

Exploratory Visualization of Data Pattern Changes in Multivariate Data Streams

by

Zaixian Xie

A Dissertation

Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Ph.D. in

Computer Science

by

September 2011

Dr. Matthew Ward, Professor, Dissertation Advisor

Dr. Elke Rundensteiner, Professor, Dissertation Co-advisor

Dr. Robert Lindeman, Associate Professor, Committee Member

Dr. Martin Wattenberg, Google, External Committee Member

Professor Craig Wills, Head of Department

Abstract

More and more researchers are focusing on the management, querying and pattern mining of streaming data. The visualization of streaming data, however, is still a very new topic. Streaming data is very similar to time-series data since each datapoint has a time dimension. Although the latter has been well studied in the area of information visualization, a key characteristic of streaming data, unbounded and large-scale input, is rarely investigated. Moreover, most techniques for visualizing time-series data focus on univariate data and seldom convey multidimensional relationships, which is an important requirement in many application areas. Therefore, it is necessary to develop appropriate techniques for streaming data instead of directly applying time-series visualization techniques to it.

As one of the main contributions of this dissertation, I introduce a user-driven approach for the visual analytics of multivariate data streams based on effective visualizations via a combination of windowing and sampling strategies. To help users identify and track how data patterns change over time, not only the current sliding window content but also abstractions of past data in which users are interested are displayed. Sampling is applied within each single time window to help reduce visual clutter as well as preserve data patterns. Sampling ratios scheduled for different windows reflect the degree of user interest in the content. A degree of interest (DOI) function is used to represent a user's interest in different windows of the data. Users can apply two types of pre-defined DOI functions, namely RC (recent change) and PP (periodic phenomena) functions. The developed tool also allows users to interactively adjust DOI functions, in a manner similar to transfer functions in volume visualization, to enable a trial-and-error exploration process. In order to visually convey the change of multidimensional correlations, four layout

strategies were designed. User studies showed that three of these are effective techniques for conveying data pattern changes compared to traditional time-series data visualization techniques. Based on this evaluation, a guide for the selection of appropriate layout strategies was derived, considering the characteristics of the targeted datasets and data analysis tasks. Case studies were used to show the effectiveness of DOI functions and the various visualization techniques.

A second contribution of this dissertation is a data-driven framework to merge and thus condense time windows having small or no changes and distort the time axis. Only significant changes are shown to users. Pattern vectors are introduced as a compact format for representing the discovered data model. Three views, juxtaposed views, pattern vector views, and pattern change views, were developed for conveying data pattern changes. The first shows more details of the data but needs more canvas space; the last two need much less canvas space via conveying only the pattern parameters, but lose many data details. The experiments showed that the proposed merge algorithms preserves more change information than an intuitive pattern-blind averaging. A user study was also conducted to confirm that the proposed techniques can help users find pattern changes more quickly than via a non-distorted time axis.

A third contribution of this dissertation is the history views with related interaction techniques were developed to work under two modes: non-merge and merge. In the former mode, the framework can use natural hierarchical time units or one defined by domain experts to represent timelines. This can help users navigate across long time periods. Grid or virtual calendar views were designed to provide a compact overview for the history data. In addition, MDS pattern starfields, distance maps, and pattern brushes were developed to enable users to quickly investigate the degree of pattern similarity among different time periods. For the merge mode, merge algorithms were applied to selected time windows to generate a merge-based hierarchy. The contiguous time windows having

similar patterns are merged first. Users can choose different levels of merging with the tradeoff between more details in the data and less visual clutter in the visualizations. The usability evaluation demonstrated that most participants could understand the concepts of the history views correctly and finished assigned tasks with a high accuracy and relatively fast response time.

Acknowledgements

This dissertation and the growth in my knowledge over the last few years owe a great deal to many professors, colleagues, and friends.

First among them is my advisor, Prof. Matthew O. Ward, and my co-advisor, Prof. Elke A. Rundensteiner. They inspired my interest in information visualization research and gave me direction by suggesting interesting problems. It has been my luck to have them as my advisors. Their technical and editorial advice were essential to the completion of this dissertation. I express my sincere thanks for their support, advice, patience, and encouragement throughout my graduate studies. Their persistence in tackling problems, confidence, and great teaching will always be an inspiration.

My thank goes to the members of my Ph.D. committee, Prof. Robert W. Lindeman and Dr. Martin Wattenberg, who provided valuable feedback and suggestions to my dissertation proposal talk and dissertation drafts. All these helped to improve the presentation and content of this dissertation.

I would like to thank all previous and current members in Xmdv group during my stay at WPI for their valuable discussions on my research work. The friendship of all the other pervious and current ISRG members is much appreciated. They have contributed to many interesting and good-spirited discussions related to this research. I thank the wonderful professors in the Computer Science department for both their serious lectures and casual chats. I thank the system support staff in our department for providing a well-maintained computing environment and utilities for my research needs.

Finally, I would like to thank my wife, Dr. Mingzhu Wei for her understanding and love in my life. Her support and encouragement was in the end what made this dissertation possible. My parents receive my deepest gratitude and love for their dedication and the many years of support during my studies. My sincere gratitude goes to my parents-in-law because they provided selfless help on taking care of my son when he is very young. The

final but not the least thanks go to my son, Hansen. His smile and crying were always mixed with the writing of this dissertation, but were a big support.

This work was funded by the NSF under grants IIS-0119276, IIS-0414380, IIS-0812027, and CCF-0811510.

Publications

Aspects of the work describe in this dissertation feature in the following publications:

- Z. Xie, S. Huang, M. O. Ward, and E. A. Rundensteiner. Exploratory visualization of multivariate data with variable quality. *Proc. IEEE Symposium on Visual Analytics Science and Technology*, pages 183–190, 2006.
- Z. Xie, M. O. Ward, E. A. Rundensteiner, and S. Huang. Integrating data and quality space interactions in exploratory visualizations. *Proc. 5rd Intl Conf on Coordinated & Multiple Views in Exploratory Visualization*, pages 47–60, 2007.
- Z. Xie, M. O. Ward, and E. A. Rundensteiner. Exploring multivariate data streams using windowing and sampling strategies. *Interacting with temporal data workshop, CHI, 2009*.
- Z. Xie, M. O. Ward, and E. A. Rundensteiner. Visual analysis of multivariate data streams based on DOI functions. Technical Report TR-10-06, Worcester Polytechnic Institute, Computer Science Department, 2010.
- Z. Xie, M. O. Ward, and E. A. Rundensteiner. Visual exploration of stream pattern changes using a data-driven framework. *Proc. 6th International Symposium on Visual Computing*, 6454:522–532, 2010.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	The Sketch of Proposed Solutions	3
1.2.1	A User-driven Approach Based on DOI Function	5
1.2.2	A Data-driven Approach Using Compression	8
1.2.3	History Views with Nested Hierarchical Timelines	11
1.3	Major Contributions of the Dissertation	14
2	Related Work	16
2.1	Representation of Univariate Data	17
2.2	Layout Strategies for Time-series Data	20
2.3	Conveying Multi-dimensional Patterns	24
2.4	Handling Large Scale and Real Time Data in Time-Series Data Visualization	28
3	Data Model and Example Datasets	32
3.1	Data Model	32
3.2	Example Streaming Datasets	33
4	A User-driven Approach Based on DOI Functions	35

4.1	The User-driven Framework Based on Windowing, Sampling and DOI Functions	35
4.1.1	Basic Concepts	35
4.1.2	User-driven framework	36
4.2	DOI Functions	40
4.3	Visualization Techniques	42
4.3.1	Layout Strategies	44
4.3.2	Extension of Layout Strategies to Type PP DOI Functions	50
4.3.3	Integrating Time-series and Multivariate Data Visualizations	51
4.4	DOI Function Interaction Tool	53
4.5	Evaluation 1: User Studies on Layout Strategies	54
4.5.1	Experimental Design	55
4.5.2	Experimental Settings	62
4.5.3	Experimental Results	62
4.5.4	Evaluation Summary and Implications	65
4.6	Evaluation 2: Case Studies on Interaction Techniques and Other Multivariate Visualizations	66
5	A Data-driven Approach to Merging Windows	72
5.1	The Data-driven Framework	72
5.2	Merge Algorithm	74
5.2.1	The State of The Art in Data Compression	75
5.2.2	The Quality Measure for The Merge Algorithm	76
5.2.3	Brute Force	79
5.2.4	Heuristic Merge	81
5.2.5	Stream-based Merge	85

5.3	Visualization of Patterns and Their Changes	88
5.3.1	Juxtaposed Views	88
5.3.2	Pattern Vector and Pattern Change Views	91
5.3.3	A Guide to Choose Visualization Techniques	95
5.4	Evaluation	96
5.4.1	Comparisons among Two Merge Algorithms	97
5.4.2	Comparing Proposed Techniques with Uniform Time Axis	100
6	History Views for History Data Using Nested Hierarchical Timelines	103
6.1	A Framework to Visualize History Data Using Nested Hierarchical Time- lines	103
6.2	Visualization Techniques for Merged Mode	109
6.3	Visualization Techniques for Non-merged Mode	111
6.3.1	Virtual Calendar View	112
6.3.2	Explicitly Conveying Pattern Changes	114
6.4	Usability Evaluation	120
7	Conclusions and Future Work	125
7.1	Conclusions	125
7.2	Future Works	127
	Appendices	129
.1	Pattern Vectors for Linear Models	130
.2	Pattern Vectors for Data Range	131

List of Figures

1.1	A motivating example for the user-driven framework	7
1.2	Showing how DOI functions help reduce visual clutter	7
1.3	A step juxtaposition visualization to show 13 windows	9
1.4	A motivating example for merging time windows	9
1.5	A motivating example of history views.	13
2.1	A candlestick chart	17
2.2	Two kinds of candlesticks	18
2.3	The <i>Spreads</i> to represent data uncertainty [46]	18
2.4	Coded timelines by Bade et al. [3]	19
2.5	Visualizations in <i>WikiReactive</i> [9]	19
2.6	<i>Timebox</i> by Hochheiser and Shneiderman [29]	20
2.7	<i>NameVoyager</i> by Wattenberg [62]	21
2.8	The visualization of DNS traffic by Ren [52]	21
2.9	Spiral layout by Ward and Lipchak [60]	22
2.10	Calendar view by Wijk et al. [64]	23
2.11	Time-series bitmaps [36]	23
2.12	The importance-driven layout by Hao et al. [25]	24
2.13	An example of IVQuery [24]	25
2.14	A TimeWheel by Tominski et al. [58]	26

2.15	Using heatmap to convey multivariate data patterns [69]	27
2.16	A multi-resolution display using DOI function [26]	28
2.17	Distorted timeline by Bade et al. [3]	29
2.18	The <i>timeline</i> in SIMILE project developed by Huynh [31].	30
2.19	<i>WireVis</i> by Chang et al. [13]	31
4.1	The user-driven framework	37
4.2	The screenshot of the implemented system under user-driven framework	39
4.3	Two types of DOI functions	41
4.4	A motivating example for the user-driven framework	43
4.5	A step juxtaposition output using a type PP DOI function	46
4.6	A juxtaposition output for 25 time windows	48
4.7	A step juxtaposition output for 25 time windows	49
4.8	Two grouping approaches in visualizations for type PP DOI functions . .	51
4.9	The embedded views for the sleep data stream	52
4.10	Showing how DOI functions help reduce visual clutter	54
4.11	An example for constructing experimental datasets	57
4.12	The methodology to construct experimental datasets	58
4.13	How to split the whole canvas into multiple cells	60
4.14	A question used in user studies for user-driven framework	61
4.15	The experimental results of the user study for all questions	62
4.16	The response accuracy for datasets having only 3 time windows	63
4.17	Applying step juxtaposition to parallel coordinates	71
5.1	The data-driven framework by merging windows	74
5.2	An example to show how to measure result quality of merge algorithms .	78
5.3	An example to show the result of brute force merge	79

5.4	An example to show heuristic merge algorithm in three passes	81
5.5	An example to show pattern-blind averaging	84
5.6	An example to show stream-based heuristic merge	86
5.7	A motivating example for merging time windows	89
5.8	A pattern outline view	89
5.9	A pattern outline view in grid layout	91
5.10	Three approaches to choosing an even or uneven time axis	92
5.11	A pattern vector view	94
5.12	The experiment result 1 for comparing merge algorithms	98
5.13	The experiment result 2 for comparing merge algorithms	98
5.14	Comparing two brute force algorithms (regular and stream)	99
5.15	Comparing two heuristic algorithms (regular and stream)	99
5.16	A question used in user studies for data-driven framework	101
5.17	The response time for five techniques in data-driven approach	102
6.1	The framework for generating history views	104
6.2	A snapshot of history views in merged mode.	106
6.3	A motivating example of history views.	108
6.4	Merged mode with less levels in merge-based hierarchy.	110
6.5	Merged mode with two levels selected.	111
6.6	Extended 2D grid view	113
6.7	Added a grid level to history view framework	113
6.8	The non-merged mode views generated by MDS algorithm	114
6.9	Removing two days from the original MDS algorithm	116
6.10	The non-merged mode views with a distance map	117
6.11	The pattern brush on the distance map with $\delta = 0.4$	119
6.12	The pattern brush on the distance map with $\delta = 0.5$	120

List of Tables

4.1	A guideline in the user-driven framework	65
6.1	RT and RA of the usability experiment for history views	123

Chapter 1

Introduction

1.1 Motivation

Advances in hardware enable people to record data at rapid rates, e.g., kilobytes or megabytes per second or even higher speeds. Some real application areas require data analysis at the same speed as the data being collected, such as monitoring the health of a collection of people, investments, or computer systems. Moreover, in many cases, the volume of data precludes storage for later analysis. For example, network traffic monitoring involves tracking each packet to identify features of interest, such as bottlenecks and potential intrusions. It is insufficient to detect potential network attack with significant delay; immediately revealing an intrusion will increase the network administrator's effectiveness in handling the hacking. In the areas of database and knowledge discovery, the term *data streams* or *streaming data* has been used to refer to such data that keeps growing and needs to be processed on the fly. Researchers have developed many techniques to manage, query and analyze data streams in real-time [22].

