher might be useful as a platform for investigation of evaluation strategies that bridge the gap between the narrowly defined tasks often used in controlled studies and the vague tasks that are most interesting to users of information visualization tools.

Acknowledgments

Thanks to Eric Baehrecke and Steve Mount for their help with the case studies, and to the anonymous reviewers for their helpful comments. Harry Hochheiser was supported by the America Online fellowship in Human–Computer Interaction.

References

1 Agrawal R, Lin K, Sawhney HS, Shim K. *Fast similarity search in the presence of noise, scaling, and translation in time-series databases*. The VLDB Journal, 1995; 490–501.
2 Agrawal R, Psaila G, Wimmers EL, Zait M. *Querying shapes of histories*. In Proceedings of the 21st International Conference on Very Large Databases, (Zurich, Switzerland)1995; 502–514.
3 Agrawal R and Srikant R. *Mining sequential patterns*. In Yu PS, and Chen ALP, (eds). Proceedings of the 11th International Conference on Data Engineering, ICDE, (Taipei, Tawian). IEEE Press: New York, March 1995; 3–14.
4 Faloutsos C, Ranganathan M, Manolopoulos Y. *Fast subsequence matching in time-series databases*. In: Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, (Minneapolis, Minnesota), ACM Press: New York, May 1994; 419–429.
5 Hochheiser H and Shneiderman B. *Interactive exploration of time series data*. In: Proceedings, Discovery Science 2001, (Washington, DC), Springer-Verlag: Berlin, November 2001.
6 Bederson B Grosjean J Meyer J. *Toolkit design for interactive structured graphic*. Technical Report HCIL-2003-01,CS-TR-4432, and UMIACS-TR-2003-03, University of Maryland, Human–Computer Interaction Lab, Department of Computer Science, and Institute for Advanced Computer Studies, 2003.
7 North C and Shneiderman B. *Snap-together visualization: a user interface for coordinating visualizations via relational schemata*. In: ACM Advanced Visual Interfaces 2000, (Palermo, Italy) ACM Press: New York, 2000; 128–135.
8 Keogh E Hochheiser H and Shneiderman B. *An augmented visual query mechanism for finding patterns in time series data*. In: Proceedings of the Fifth International Conference on Flexible Query Answering Systems, Lecture Notes in Artificial Intelligence, (Copenhagen, Denmark), Springer-Verlag: Berlin, 27-29 October 2002; 240–250.
9 Hauser H Ledermann F and Doleisch H. *Angular brushing of extended parallel coordinates*. In: Proceedings, IEEE Symposium on Information Visualization, (Boston, MA), IEEE Press: New York, October 2002.
10 Winkler S. [WWW document] http: //stats.math.uni-augsburg.de/CASSATT/index.html (accessed 14 May 2003).
11 de Berg M, van Kreveld M, Overmars M Schwarzkopf O. *Computational Geometry: Algorithms and Applications*. Springer-Verlag: Berlin, 2000; 379pp.
12 Hochheiser H. Interactive graphical querying of time series and linear sequence data sets. Ph.D. thesis, Department of Computer Science, University of Maryland, College Park, MD, May 2003.
13 Beyer K, Goldstein J, Ramakrishnan R, Shaft U. *When is ''nearest neighbor'' meaningful?* In Beeri C, Buneman P, (eds). 7th International Conference on Database Theory (ICDT '99), (Jerusalem, Israel) Lecture Notes in Computer Science, Vol. **1540**, Springer-Verlag, Berlin, January 1999; 218–236.
14 Shaft U, Goldstein J and Beyer K. *Nearest neighbors query performance for unstable distributions*. Technical Report TR1388, Computer Sciences Department, University of Wisconsin, October 1998.
15 Weber R, Schek H and Blott S. *A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces*. In: Proceedings of the 24th International Conference on Very Large Data Bases, VLDB, (New York City, NY) 1998; 194–205.
16 Hamadeh H and Afshari C. *Gene chips and functional genomics*. American Scientist, 2000; **8**: 508–515.
17 DeRisi J, Iyer V and Brown P. *Exploring the metabolic and genetic control of gene expression on a genomic scale*. Science, 1997; **278**: 680–686.
18 Eisen M, Spellman P, Brown P, Botstein D. *Cluster analysis and display of genome-wide expression patterns*. Proceedings National Academies of Science USA, 1998; **95**: 14863–14686.
19 Seo J and Shneiderman B. *Understanding hierarchical clustering results by interactive exploration of dendrograms: a case study with genetic microarray data*. IEEE Computer, 2002; **35**: 80–86.
20 Baehrecke E. *How death shapes life during development*. Nature Reviews Molecular Cell Biology, 2002; **3**: 779–787.
21 Lee C-Y Wendel D Reid P Lam P Thummel C and Baehrecke E. *E93 directs steroid-triggered programmed cell death in Drosophila*. Molecular Cell, 2000; **6**: 433–443.
22 Lee C-Y Clough E Yellon P Teslovich T Stephan D and Baehrecke E. *Genome-wide analyses of steroid-and radiation-triggered programmed cell death in Drosophila*. Current Biology, 2003; **13**: 350–357.