In recent years, people have agreed that visualization can play a critical role in the processes of data analysis and decision-making, since it can help analysts use visual per-

ception to uncover different patterns, such as clusters, associations, relationships, and trends. Moreover, visual analytics can provide an interactive environment that combines human visual cognitive capabilities with high performance computations, thus improving the speed and accuracy at which analysts discover data patterns. However, there has been little work to date on stream visualization.

Streaming data is similar to *time-series* data, which has been identified as one of basic data types [55] in the area of information visualization. In both data types, each datapoint has a time attribute, i.e., a timestamp. One can find a rich set of visualization techniques for time-series data in the literature. As the first step, one thus should consider the direct application of such existing time-series data visualization techniques to streaming data. Hence, the problem of visually exploring data streams can be partially addressed. For example, a continuously expanding line chart can convey the trend of a univariate data stream. However, an important characteristic of streaming data, *unbounded input*, makes this simple approach ineffective and incomplete, as existing visualization techniques for time-series data generally regard the whole dataset as static and assume that all of the data is available before rendering. This is inappropriate for streaming data. Instead, it is necessary to design techniques capable of processing streaming data in a continuous and unbounded fashion.

Two other challenges arise in the visual exploration of both time-series and streaming data that must be addressed in order to help users perform common data analysis tasks:

- (1)**Temporal Visual Mining:** Many data mining tasks can be targeted on time-series data, including the discovery of temporal association rules and pattern evolution [53]. Existing time-series data visualization techniques only support a small fraction of these tasks. In this dissertation, I focus on one important type of temporal mining: how data patterns change over time.

(2) **Multivariate Correlations:** Although a few existing visualization techniques for time-series data try to present the relationships among multiple dimensions, their usefulness is often limited. For example, Hao et al. compute and display the degree of importance for dimensions [25], and present some pre-specified statistical values among dimensions [24]. However, dimensions often have complex interrelations that these methods do not convey. In this dissertation, I aim to combine multivariate and time-series data visualization techniques for streaming data to fill this gap.

The main goal of this dissertation is to present a framework for visually exploring unbounded multivariate data streams online and afterwards. In other words, this goal can be split to two parts: (1) visualize streaming data without too much delay when new data arrives, and (2) provide a history view to help users identify recurring or changing patterns over history. Although different patterns exist in many application areas, this dissertation will focus on how multivariate data patterns change over time. Two examples to be discussed are: (1) the change of the regression line slope; and (2) The movement of a cluster from one time window to the next

1.2 The Sketch of Proposed Solutions

To achieve the goal mentioned in the previous section, I proposed solutions based on the following three aspects:

- **Data Definition and Preprocessing:** A fact is that streaming data is dynamic, since the new data keeps arriving at data processing centers. But most existing visualization techniques are designed for static data. It is true that one possible solution is to update the existing techniques a little bit to adapt them to the dynamic data. For example, shifting a line chart can leave space for new data. However, this design is awkward because it lacks extensibility. For example, if the application requirement

is to convey complex multivariate data patterns, it might be unavoidable to use scatterplots or other visualization techniques that are difficult to shift. In addition, some tasks require the visualization systems to retrieve data patterns using data mining algorithms before generating the final output. To solve these problems, I propose to separate the data preprocessing from the visualizations. In this dissertation, a data preprocessing unit was designed that can retrieve data patterns and detect pattern changes in the streaming data, and wrap a part of the streaming data in a static dataset before sending it to the visualization unit.

- **Visualizations for Current Data:** Although I mainly use existing visualization techniques for static data, the visualization itself still can change when a new time window arrives. Since data changes might be very quick, or there are too many changes in one figure, under certain circumstances, it is possible that the visualizations have changed to the next frame before users fully extract the pattern changes of interest. Thus the design of visualization techniques should reduce users' response time as much as possible.
- **Visualizations for Historical Data:** As well as monitoring the live stream, users might want to explore the past data over a long period, e.g, within the past year. Some common data analysis tasks include: What is the trend for the data patterns within the past year? Are there any cycles or rules for the data pattern changes? Do some time periods have similar data patterns to a specific time window? To help users answer these types of question, data preprocessing algorithms and visualization techniques are needed for history data. For the data preprocessing, I made use of the algorithms for current data but with some adaptations. For example, users might want to observe the data in a hierarchical time structure, so the data model must be modified to accommodate this goal. When designing the visualizations and

interactions, the data can be regarded as static and the restrictions for current data can be ignored. In other words, the historical data can be regarded as regular time series data.

In the remaining part of this section, I will briefly describe the three developed approaches. The first two aim to visualize dynamic streaming data, while the last focuses on history data.

1.2.1 A User-driven Approach Based on DOI Function

This approach aims to visualize the dynamic data as follows. The data in the current window is mixed with those in the past windows and displayed in an interactive view. Datapoints from different time windows mixed within the view are distinguished using different visual attributes, or the data is juxtaposed in an ordered set of views. A degree of interest (DOI) function [21] is introduced to describe the degree of user interest in a particular window. A lower DOI value results in a smaller sampling ratio. This approach works in two ways: (1) Users can choose which windows to show, normally those containing data patterns that users want to compare; and (2) Users can reduce visual clutter by assigning lower DOI values to selected windows. A special consideration during the design of the DOI function was to adapt it to a dynamic context. For example, in highway traffic monitoring, one time window could be 30 minutes, and users want to compare the data patterns between the current window and the previous one. When it is 6:00AM, the two most recent windows are 5:00AM-5:30AM and 5:30AM-6:00AM. After one hour, the two recent windows will be 6:00AM-6:30AM and 6:30AM-7:00AM. Thus, a reasonable way is to refer to the time window using the distance between a given window and the current window in the DOI function.

Figure 1.1 shows examples of visualization layouts. This figure uses a small slice (5:00AM-6:30AM on Feb. 16, 2009) of a traffic data stream provided by Mn/DOT (Min-

nesota Department of Transportation) [48]. In this slice, each datapoint includes three measured values during a 30 second period from sensor D722. I only choose two dimensions to investigate their correlations here. One dimension is the average vehicle speed (*Speed*), and the other is the percentage of time that the detector sensed a vehicle (*Occupancy*). A traditional time-series data visualization technique, a line chart, is shown in Figure 1.1(a). Figure 1.1(b) shows a naïve solution that treats all datapoints in these 1.5 hours as a static dataset. One can neither identify any strong relationship between *Speed* and *Occupancy*, nor learn how patterns changed over time. Figure 1.1(c) splits the data stream into three time windows and uses colors to denote the age of the windows. One can draw a conclusion that *Occupancy* does not correlate with the change of *Speed* in the early period, but an obvious negative relationship exists between these two dimensions later. In Figure 1.1(d), each time window is visualized by a scatterplot and three of them are juxtaposed in the order of the time attribute. One can easily confirm the pattern changes that were found in Figure 1.1(c), but the last visualization technique relieves the visual clutter found in Figure 1.1(c). In Figure 1.2, One can see how DOI functions work to reduce visual clutter. After the DOI function is adjusted to reduce sampling ratios of the three windows, visual clutter is reduced, and users can more clearly see how a single cluster moves over time.

Although the above approach can be applied to regular static time-series data, the following issues are some considerations to make it efficient for streaming data:

- As mentioned above, the definition of the DOI function is based on the dynamic context of the data stream.
- When designing the visualization techniques to convey the data pattern changes, users' response time is the first priority, since the visual output will get refreshed at regular intervals and users have limited time to observe each figure.

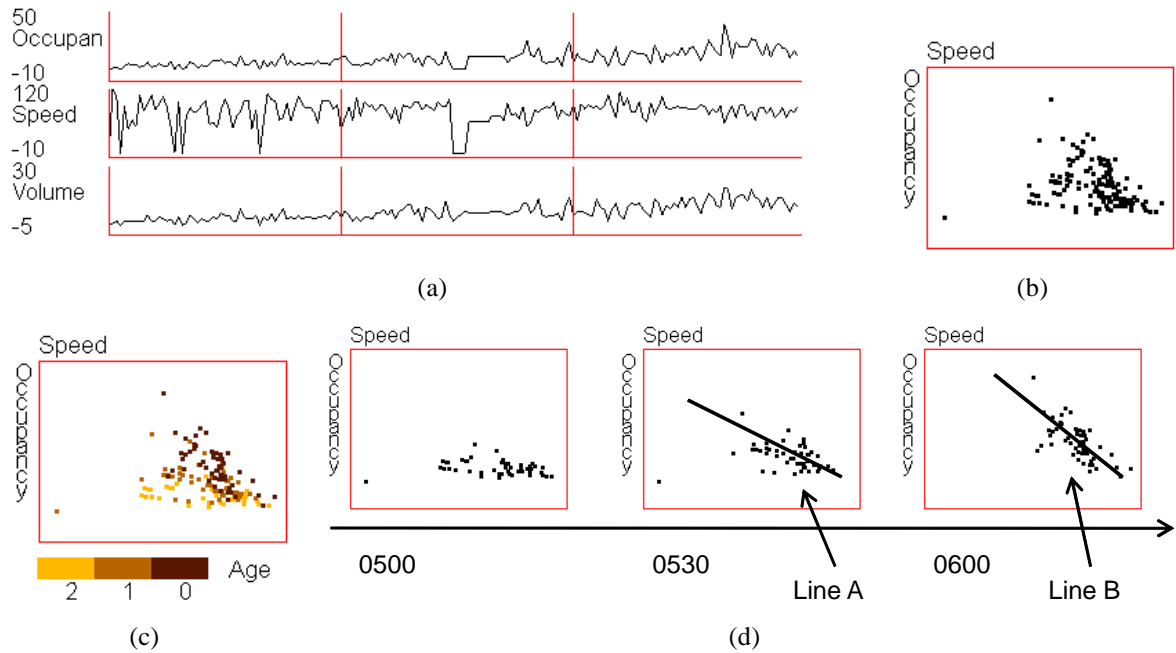


Figure 1.1: This figure shows some of the main ideas of the user-driven approach based on DOI functions using traffic data collected from a highway entrance. (a) A traditional time-series data visualization; (b) All datapoints are shown together in a traditional scatterplot; (c) The ages of data are denoted by colors; (d) Juxtaposition of data in the order of timestamps. Figures (c) and (d) can convey how the fit line slope (I added lines A and B to Figure (d) to make this obvious) changes from one window to the next, but it is difficult to see this change in (a) and (b).

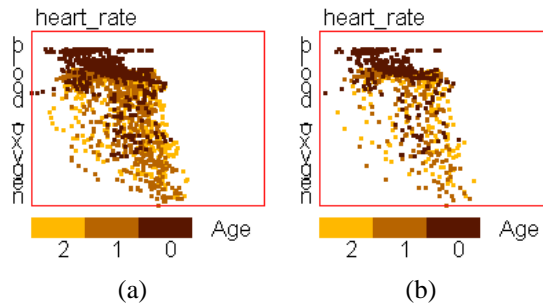


Figure 1.2: Using DOI functions to reduce visual clutter on a sleep data stream. (a) All datapoints are displayed; (b) Sampling is applied to each time window based on the DOI function after user adjustment.

1.2.2 A Data-driven Approach Using Compression

The solution based on DOI functions is effective to visualize the data pattern changes across several time windows. However, if the number of time windows is very big, say 20, this approach does not work well. Figure 1.3 uses step juxtaposition to visualize a slice of traffic data having 13 windows. In this figure, the focus is still the changes of the regression line slope for the linear trend between *Occupancy* and *Speed*. One interesting phenomena is that the regression line in the first subfigure is horizontal, and it becomes almost vertical in the last subfigure. So, one data analysis task is to find where the regression line changed. This might take 20 or 30 seconds to find the change in the subfigures with label “big change”. Under some circumstances, this is not an acceptable response time for urgent tasks. The reason is that the subfigures having “big change” are buried by other ones. In conclusion, when the number of time windows chosen by the DOI function is too big, the disadvantages of the proposed solution include: (1) it might result in a slow response rate, which is not acceptable for some applications, e.g., intensive care units; (2) the display canvas is wasted by a lot of subfigures with small or no changes.

One intuitive solution is using a distorted time axis via assigning more space to subfigures with big pattern changes and merging those with small and no changes. This is a commonly-used technique in many time-series data visualizations [3, 26, 46]. However, all of these techniques are user-driven, which means users decide which slice of the data gets more screen space. For data streams, this is not acceptable, because many applications need a quick response. Thus, the basic idea is to design algorithms to automatically merge windows with small or no changes and assign more screen space to periods having large pattern changes. Figure 1.4 shows a motivating example, where 48 original windows (24 hours) are merged to 3 windows and then visualized by 2 subfigures. Note that each subfigure contains the data in two adjacent time windows, and is linked to the time

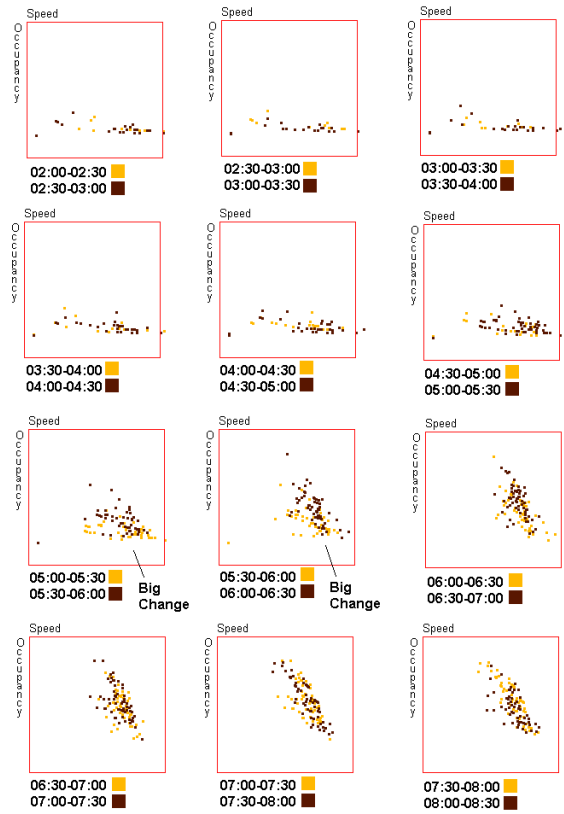


Figure 1.3: A juxtaposed output using the traffic data of 6.5 hours from a specific sensor. 13 windows are shown in this figure. Each subfigure shows two contiguous windows. Data for the current time period is black, and for the previous is yellow. Significant changes are buried in a lot of subfigures with few or no changes.

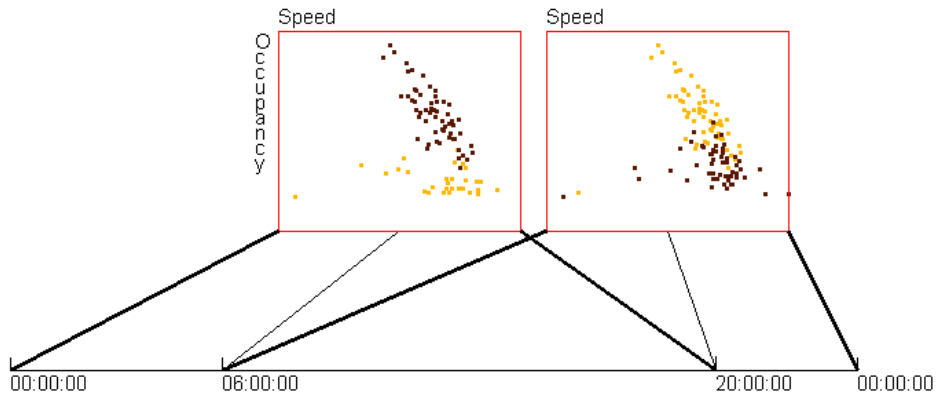


Figure 1.4: 48 windows, containing the data in Figure 1.3, are merged to 3 windows and then shown with 2 scatterplots. Each scatterplot contains two windows, and is linked to the time axis via three lines to delimit the time range for these two windows.

axis via three lines (two thick and one narrow) to delimit the time ranges for these two windows. Obviously, Figure 1.4 reduces users' response time significantly, and merging maintains most of the information about recognizable changes of the fit line slope: the increasing at 6AM, and the decreasing at 8PM.

After merging windows, the next problem is to visualize the data pattern changes. In this dissertation, I describe three types of views to achieve this goal: (1) *juxtaposed views*: laying out all windows using small multiples to enable users to detect the pattern changes; (2) *pattern vector views*: representing the data pattern in each window via a vector, and then visualizing these vectors using traditional time-series data visualizations, e.g., line charts; (3) *pattern change views*: calculating the distance (change) between pattern vectors and visualizing these changes directly.

For some applications, a pattern vector contains only several variables, and users are interested in how each variable changes across time windows. For example, in the prior examples about traffic data, the pattern of interest is the slope change of the regression line. This is equal to tracking a univariate variable. Thus it is reasonable to use traditional time series data visualization techniques, such as line charts or bar charts, to represent the trend. For the proposed three views in the prior paragraph, pattern vector and pattern change views are suitable for this type of case, even across a relatively long time period, e.g, the changes of traffic patterns across one week.

In the algorithm to merge windows, the number of original windows in the current view shown to users is fixed. If one new window arrives, the oldest window in the final view must be removed. This expired window cannot be discard, because users may want to study the change of data patterns over a long time range, e.g., the traffic change within one month. Thus another important issue that needs to be addressed is about storage. Policies have been designed to choose whether to store all tuples or only a vector to describe the data patterns (the pattern vector), for a specific time window. The latter

solution can save memory space at the cost of losing data details and requiring that the pattern vector of a merged window must be computed from the vectors of its associated original windows. This assumption is true for most statistical values, such as average, standard deviation, minimum and maximum. Chen et al. indicate that this is possible for regression analysis on data cubes [15].

1.2.3 History Views with Nested Hierarchical Timelines

In the prior section, I proposed pattern vector and pattern change views with traditional time series data visualization techniques for streaming data across a relatively long time period. This is a possible solution for the visualization of history data. However, if the duration for the whole dataset is very long, e.g., one year, this solution might fail. It is true that the merge algorithm can be applied to the whole dataset, but this cannot satisfy users' requirements. Reasons are as below:

- It is not practical to apply the merge algorithm to the whole dataset because it is very possible to lose some important details. For example, if the traffic data of one year is compressed to less than 365 time windows, nobody can see the traffic pattern changes in some days.
- Only showing line or bar charts cannot solve some more practical data analysis tasks, such as discovering the similarity among different time periods.
- Many temporal datasets have hierarchical time structure and their patterns change in a cyclic way. A traditional line chart or bar chart, as mentioned in Sections 1.2.1 and 1.2.2, cannot effectively convey this phenomena.

In the recent literature, distorted timelines are commonly used in the visualizations for large scale time series data [3, 26, 46, 62]. For example, Bade et al. [3] used three

timelines at different resolutions to represent a time series dataset across 9 years. Users can easily navigate to any time period. Borrowing from this idea, I have developed a hierarchical structure to represent timelines. This structure is defined by users. It can be based on natural time units, such as years, quarters, weeks, and days, or from an arbitrary definition. Figure 1.5 shows an example of hierarchical timelines having five levels. Users have specified a perspective level on quarters and a pattern level on days. It means that users wanted to investigate how the traffic patterns change across days within a selected quarter. Figure 1.5 also shows a history view composed of glyphs in a grid. Each glyph corresponds to one day in the selected quarter and conveys the slope change of regression line (*Occupancy* against *Speed*) via a curve.

Although this is a natural extension of *pattern vector views* described in Section 1.2.3, this solution has some obvious advantages:

- This solution makes it easy for users to discover cyclic pattern change phenomena on the streaming datasets having hierarchical time structure.
- The goal to investigate the similarity among time windows cannot be achieved by applying more visualization and interaction techniques to the above proposed solution. For example, distance measures can be used to represent the difference among glyphs and map them to colors.

When users explore the traffic data, they might want to observe the detailed information within one day instead of pattern abstractions. That is to say, they want to observe the datapoints themselves instead of pattern vectors. This will bring the same problem as that in Section 1.2.2, long response time caused by too many time windows (Figure 1.3). Therefore, I use the merge algorithm mentioned in Figure 1.2.2 to reduce the number of visualized windows in each day. Different from the solution described in Section 1.2.2, this generates multiple merge results, each of which has a different number of time

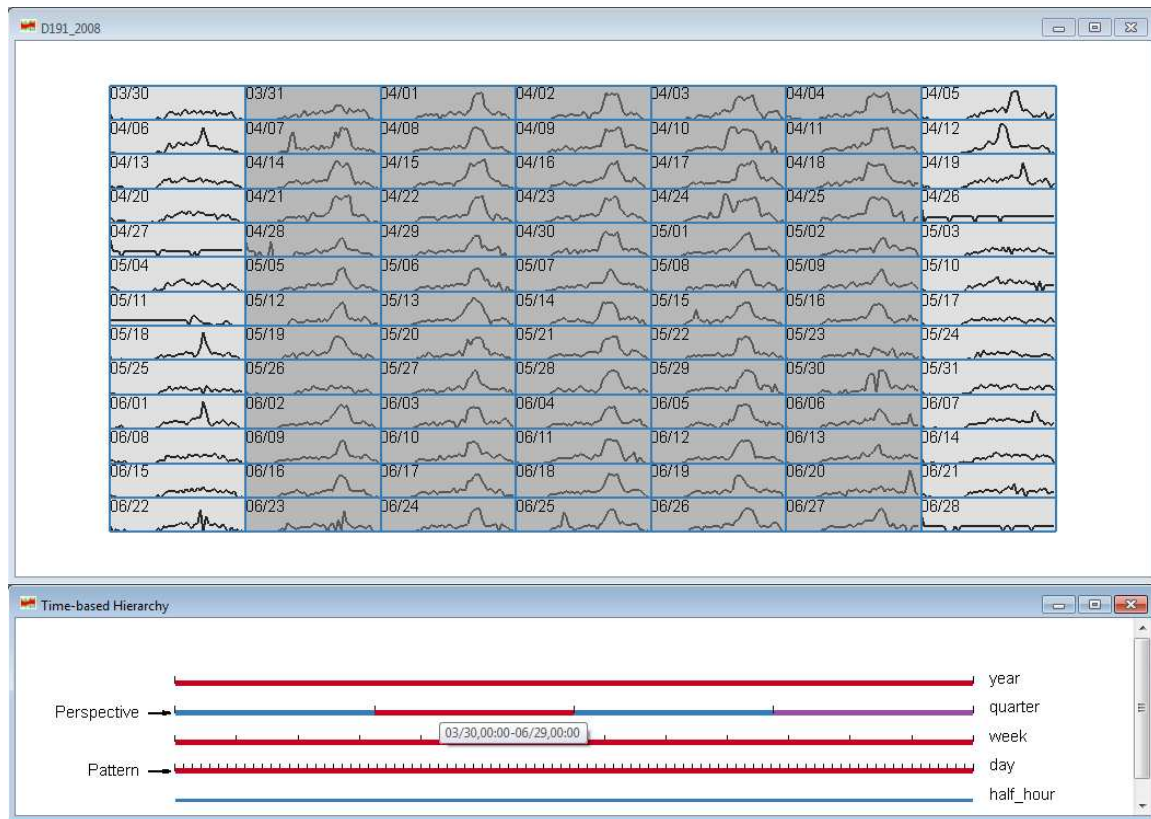


Figure 1.5: This figure shows a history view (top) with hierarchical time structure (bottom) defined by users. I focus on the changes across contiguous windows on the pattern level (days) in this figure. The selected quarter (March 30 - June 29) on the perspective level is highlighted in red color and indicates the time periods of interest. The red color on the week and day level means all segments in the selected quarter are selected. In the history view, each glyph corresponds to one day (pattern level) and contains a curve to represent the slope change of regression lines within 48 time windows for each day. Grey background is applied to all weekdays to help readers observe data patterns.

windows. One example is first merging 48 original windows to 22 windows, and then changing the threshold for the pattern change measures for merging 22 windows to 9 windows. This merging process can be repeated until until only one window contains all datapoints in this day. Thus a hierarchical structure is generated to allow users to select an appropriate level to observe data details.

1.3 Major Contributions of the Dissertation

The main contributions of this dissertation are as follows:

A user-driven approach based on a DOI function

- A framework for the visual exploration of streaming data is presented. This framework applies the strategies of windowing and sampling to multivariate and time-series data visualization techniques. Its aim is to handle unbounded input as well as to convey trends, multivariate correlations, and the changes of data patterns over time.
- This framework allows users to define DOI functions to describe the degree of users' interest [21] for different portions of the data. This function enables users to choose which windows to display, and to adjust sampling ratios to reduce possible visual clutter.
- Four layout strategies are designed to organize multivariate data visualizations and convey the change of multi-dimensional correlations.
- User studies showed that three of four proposed strategies can effectively convey multivariate pattern changes compared to traditional time-series data visualization techniques. Using the experimental results, a guide was derived to advise data analysts and visualization system developers to choose appropriate layout strategies in terms of the characteristics of datasets and data analysis tasks.
- Interaction techniques were designed and integrated to help analysts explore data streams, including a DOI function interaction tool that helps users analyze data via a trial-and-error process, and linked brushing across multiple views. Several cases studies are discussed to show the effectiveness of these interaction tools.

A Data-driven Approach Using Compression

- A framework was developed to visualize data streams with the goal of showing significant pattern changes to users. The main approach is to merge those windows with few or no changes when visualizing and storing recent as well as old data.
- The above framework is materialized using two frequently used data patterns: linear trends and data range.
- Experiments were performed to show that the merge algorithm preserves more change information than an intuitive pattern-blind averaging. User studies were conducted to demonstrate that the techniques can significantly reduce users' response time when looking for significant pattern changes in a data stream.

History Views with Nested Hierarchical Timelines

- A framework was designed to convey the pattern changes within a relatively long time period. The main idea is to generate a hierarchical structure for timelines, with which users can easily navigate within the history data.
- An MDS algorithm and brushing technologies were applied to the history view in order to ease the exploration on the similarity among time windows.
- A usability evaluation was performed to confirm that most users can correctly understand the concepts involved in these new techniques and can learn to use the implemented system without too much difficulty.

Chapter 2

Related Work

Based on the analysis in Chapter 1, one can see that the techniques of streaming data visualization can be regarded as real-time time-series visualization with unbounded and large-scale input. Therefore, it is necessary to go back to existing visualization techniques for time-series data in recent literature and seek inspirations.

Time-series data visualization is a very popular topic in information visualization. It aims to help analysts perceive data patterns in datasets, where each data item has a timestamp or a time dimension. In the recent literature, one can find many techniques related to this topic [2, 3, 6, 12, 24, 25, 26, 27, 28, 29, 45, 46, 49, 58, 62, 63, 64].

Most of these techniques for time-series data visualization focus on one or more of the following four sub-problems: (1) representation of the change of univariate data over time; (2) layout of data items in terms of the time dimension; (3) how to convey relationships and data patterns among multiple data dimensions; and (4) how to handle very large and real-time data.

These four sub-problems are not independent. For example, a good layout can help mining multi-dimensional patterns. In the following sections, I will discuss existing techniques in terms of the above four aspects.

2.1 Representation of Univariate Data

In order to represent how univariate data change over time, line charts are frequently used. One can find many applications in routine life, such as financial areas, meteorology, and so on, where line charts can represent the change of stock prices, currency ratios, temperatures, or some other numerical values. Since simple line charts cannot convey too much information, people normally add some variations to represent more patterns of data. The candlestick chart [10] is such a variation widely used in representing the change of stock prices, currency ratios or other equities (see Figure 2.1). This chart combines line charts and bar charts, which are used to reflect the change of univariate data in the specified period (e.g., a day). Figure 2.2 shows two types of candlesticks corresponding to the increasing and decreasing price (ratio) in a time period.