23 Baehrecke E. *Steroid regulation of programmed cell death during Drosophila development*. Cell Death and Differentiation, 2000; **7**: 1057–1062.
24 Clough E, Lee C-Y, Hochheiser H, Shneiderman B and Baehrecke E. *Temporal analyses of genome-wide transcription during steroid-triggered programmed cell death in Drosophila*. 2003, In preparation.
25 North C and Shneiderman B. *Snap-together visualization: Evaluating coordination usage and construction*. International Journal of Human–Computer Studies, 2000; **53**: 715–739.
26 Baehrecke E Dang N Barbaria K Shneiderman B. *Visualization and analysis of microarray and gene ontology data with treemap*. 2003, In preparation.
27 Shneiderman B. *Tree visualization with tree-maps: 2-d space-filling approach*. ACM Transactions on Graphics, 1995; **11**: 92–99.
28 Fairbrother W, Yeh R, Sharp P, Burge C. *Predictive identification of exonic splicing enhancers in human genes*. Science, 2002; **297**: 1007–1013.
29 Mount S, Burks C, Hertz G, Stormo G, White O and Fields C. *Splicing signals in Drosophila: intron size, information content, and consensus sequences*. Nucleic Acids Research, 1992; **20**: 4255–4262.
30 Simpson C, Thow G, Clark G, Jennings S, Watters J and Brown J. *Mutational analysis of a plant branchpoint and polypyrimidine tract required for constitutive splicing of a mini-exon*. RNA, 2002; **8**: 47–56.
31 Chi E Pitkow J Mackinlay J Pirolli P Gossweiler R and Card SK. *Visualizing the evolution of web ecologies*. In: Proceedings of the 1998 Conference Human Factors in Computing Systems, Los Angeles CA, ACM Press: New York, April 1998; 400–407.
32 Keim D. *Pixel-oriented visualization techniques for exploring very large databases*. Journal of Computational and Statistical Graphics, 1996; **5**: 58–77.
33 Carlis J and Konstan J. *Interactive visualization of serial periodic data*. In: ACM Symposium on User Interface Software and Technology, (San Francisco, CA), ACM Press: New york, November 1998; 29–38.
34 Roth W. *MIMSY: a system for analyzing time series data in the stock market domain*. Master's thesis, Department of Computer Science, University of Wisconsin, 1993.
35 Wattenberg M. *Sketching a graph to query a time series database*. In: Proceedings of the 2001 Conference Human Factors in Computing Systems, Extended Abstracts, (Seattle, WA), ACM Press: New York, March 31-April 5, 2001, 381–382.
36 Spotfire. [WWW document] http: //www.spotfire.com (accessed 14 May 2003).
37 Morrill JP. *Distributed recognition of patterns in time series data*. Communications of the ACM, May 1998; **45**(5): 45–51.
38 Shneiderman B. *Dynamic queries for visual information seeking*. IEEE Software, 1994; **11**: 70–77.
39 Martin A and Ward M. *High dimensional brushing for interactive exploration of multivariate data*. In: Proceedings of the 6th IEEE Visualization Conference, (Atlanta, Ge), IEEE Press: New York, October 29–November 3 1995; 271–278
40 Girardin L and Brodbeck D. *Interactive visualization of prices and earnings around the globe*. In: Interactive Posters, IEEE Symposium on Information Visualization 2001, (San Diego, CA), October 22–23 2001.
41 Inselberg A and Avidan T. *Classification and visualization for high-dimensional data*. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery in Data 2000, (Boston, MA), 2000. ACM Press: New York, 370–374.
42 Keogh E and Pazzani M. *Relevance feedback retrieval of time series data*. In: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval SIGIR '99, (Berkeley, CA), ACM Press: New York, August 1999; 183–190.