Figure 2.1: A candlestick chart which denote change of currency ratio between USD and JPY from Nov. 1, 2005 to Jan. 7, 2006 [10].

Miksch et al. put a vertical colored line in each time point to represent the data uncertainty for a univariate temporal dataset [46]. They first applied a linear regression model to a time window of fixed size sliding over the entire curve in small steps. Then they plotted a vertical red line, termed the *spread*, around the center of each regression line to represent the distribution of datapoints in the corresponding sliding window. The height



Figure 2.2: Two kinds of candlesticks (Low: the lowest price, high: the highest price, open: the price in the beginning of the period, close: the price in the end of the period).

of each red line is $2D$, where D is the standard deviation of the datapoints in that sliding window. By connecting all upper and lower ends of the *spread*, this approach can describe a region containing most of the datapoints. Figure 2.3 shows such a region with three *spreads*.

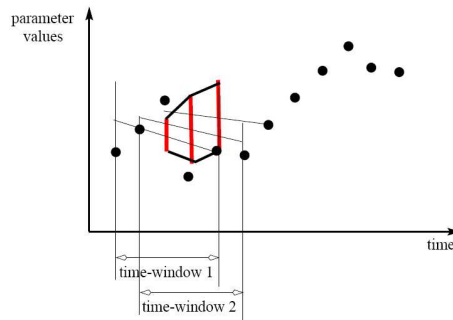


Figure 2.3: The red stripes around the regression line denote the standard deviation, thus the distributions of the datapoints in the corresponding sliding window. The polygon generated by connecting the ends of these stripes contains most of the datapoints [46].

Bade et al. [3] created an ICU monitoring system, named *MIDGAARD*, which utilizes line charts to display changing pathological measurements of patients, such as body temperatures and blood pressure. They applied various visual encoding styles to the basic line charts to better convey the data trends. Figure 2.4 shows two kinds of encoding for a fever curve, which can display clearly the change of body temperature from one threshold value to the other.

Boukhelifa et al. designed an open architecture, named *WikiReactive*, to collect sev-



(a) Color-coded



(b) Height-coded

Figure 2.4: Two types of coded timeline representation of a fever curve [3].

eral aggregated measures on the French Wikipedia. This architecture can provide the data via Web Service or visualization [9]. For example, in Figure 2.5, line charts are used to show the increased activities on Wikipedia Fr.

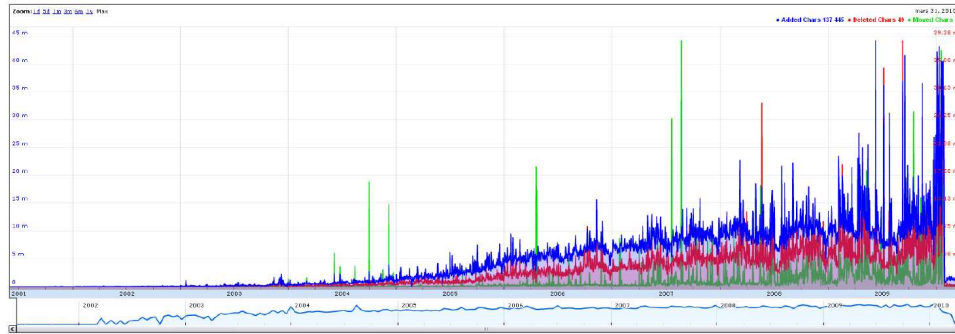
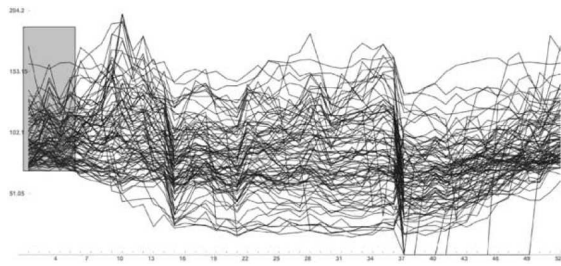


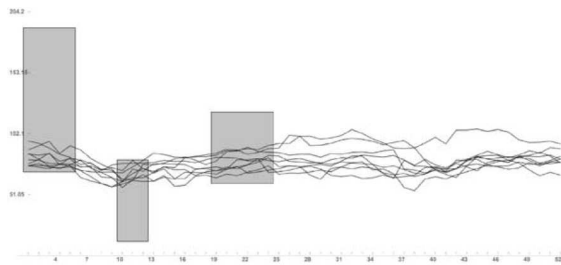
Figure 2.5: This figure shows the number of added (in blue), deleted (in red), and moved (in green) characters for all articles on Wikipedia.fr. [9].

Hochheiser and Shneiderman developed *TimeSearcher*, an information visualization tool to help analysts explore line charts composed of multiple univariate time-series profiles [29]. They used *timebox* and *angle query* widgets in this tool to represent queries on a univariate profile. If $n_i \in N$ is an item in a time-series dataset, and $n_i(j)$ is the value of n_i at time j , then a *timebox* is a 4-tuple: $b = (t_{min}, t_{max}, v_{min}, v_{max})$. Note that $t_{min} \leq t_{max}$ and $v_{min} \leq v_{max}$. If one item n_i satisfies $\forall t(t_{min} \leq t_{max} \rightarrow v_{min} \leq n_i(t) \leq v_{max})$, one can say that this item satisfies this timebox. An *angle query* widget is defined as a 4-tuple: $b = (t_{min}, t_{max}, \theta_{min}, \theta_{max})$. It can help users find items with similar slopes over several specified time periods. Figure 2.6 shows the results of applying timeboxes to a dataset of stock prices that changed over time.

The *river metaphor* is frequently used to visualize a special kind of univariate time-



(a)



(b)

Figure 2.6: The original dataset contains prices of 1430 stocks. In order to find stocks satisfying some specified constraints, a single timebox is used in (a), thus filtering a subset of items. Three timeboxes in (b) result in a further refinement of the query in (a) [29].

series data, which represents the number of occurrences over time for repeated items, e.g., baby names [62], DNS traffic [52], and keywords of patents [27]. In this technique, The river’s changing width corresponds to the magnitude of occurrences. Figure 2.7 shows a screenshot from *NameVoyager*, which shows the frequency of baby names. In Figure 2.8, the band width represents the request frequency from different source IP addresses.

Although my dissertation will focus on multivariate data, some measures in the target patterns, such as slope of the regression lines, are still univariate. For example, in history view, line charts will be used to represent the regression lines across a long time period.

2.2 Layout Strategies for Time-series Data

An intuitive approach to time-series data layout is timelines[59], which use the horizontal axis to represent time. The horizontal position of a data item conveys its time attributes.

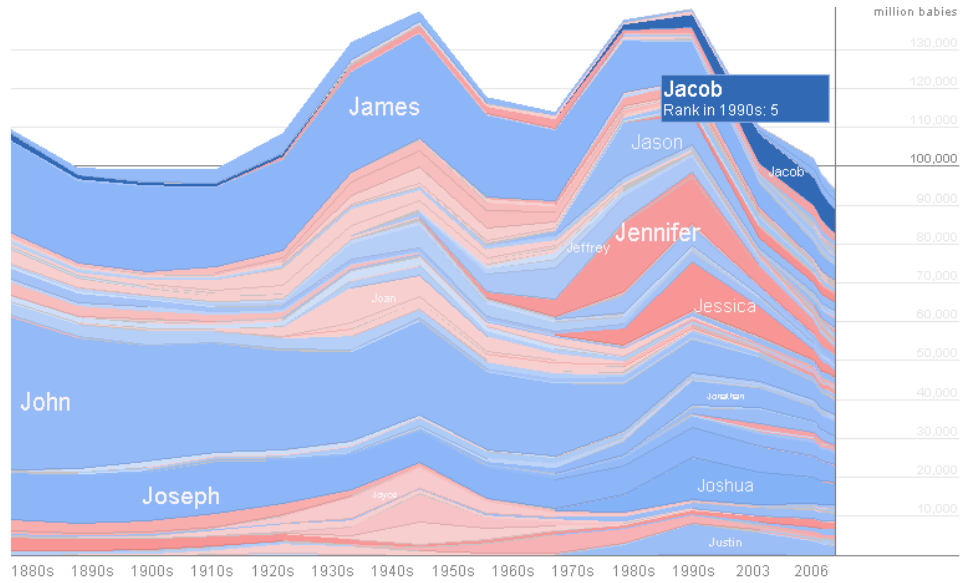


Figure 2.7: This figure shows the change of frequency for baby names starting with 'J' from 1880s to 2006. Blue and red rivers correspond to boy and girls respectively. The river width is proportional to the number of occurrences [62].

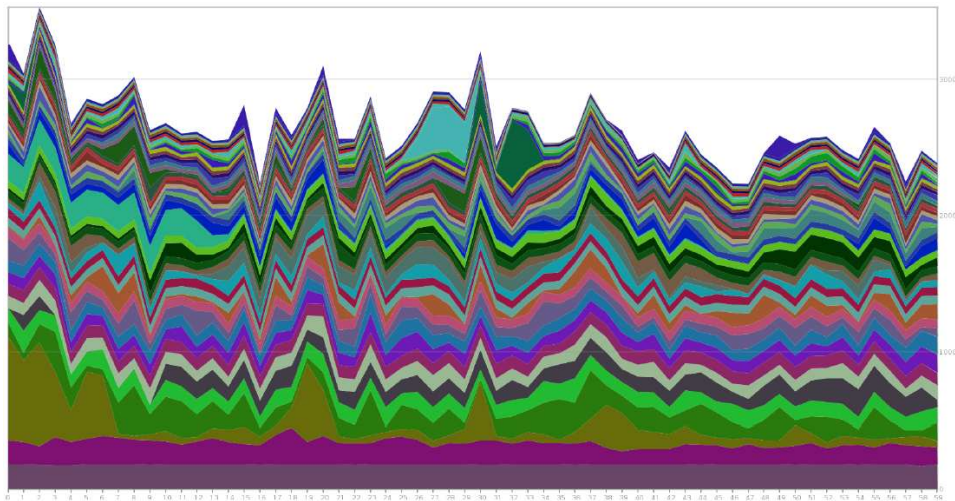


Figure 2.8: In this figure, each stream band corresponds to one source IP address. Its width is proportional to the number of request sent from this IP [52].

Timelines are effective in showing how one variable changes over time. We can see that all of the discussed techniques in Section 2.1 use a timeline as the layout strategy.

Other than timelines, researchers have developed other layout methods to facilitate specific analytic tasks. *Spiral Graphs* [12, 28, 60, 63] use a spirally shaped time axis to visualize temporal data having seasonal cyclic characteristics. If an appropriate visual encoding is utilized, repeated patterns can be easily identified. Figure 2.9 shows a spiral layout of star glyphs [60]. Each glyph contains the average values of four statistical variables in one month: Dow Jones Industrial Average (top), Standard and Poor's 500 Index (bottom), retail sales (right), and unemployment (left). One circle is one year. 12:00 corresponds to January. We can easily identify some data patterns which are difficult to obtain with regular timelines. For example, normally, sales peak happens in December, and unemployment reaches the maximum in June.

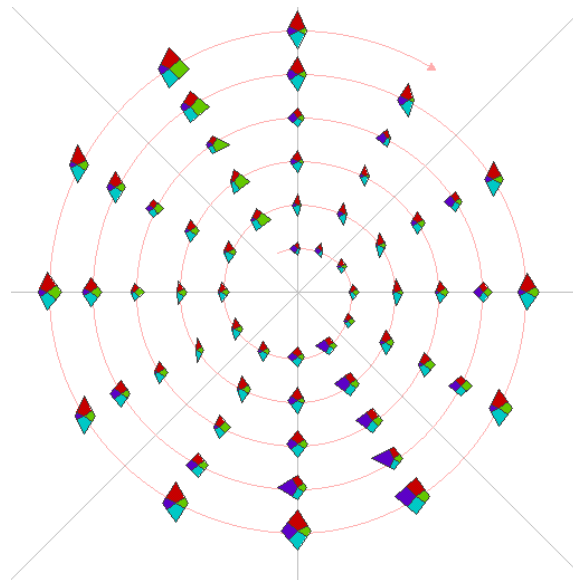


Figure 2.9: The star glyphs with spiral layout for a time-series dataset about business, sales, and unemployment data [60].

Van Wijk and Selow introduced a *Calendar View* combined with clustering to visualize natural phenomena that change weekly, monthly, or yearly [64]. Figure 2.10 shows

such a view to show the number of employees present at a research center. Days having similar behavior of employee presence are organized into one cluster tagged with different colors. For instance, cluster 718 (green color) denotes the Fridays during the summer. In this calendar view, the changing data patterns over time are showed clearly with the help of the calendar and clustering.

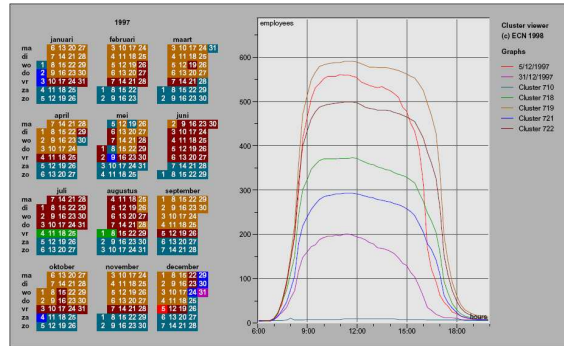


Figure 2.10: The calendar view of the number of employees present at a research center [64].

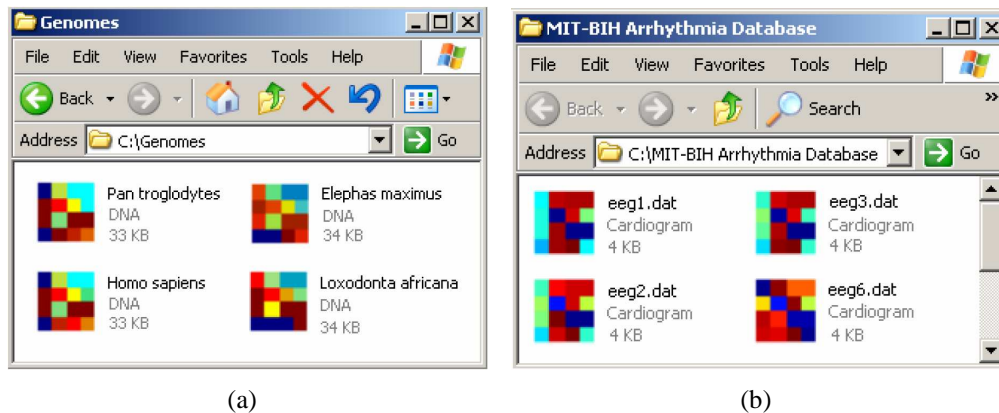


Figure 2.11: This figure shows time-series bitmaps to color encode time series and represent them via images. (a) The DNA of four animals are represented by images using the *Chaos game* algorithm, which is the base of Time-series bitmaps. It is obvious that the left two animals are similar to each other, and a high degree of similarity exists between the other two. (b) Four files correspond to different patients with congestive heart failures. One can find that *eeg6* is different from other three people [36].

Kumar et al. developed *Time-series bitmaps* to use a compact way to represent a time

series dataset [36]. This technique used an algorithm adapted from the *Chaos game* [4] that visualizes DNA sequences by color encoding the genome patterns. It first uses the Symbolic Aggregate approXimation(SAX) [42] to convert real valued series into discrete symbols, and then applies the *Chaos game* to them to produce bitmaps. Figure 2.11 shows DNA visualizations by the *Chaos game* (Figure 2.11(a)) and *Time-series bitmaps* (Figure 2.11(b)).

2.3 Conveying Multi-dimensional Patterns

For the third sub-problem, most techniques discussed above can represent some relationships or data patterns implicitly. For example, integrating multiple lines in one graph can help identify how the change of one dimension relates to the others. However, direct visualization of the differences might be more effective.

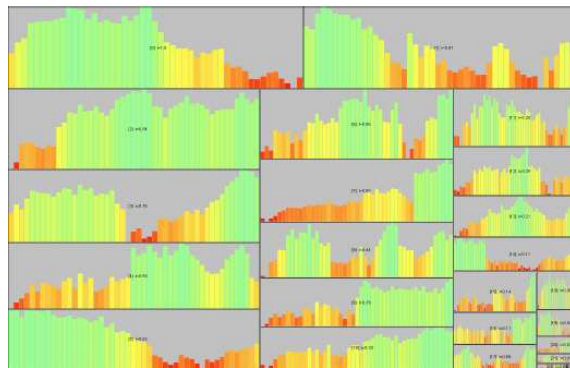


Figure 2.12: The importance-driven layout of a time-series dataset of 24 stock prices [25]. The stock price is normalized and redundantly mapped to the bar colors and height.

Hao et al. used an importance-driven layout to represent the degree of importance for different dimensions via assigning the important dimensions more space [25]. Figure 2.12 shows a time-series dataset of 24 stock prices. Note that each stock is represented by a bar chart. The size of each bar chart is proportional to the degree of importance for the corresponding stock. From this figure, one can easily learn which stocks are more

important than others. Obviously, this approach can be easily adapted to other measures for multiple data dimensions in multivariate time-series datasets or data streams.

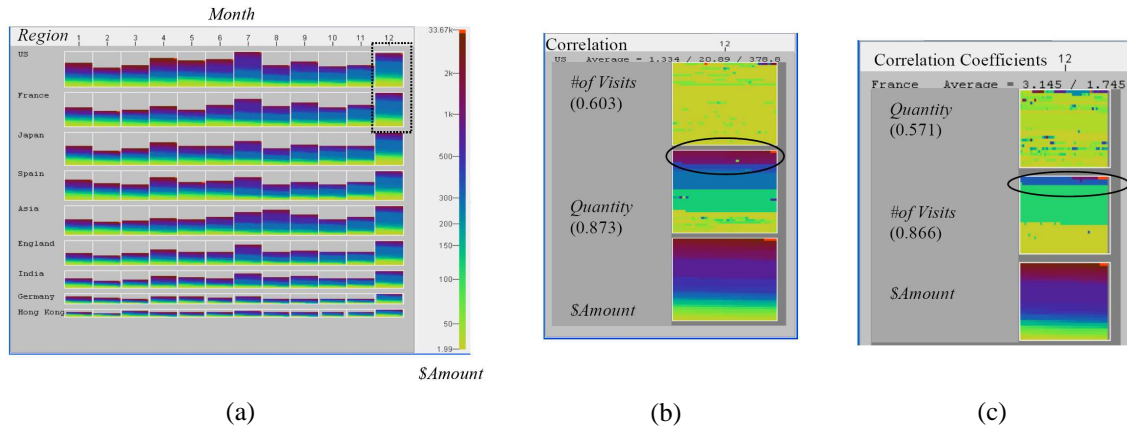


Figure 2.13: An example of applying IVQuery to a time-series dataset about sales data for a company in different countries [24].

The Intelligent Visual Analytics Query (IVQuery) is a visual analysis tool to help users perceive relationships among multiple time-series data dimensions [24]. The IV-Query can report some analysis results, such as relevance measures, classification, and clustering, after users select the region of interest. An example is shown in Figure 2.13. Figure 2.13(a) is a monthly regional sales map. Each line is a bar chart representing the sales amount in a country, with one bar denoting one month. One pixel in the bar corresponds to one invoice and its color is determined by this transaction $\$Amount$ in terms of the legend on the right side. We can find that US and France have the most invoices in December. If analysts want to mine the sale behavior of this month in these two countries, they can select the corresponding two bars and perform an IVQuery. The analysis results are shown in Figures 2.13(b) and 2.13(c), corresponding to US and France respectively. The numbers below the *# of Visits* and *Quantity* are the Pearson correlation coefficients used to measure the relevance between these two dimensions and the invoice $\$Amount$. These two dimensions and $\$Amount$ are also visualized via pixel maps. Figure 2.13(b)

indicates that sales $\$Amount$ in US correlates more to the quantities in each transaction ($Quantity$, 0.873) than to the number of visits of the customers ($\# of Visits$, 0.603). However, the reverse situation happened in France. Based on the IVQuery results, analysts might design better sale strategies for different countries in the future.

The above two techniques both have a common goal to convey multi-dimensional relationships in time-series data. However, their limitations are obvious. Importance-driven layout is limited to representing the ordering of multiple dimensions. The IV-Query currently can only measure relevance, conduct classification and clustering. More non-trivial design and implementation are needed if users want other types of statistical or data mining analysis. Compared to these techniques, the goal of my dissertation is to enable visualizations to convey general multi-dimensional relationships, such as with traditional multivariate visualizations: parallel coordinate [32], scatterplot matrices [1], star glyphs [56], dimension stacking [38], and pixel-oriented displays [33]. My basic idea is to adapt these techniques to data streams.

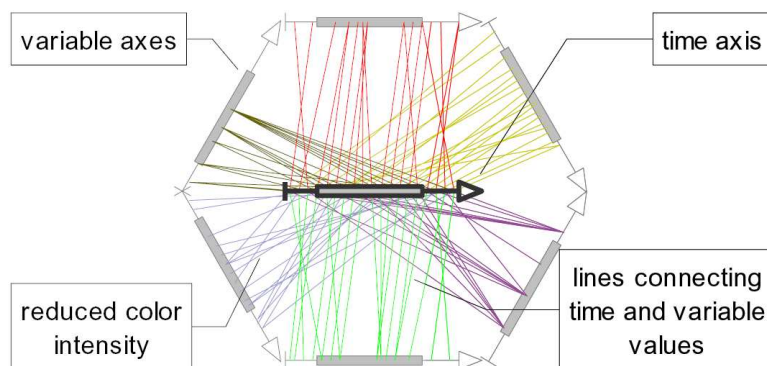


Figure 2.14: A TimeWheel with six variables changing over time [58].

TimeWheel puts the time axis in the center and other data dimensions arranged circularly. Each data item corresponds to a group of lines from the time axis to the axes for data dimensions. Figure 2.14 shows a TimeWheel to visualize a time-series dataset having six dimensions, not including the time dimension. However, one limitation of TimeWheel

is that it is difficult for users to identify data patterns among different data dimensions because it does not explicitly convey the relationship among dimensions. I believe that users will be able to identify and classify complex temporal patterns and relations via the research efforts that will be presented in my dissertation.

Yu et al. developed a tool for the visual analysis of multi-stream multimedia data [69]. Continuous time-series data and event data are first extracted from the multimedia stream, and then are visualized via line charts and heatmaps. Users can highlight selected data portions, or zoom in on the region of interest to study the data trends and the multivariate data patterns. Figure 2.15 shows a screenshot from the system developed by Yu et al. Compared to the techniques that are presented in this dissertation, this tool utilizes line charts and heatmaps to convey multivariate relationships. This is not effective since these visualization techniques only excel in conveying univariate trends.

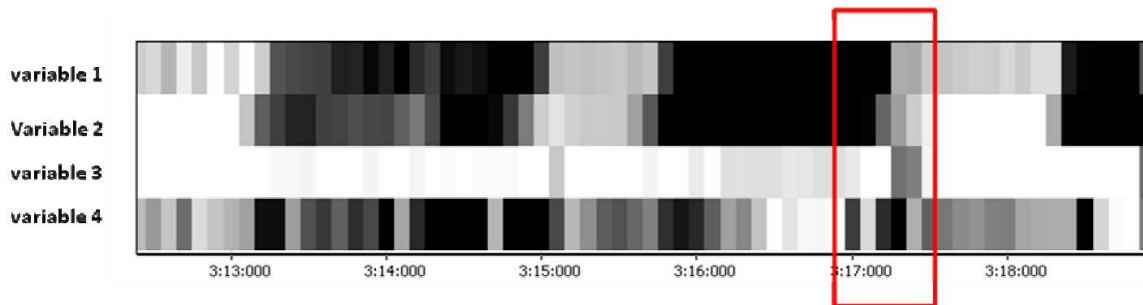


Figure 2.15: This figure shows an example of using heatmap to investigate the multidimensional correlations among four variables in a multimedia data stream [69]. Although one can see that variable 1 and variable 2 are correlated and variable 3 and variable 4 are not, this technique only excels in conveying univariate trends and might cause a long response time.

2.4 Handling Large Scale and Real Time Data in Time-Series Data Visualization

In order to deal with large time-series datasets, some abstraction algorithms have been introduced into time-series visualization for adapting large temporal datasets to limited display space. The approaches can be categorized into two ways: data-driven [45] and user-driven [26, 31] means.

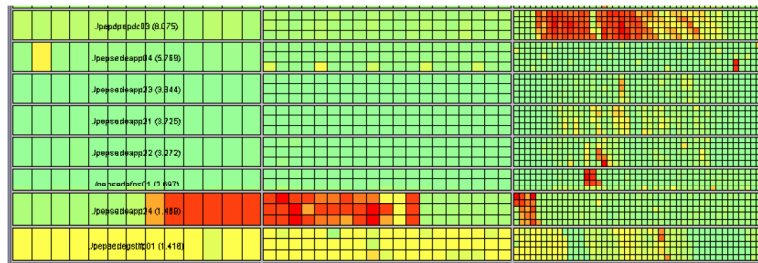


Figure 2.16: A multi-resolution display for a time-series dataset containing the CPU utilization history of 8 hosts. One host corresponds to one row in the figure. Three parts of each row have different DOI function values. The most recent data is at the right part, and has the largest DOI values, so it has a higher sampling rate and a smaller grid size. The older data is in the middle and the oldest is at the left. They have lower sampling rates, thus yielding a bigger grid size [26].

Miksch et al. developed an abstraction algorithm for temporal univariate data that aims to transform numerical values to qualitative descriptions [45]. It can smooth data oscillation near thresholds. This algorithm was implemented in VIE-VENT, an open-loop knowledge-based monitoring and therapy planning system for artificially ventilated newborn infants. Hao et al. used the sampling technique to abstract time-series data and introduced DOI (degree of interest) functions to determine the sampling rate. The DOI function is used to represent how users are interested in different portions of a time-series dataset [26]. The subset of the original dataset with a high DOI value will be abstracted using a high sampling rate and displayed in high resolution. Otherwise, the overview with lower resolution corresponding to the DOI value will be displayed. An example is shown

in Figure 2.16. In this dissertation, I will extend the DOI function to dynamic context for streaming data, as well as the use case that needs to convey repeated occurring data patterns.

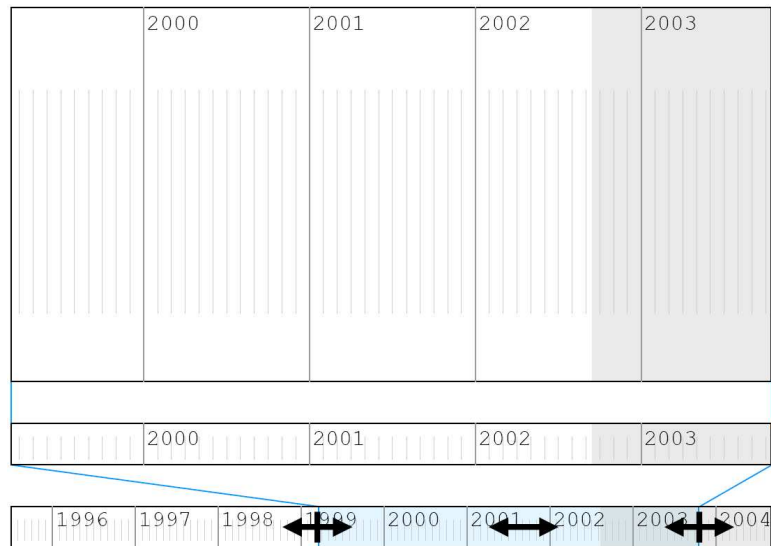


Figure 2.17: The timeline interaction: the bottom timeline refers to the whole time period. Users can select a subrange, thus getting a rescaled view in the middle and top timelines [3].

If using an abstraction algorithm, a distorted timeline might be necessary to give important data more space. This technique is used in many research efforts [3, 26, 31, 46, 62]. Figure 2.17 shows an interaction interface to distort a timeline driven by users [3].

Huynh’s *timeline* in SIMILE project [31] uses the same way to distort a timeline as Figure 2.17, but focused on representing instant (focusing on a specific time) and duration(occurring over a period of time) events. Figure 2.18 shows a screenshot of *timeline* that visualizes the instant events before and after the President Kennedy was assassinated. It represents the time axis in hierarchical ways. The bottom part has a bigger interval (15 minutes per cell), while the top uses one grid to denote 5 minutes. Dragging the bottom part horizontally can make the visualizations shift more quickly than doing it on the top.

Actually, I followed two basic approaches, namely the data-driven and user-driven

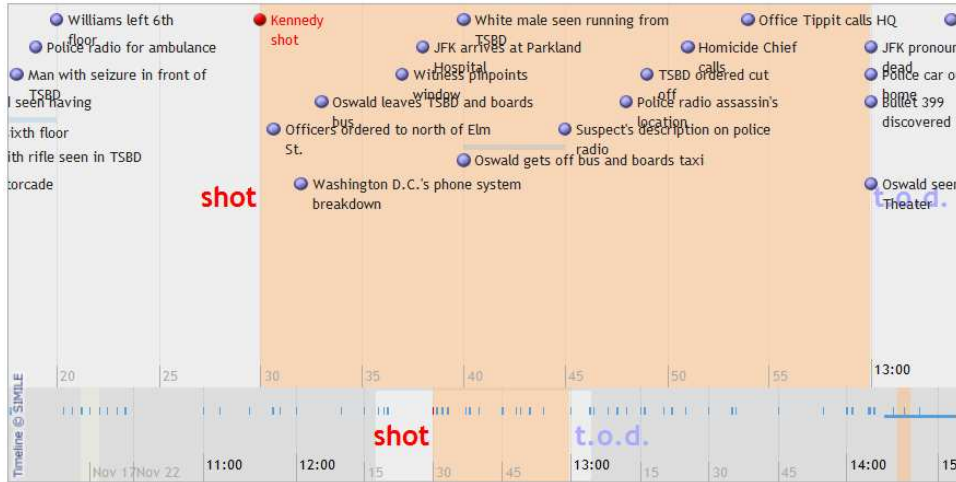


Figure 2.18: A screenshot of the *timeline* in SIMILE project developed by Huynh [31]. There are two time axes in this figure. The top one has a finer granularity.

means discussed in previous text, in my dissertation. The difference between the above research efforts and mine is that they are only applied to univariate data or events and did not consider real-time requirements. I focused on unbounded multivariate data generated in real-time to overcome this limitation.

Visualization experts have developed some systems used in real applications to monitor large-scale time series data. *WireVis* is a system that can help banks detect fraudulent activities using a set of coordinated views based on identifying specific keywords within the wire transactions [13]. Figure 2.19 shows a screen snapshot of this system. It is composed of four views: (1) *the heatmap* (top left), which uses colors to represent the frequency of keywords (column) in account clusters (row); (2) *search by example* (top right), that can search accounts having activities similar to a specific wire transfer; (3) *keyword graph* (lower right) to show the relationship between keywords, with the most frequent keywords in the middle of the view; (4) *strings and beads view* (lower left), in which the strings (the curve in the view) represents the accounts or clusters of accounts over time, and the beads (the dots on the strings) refer to specific transactions on a given

day. Note that, in the fourth view, the x-axis shows the progression of time, and the y-axis denotes the “value” of the transaction, such as the amount of transactions, and the frequency of activities. Four views are coordinated. For example, if users selected a keyword in the heatmap, then all activities (beads) in the *strings and beads* view having this keyword will be highlighted. If the fraud analysts are interested in one activity and select it, accounts having similar behavior will be shown in the *search by example* view. Users further can select one search result and look at it in other views. Because illegitimate wire transfers normally have similar pattern on keywords, amount and frequency, this system can effectively help risk managers in banks find and report those transfers related to criminal endeavors.

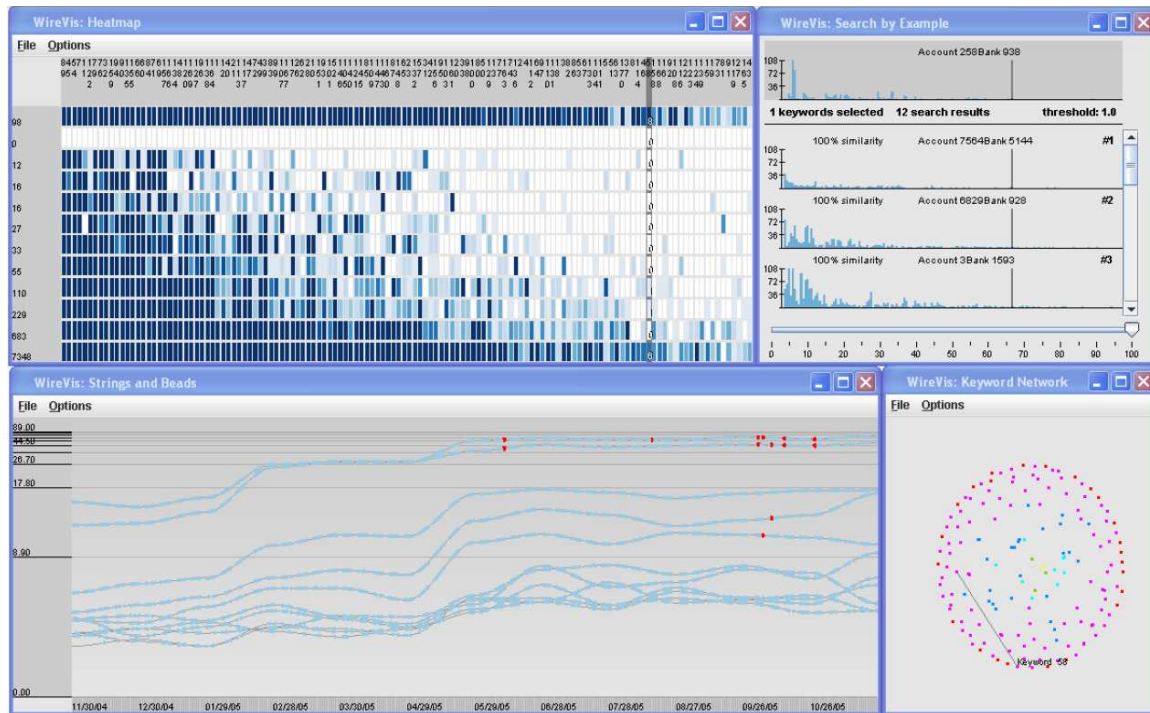


Figure 2.19: This figure shows the *WireVis* developed by Chang et al. [13]. It visualizes the information within wire transfers and help fraud analysts in banks to identify those related to criminal endeavors such as money laundering. This system shows four views:the heatmap (top left), search by example (top right), keyword graph (lower right), and strings and beads (lower left).

Chapter 3

Data Model and Example Datasets

3.1 Data Model

In the context of this dissertation, a data point is defined as:

$$(V, ts) = (v_1, v_2, \dots, v_n, ts) \quad (3.1)$$

where n is the number of dimensions, $v_i (1 \leq i \leq n)$ are real numbers denoting attribute values, and ts is the timestamp when the datapoint originated. Nominal values or other types of data, such as documents, images and video, are left for future work.

When investigating different streaming datasets, I found three main types of multivariate data streams, which is important for the selection of visualization techniques:

- **Univariate-Aggregation:** In this type of data stream, each dimension can be regarded as a univariate data stream. For example, in Equation 3.1, if v_i represents the price of one company's stock, this dataset belongs to a univariate-aggregation type.
- **Object-Aggregation:** For some data streams, arriving datapoints belong to differ-

ent objects, so trends on each dimension do not always make sense. One example is the KDD CUP'99 dataset [50], which is a network intrusion detection stream dataset obtained from an experimental network at MIT Lincoln labs. It recorded some attributes, such as duration and the number of bytes, for all TCP and UDP connections within nine weeks when some network attacks were simulated.

- **The Combination of above Two Types:** The KDD CUP'00 dataset [34] is a combination of the above two types. It contains clickstream and order data from Gazelle.com. Obviously, the dimensions in this data stream, such as quantity, and tax amount, cannot form univariate data streams because they belong to different purchasers. However, if a subset corresponding to a specific buyer or product is picked, it can be regarded as univariate aggregation.

In this dissertation, I will focus on univariate-aggregation and try to visually convey trends for each dimension as well as multivariate data patterns.

3.2 Example Streaming Datasets

The two streaming datasets used in this dissertation are the following.

Traffic Data Stream: I got this dataset from Mn/DOT [48] (Minnesota Department of Transportation). In Section 1.2.1, I have shown a slice of this data stream. Mn/DOT installed more than one thousand sensors on highway entrance/exit ramps and main lanes throughout the Twin Cities Metro area. Each detector can collect a value for each of the following measures with an interval of 30 seconds: (1) Volume: the number of vehicles passing the detector; (2) Occupancy: the percentage of time that the detector sensed a vehicle; and (3) Speed: the average speed of all vehicles passing the detector. The website of Mn/DOT provides a Java-based tool, *DataExtract*, to allow users to extract detector data to csv files. Thus several thousand values can be obtained every 30 seconds. Instead

of using all of these values at the same time, I select one detector and retrieve its three measures during a specific time period, e.g., one day or one week.

Sleep Data Stream: This data stream is a physiological dataset (Santa Fe time series competition data set B) selected from the PhysioBank archive [23]. It is recorded from a patient suffering from sleep apnea (periods during which he takes a few quick breaths and then stops breathing for up to 45 seconds) in a sleep laboratory. Since it is relatively long (about 4 hours at a frequency of 2Hz), it can be used to simulate a data stream. This dataset has three measures: heart rate, chest volume (respiration force), and blood oxygen concentration.

Although the above datasets both have only three dimensions, the proposed techniques in this dissertation can support more dimensions because they are based on some traditional multivariate visualization technologies, such as scatterplot matrices and parallel coordinates.

Chapter 4

A User-driven Approach Based on DOI Functions

4.1 The User-driven Framework Based on Windowing, Sampling and DOI Functions

In this section, I will discuss my proposed user-driven framework. The basic goal of this framework is to provide a mechanism that allows users to select multiple time windows of interest, from which they can observe how data patterns change across windows or across cycles. The design of this framework also can claim its validity within the dynamic context of the data stream.

4.1.1 Basic Concepts

Sampling Ratio: The proposed method is based on sampling. The term *sampling ratio* is used as the percentage of datapoints to be selected to display. Sampling ratio is defined

as follows:

$$r(\text{sampling ratio}) = \frac{\text{the number of selected datapoints}}{\text{the number of all datapoints}} \quad (4.1)$$

Note that sampling ratio r must satisfy $0 \leq r \leq 1$.

DOI Functions: A DOI function represents how interested the user is in seeing a particular time window. In a regular static dataset, a DOI function calculates a value to represent the degree of interest for a portion of the dataset based on, for example, the age of the data or the presence or absence of a data feature. Then the portion of data with high DOI values will be displayed with more details [21]. For streaming data, when the stream system gets a new time window (Window i), a specific DOI level should be applied to this new portion of data. However, when Window i expires and a new window (Window $i + 1$) becomes the current one, users might want to focus on Window $i + 1$ and show less interest in Window i . In this situation, the sampling ratio for Window i must change. Hence, an age-based DOI function should have two parameters: a timestamp representing the specific time window and the current time point. Formally, the DOI function is given as $\text{DOI} = f_{doi}(t_d, t_c)$, where t_d and t_c are the timestamps corresponding to the target time window and the current one.

4.1.2 User-driven framework

In order to describe this framework clearly, I will first briefly describe the information visualization reference model, or namely the visualization pipeline. In this model developed by Chi [16] and re-interpreted by Card, Mackinlay, and Shneiderman [11], the visualization process is divided into three steps: (1) data transformation: pre-process the raw data and generate data tables; (2) visual mapping: construct a visual abstraction of the data represented by visual properties, such as position, color, and geometry; (3) ren-

dering: create interactive views of the data. This section will mainly describe how to do data transformation, i.e., generating data tables from the data streams, and then feed the visual mapping stage of traditional multivariate data visualization techniques.

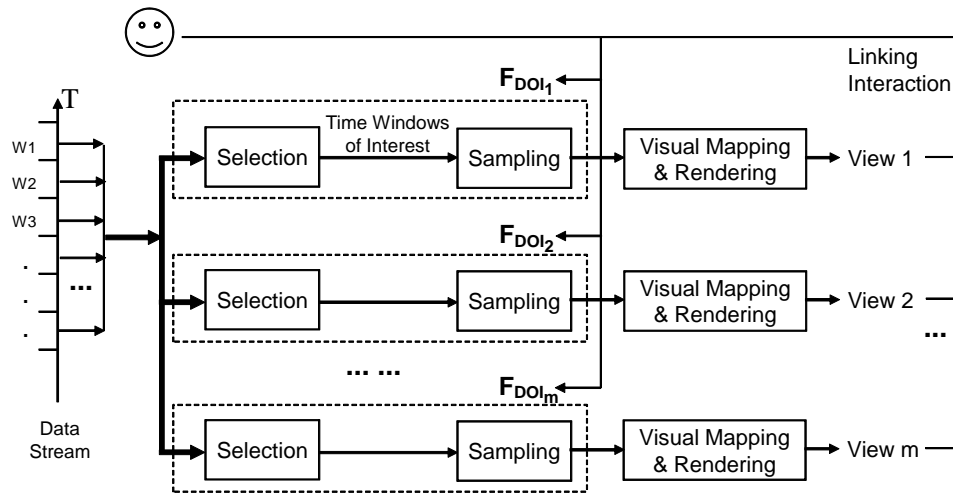


Figure 4.1: The framework of user-driven multiple-view visualizations for data streams.

Figure 4.1 shows the proposed framework. In the data stream, W_0 is the current time window, and W_{-1}, W_{-2}, \dots , are past windows over some bounded duration. The subscript of W means the distance between this window and W_0 . All data in these time windows will be sent to multiple pipelines to generate views. This framework supports linked interaction among multiple views to facilitate users' exploration on data streams. Each pipeline is composed of a selection operator, a sampling operator, visual mapping and rendering. The selection operator selects a range of time windows in which users are interested. Sampling is then applied to the data in the selected time windows. Different windows may have different sampling ratios. Users can define behaviors of selection and sampling operators using DOI functions. Hence, this framework can be called *user-driven*.

Note that all views will get updated when a new time window arrives. The mechanism for this refresh is as follows. Since data streams are dynamic, more datapoints will be

available after a limited time, e.g., the duration of a time window. Then, W_0 will contain the new datapoints in a new time window, and the datapoints in W_0 will go to W_1 , and so on. Thus, the input of each pipeline will change, which causes all views to refresh.

This framework allows users to define multiple DOI functions to get more than one output because users might want to observe multiple changes in one application. For example, during traffic monitoring, users might have two data analysis tasks: (1) identifying how the vehicle speed changes within the last hour, e.g., 5AM-6AM; and (2) comparing the traffic pattern changes within the last hour of today with that of yesterday at the same hour. For the first task, the selector operator needs to select all time windows from 5AM-6AM today. To perform the second task, it is necessary to observe all data during 5AM-6AM of today and yesterday. Obviously, two different DOI functions should be defined.

Figure 4.2 shows the screenshot of the implemented visualization system using the user-driven framework. The DOI function is shown at the top left section and can select the most recent four time windows today as well as in the last two days. So, the traffic pattern changes within recent three days are conveyed accordingly.

In this dissertation, most examples use scatterplots to show 2D relations. This does not nullify the claim that the proposed technique can convey the pattern changes for multivariate data. Reasons include: (1) The above framework and proposed visualization techniques can support any multivariate visualization; (2) The techniques using scatterplots can be easily extended to scatterplot matrices. Actually, two examples using scatterplot matrices and parallel coordinates are discussed later.

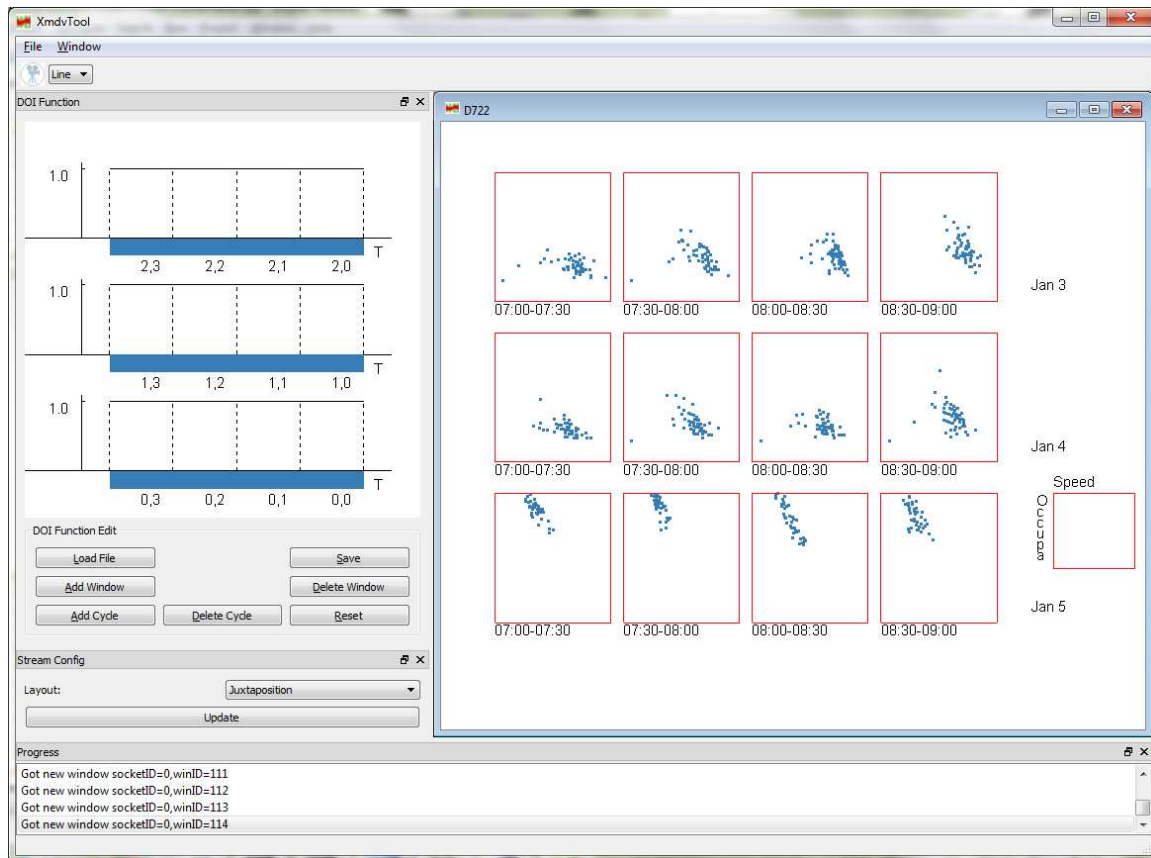


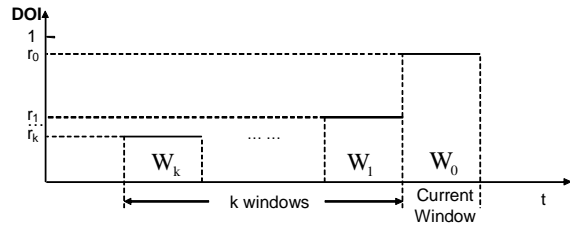
Figure 4.2: The screenshot of the implemented system under user-driven framework using a type PP DOI function to observe the traffic pattern within recent three days.

4.2 DOI Functions

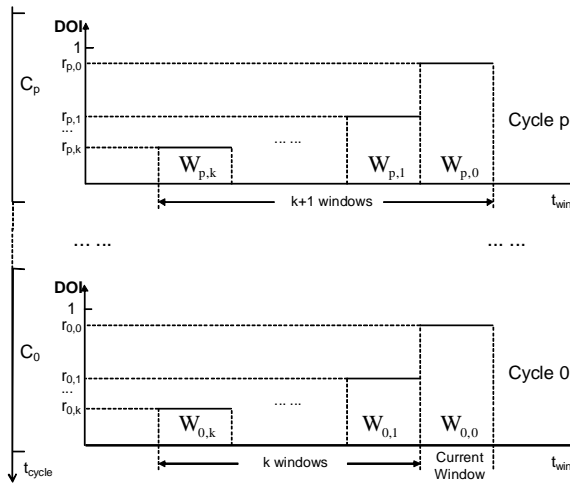
This section describes two types of DOI functions that can be used for some common tasks. As discussed in Section 4.1, the output of a DOI function $f_{doi}(t_d, t_c)$ is a DOI value, which then can be mapped to a sampling ratio via the function $r = f_r(\text{DOI})$. For the sake of convenience, DOI functions are defined in a way that their output is just the sampling ratio.

Type RC (Recent Change) : Figure 4.3(a) shows the curve for this type of DOI functions. It aims to help users study how data patterns change within the recent $k + 1$ time windows. The sampling ratios are r_0, r_1, \dots, r_k in the order from the current window to the oldest one. r_i satisfies $0 \leq r_i \leq 1$ for any i . One common setting is to let $r_0 = r_1 = \dots = r_k = 1.0$. Figure 4.4(c) is generated using this type of DOI function with arguments $k = 2$ and $r_0 = r_1 = r_2 = 1.0$. If there is too much visual clutter and users are less interested in the old data, the setting can be $r_i < 1$ for $1 \leq i \leq k$.

Type PP (Periodic Phenomena) : The DOI functions shown in Figure 4.3(b) can assist users in observing data patterns with periodic characteristics. The data stream is split into multiple cycles (the vertical time axis) with the same length. Each cycle contains multiple time windows (the horizontal time axes). In each cycle, the DOI function has a shape similar to functions of Type RC. The DOI function in Figure 4.3(b) enables users to investigate data patterns within the recent $p + 1$ cycles, namely Cycle 0, Cycle 1, \dots , and Cycle p in the order from the current one to the oldest one. In each cycle, e.g., Cycle i , this function chooses $k + 1$ windows, namely, $W_{i,0}, W_{i,1}, \dots, W_{i,k}$. Note that $W_{0,0}$ is the current window, and $W_{i,0} (1 \leq i \leq p)$ belong to the past cycles, but have the same position in the cycle as $W_{0,0}$. The design of this type of DOI function aims to help users study how data patterns change across both windows and cycles. Consider the example of monitoring traffic. Imagine the current time window is 6:00AM-6:30AM on a Monday.



(a)



(b)

Figure 4.3: Two instances of DOI functions: (a) Type RC (Recent Change); (b) Type PP (Periodic Phenomena).

The current traffic data pattern could be similar to last Monday, and less similar to last Tuesday to Friday, and totally different from last weekend for the same interval (6:00AM-6:30AM). To confirm this assumption, a Type PP DOI function can be defined to choose only time windows corresponding to 6:00AM-6:30AM in these days.

The DOI function is similar to the opacity transfer function in volume rendering [41]. The opacity transfer function assigns an opacity value to a voxel based on the voxel's intensity and can bring out certain features of those voxels having high opacity values. The relationship between the sampling ratio and the window timestamp is like the relationship between the opacity value and the voxel's intensity.

It is true that users might need to compare the data patterns within any two arbitrary windows, and the above two types of DOI functions cannot cover all data stream analysis requirements. However, Type RC and Type PP can satisfy many common tasks under a dynamic context, since users are normally interested in the difference between the current patterns and those of past time windows or cycles for a live stream. When a specific application requires to explore those time windows without the current time window, an arbitrary N-way comparison might be needed. This can be regarded as data analysis on static time-series data because all data is available and it can be visualized in a static view.

4.3 Visualization Techniques

As mentioned in Section 1.2.1, a goal of this research is to visually convey changes of multidimensional correlations. Thus the visualization techniques have been designed with the following question as the main consideration: How should datapoints be organized in different time windows to convey multivariate correlations and the changes of data patterns?

In this section, I will first introduce four layout strategies, namely *superimposition*,

juxtaposition, *step juxtaposition* and *animation playback*, to answer the above question, and then demonstrate their usage with type RC DOI functions. While this dissertation mainly uses scatterplots as examples to explain the principles, these strategies in general can be applied to any multivariate visualization technique. These four strategies will then be extended to type PP DOI functions. Finally, a new visualization technique, namely “embedded views”, is presented based on combining line charts and scatterplot matrices. This is to take advantage of the visual representation capabilities of multivariate and time-series data visualization techniques in one figure.

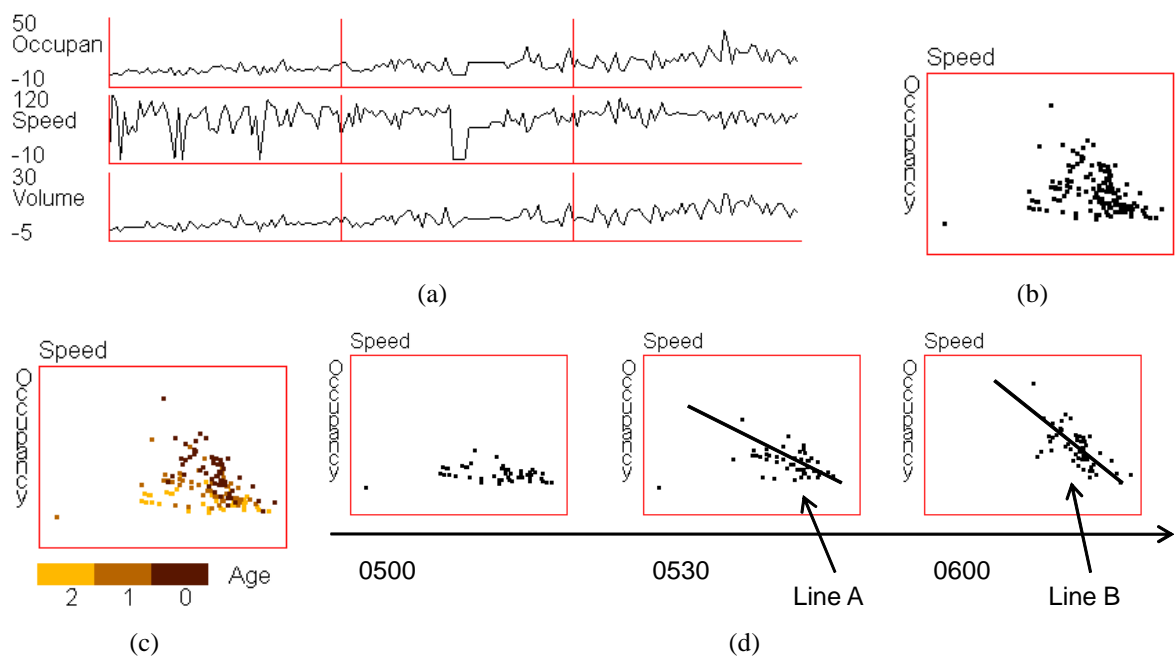


Figure 4.4: This figure shows some of the main ideas of the user-driven approach based on DOI functions using traffic data collected from a highway entrance. (a) A traditional time-series data visualization; (b) All datapoints are shown together in a traditional scatterplot; (c) The ages of data are denoted by colors; (d) Juxtaposition of data in the order of timestamps. Figures (c) and (d) can convey how the fit line slope (lines A and B in Figure (d)) changes from one window to the next, but it is difficult to see this change in (a) and (b).

4.3.1 Layout Strategies

Superimposition: This strategy fuses all datapoints into a single picture, but distinguishes datapoints from different time windows via visual attributes, the choice of which can affect the effectiveness of the final visualizations. In the previous research work, I performed a user study on visual representation of data quality and concluded that color has a stable capacity to convey data attributes under parallel coordinates and scatterplot matrices as long as the visualization is not too cluttered [66]. The reason is probably that it is processed preattentively [61] and does not require extra space. Thus colors were used to convey the timestamps of windows. Figure 4.4(c) is generated by applying superimposition to a scatterplot.

An obvious disadvantage is that displays can become overloaded with too much information, which may result in a longer analysis time. Moreover, if there are too many windows to be chosen in the DOI function and many datapoints from different time windows overlap each other, it is difficult to distinguish between them, even if using colors to convey the window to which each datapoint belongs. Also, limiting the number of colors enables users to quickly differentiate different time windows with higher accuracy. Inherently, this is a visual clutter problem. Ellis and Dix thoroughly discussed current clutter reduction techniques, and put them into eleven categories [20]. Sampling is a commonly-used technique [19, 7, 8] and was reflected in the DOI functions, but cannot reduce the overlapping when data pattern changes are small. Thus I used the filtering approach discussed by Ellis and Dix to extract datapoints in each time window, and generate small multiples, namely juxtaposition and step juxtaposition, to reduce the visual clutter as well as convey the data pattern changes.

Juxtaposition: In this method, one sub-picture is generated using a multivariate visualization for each time window, and then place these figures in order of time (horizontally, vertically, or a grid). In Figure 4.4(d), each scatterplot holds the datapoints from one time

window. Users can see the change of data patterns by comparing three sub-pictures.

Although juxtaposition overcomes some shortcomings of superimposition, it brings two new disadvantages. (1) Recall Figure 4.4(d), in which the dots in the second and third sub-pictures form two lines, A and B. As a recognizable difference exists between the slopes of lines A and B, users can draw conclusions about the change of the linear trend. If this difference is not that big, users may not easily identify the change of line slopes using juxtaposition, as distance exists between these two lines. In the superimposition layout, this difference should be recognized more easily than juxtaposition, assuming there is not too much visual clutter, because one line acts as a reference as the analysts observe the other. Thus superimposition appears to have a stronger capability to help users identify subtle changes of patterns than juxtaposition. (2) If users want to compare the data patterns between two windows, they must move their eyes back and forth. This could make the data analysis tasks cumbersome and might result in a longer response time, especially when there are a large number of windows in the DOI functions.

In order to overcome the shortcomings from both superimposition and juxtaposition, I have designed a third layout strategy that combines the advantages of these two strategies, namely *step juxtaposition*.

Step Juxtaposition: Imagine that the DOI function chooses $k + 1$ (See Figure 4.3(a)) windows to display. I create k sub-pictures: the first shows W_k and W_{k-1} , the second presents W_{k-1} and W_{k-2} , and so on. This strategy uses superimposition to help users compare the data patterns of two consecutive windows, and juxtaposition to reduce possible visual clutter and shorten completion time for data analysis tasks. Figure 4.5 shows an example. More than two windows can be superimposed in one sub-picture in this technique to save display space if there is not too much visual clutter. Note that step juxtaposition can work relatively well even if there is overlapping between consecutive time windows. For example, in the scatterplot 3 of Figure 4.5, one finding is that the dark dots

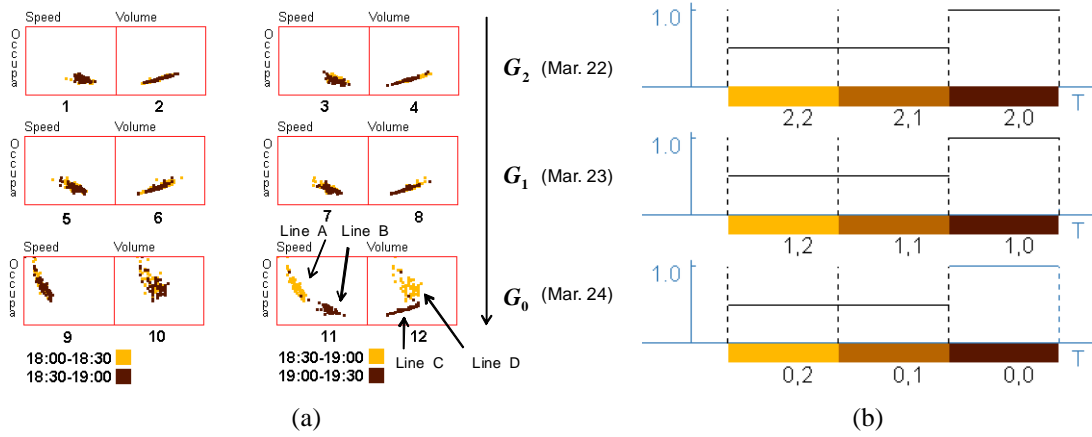


Figure 4.5: A step juxtaposition output using a type PP DOI function as shown in Figure (b). The cycle length is one day. All windows are put into three groups, G_0 , G_1 and G_2 , corresponding to three days, March 24, 23 and 22, respectively. In Figure (a), one can see clearly how data patterns change within the recent three time windows for March 24. However, data patterns do not have significant changes on March 22 and 23.

(19:00-19:30) hide almost all yellow ones (18:30-19:00). Actually, the data in the time window 18:30-19:00 is just the dark ones in scatterplot 1. Actually, there is no pattern change between these two time windows. To be general, if users want to observe n time windows and data patterns do not change too much in the corresponding period, $n + 1$ time windows can be chosen in the DOI functions to relieve the impact of overlapping in the step juxtaposition.

A more convincing example is shown in Figures 4.6 and 4.7, where a slice of traffic data (Sensor D722, Feb. 16, 2009) is used. The DOI function is of type RC and 25 windows are selected. Imagine users were asked to find when the fit line slope changes from one window to the next. This is definitely impossible if using superimposition since human eyes cannot effectively distinguish 25 colors in one figure. In figure 4.6, it is an arduous task because of the huge number of windows. However, in Figure 4.7, this task becomes much easier. In each scatterplot, users only need to use light yellow datapoints as the reference and observe dark yellow dots. One not only can find obvious changes

from Window 05:00-05:30 to Window 05:30-06:00, and from Window 05:30-06:00 to Window 06:00-06:30, but can also perceive tiny changes from Window 06:00-06:30 to Window 06:30-07:00, from Window 09:00-09:30 to Window 09:30-10:00, and from Window 09:30-10:00 to Window 10:00-10:30. These tiny changes are almost impossible to detect using juxtaposition (Figure 4.6). In the section on the user studies, the experimental result will show that step juxtaposition can help users obtain a much higher response accuracy than juxtaposition and shorten completion time for data analysis tasks.

Compared to superimposition, juxtaposition and step juxtaposition need more display space. Therefore, if users want to observe the tiny data pattern change, they should select fewer time windows in the DOI functions.

Animation: It is an intuitive idea to play the data pattern change using an animation, with each frame representing a time window. Animation combines the benefits of the prior three visualization techniques:

- (1) Because of the short term memory of the human visual system, users can normally memorize the previous frame in the animation when the current frame is shown to us. Thus it has similar capabilities to convey data pattern change as superimposition and step juxtaposition.
- (2) Compared to superimposition and step juxtaposition, animation can avoid the visual clutter caused by overlapping datapoints from different time windows.
- (3) Unlike juxtaposition and step juxtaposition, animation still uses a canvas having the same size as superimposition, which can also avoid the possible visual clutter caused by a smaller canvas size.

However, animation might delay the data analysis tasks, especially when the number of displayed windows is large. The reasons include: (1) Users need to frequently play the animation multiple times to confirm what they found. (2) A window ID must be shown



Figure 4.6: A juxtaposition output using the traffic data from sensor D722 on Feb. 16, 2009. Assume that the data analysis task is to detect the slope changes for fit lines of linear trends between consecutive time windows. It is difficult to detect tiny slope changes. Moreover, even for an obvious change, it takes a long time.

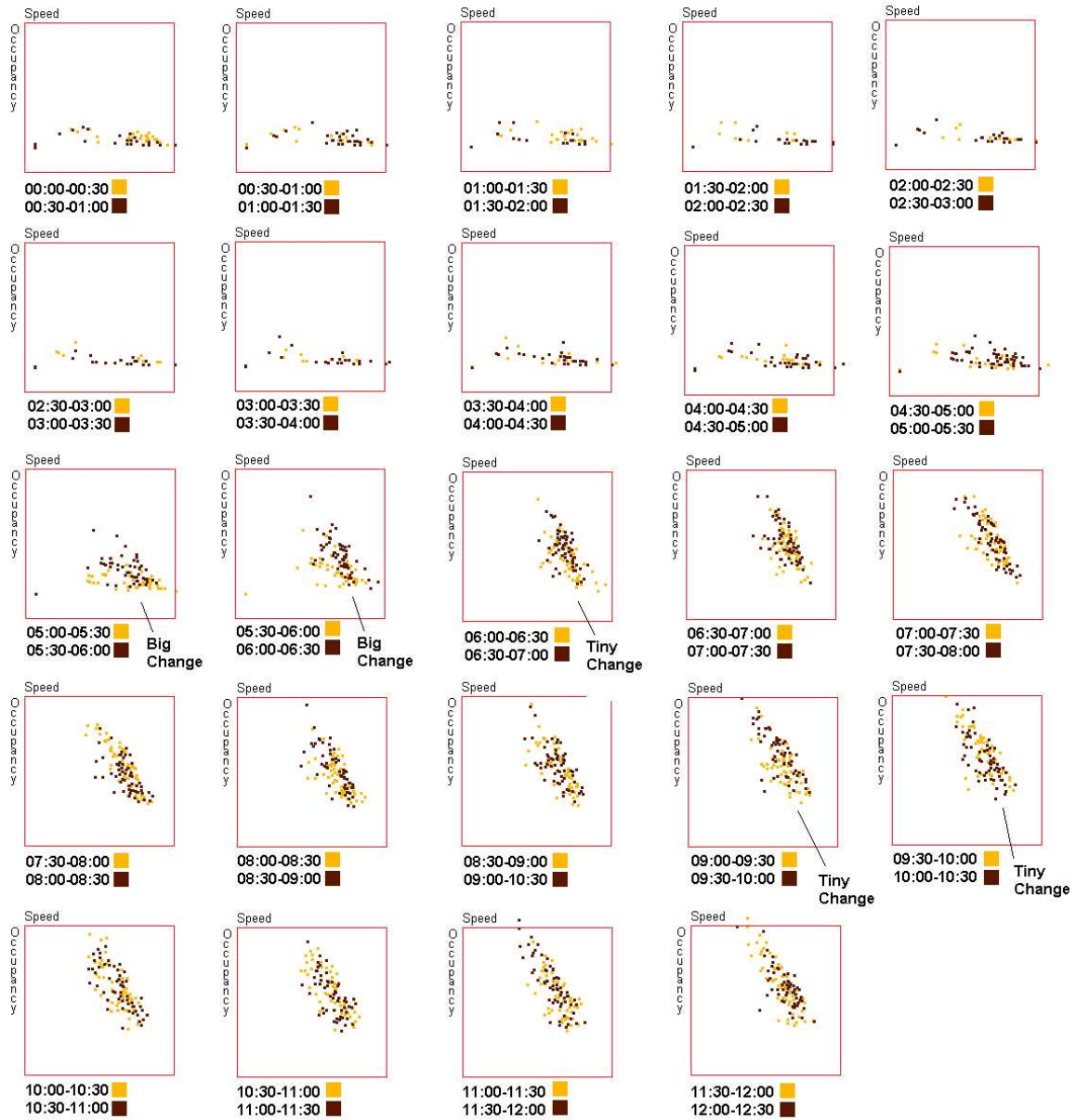


Figure 4.7: A step juxtaposition output using the same data as Figure 4.6. One can easily and quickly find when the slope of the fit line for a linear trend changes.

together with the visualization, so users know what window they are viewing. Thus users have to stay aware of this caption while watching the animation and cannot fully focus on the data patterns.

I have discussed the advantages and disadvantages for each layout strategies. How can developers choose one of them for a real application? Section 4.5 will describe an experiment to compare their representation capabilities, and then derive a guide to advise analysts on choosing appropriate techniques for their data analysis requirements.

4.3.2 Extension of Layout Strategies to Type PP DOI Functions

I have discussed four layout strategies and showed their usage together with type RC DOI functions. In theory, one can directly apply these layout approaches to the scenarios with type PP DOI functions. For example, in a type PP DOI function, users might choose two cycles and three time windows per cycle, and thus in total six windows would be shown. If a real system directly utilizes the proposed layout strategies to handle these six windows, users could retrieve the information they want. However, this initial idea is not efficient compared to the alternative approach of grouping and then visualizing them. This approach is based on the fact that users normally have two types of interests: (1) the pattern changes across windows in the same cycle; and (2) the change of patterns across cycles in the same time period, such as window 1 and window 4. If users are interested in (1), one approach is to organize windows into two groups: (a) windows 1 & 2 & 3; and (b) windows 4 & 5 & 6. For the second task, all windows can be split into three groups: (a) windows 1 & 4; (b) windows 2 & 5; and (c) windows 3 & 6. The rationale is to put those windows where users want to detect pattern changes into the same group. Then the four proposed layout strategies can be used to visualize each group. Therefore, users can observe each group and try to extract information of interest. Obviously, this grouping approach makes the pattern change analysis easier than the initial non-grouping method.

For superimposition, it can decrease the number of time windows in one figure; for the other three layout strategies, the grouping approach will put together only those windows in which users want to detect the pattern changes.

To be general, two grouping approaches are provided, namely GA1 and GA2 (Figure 4.8) for the type PP function shown in Figure 4.3(b).

GA1: If the data analysis task focuses on the pattern change across windows within one cycle, $p + 1$ groups (G_0, G_1, \dots, G_p in Figure 4.8) will be provided. Actually, each group contains all windows in one cycle. An example of this grouping approach is shown in Figure 4.5.

GA2: If users are interested in changes across cycles, $k + 1$ groups (G'_0, G'_1, \dots, G'_k in Figure 4.8) are generated. Every group has $p + 1$ windows, each of which belongs to a cycle. All windows in one group are in the same position within the cycles.

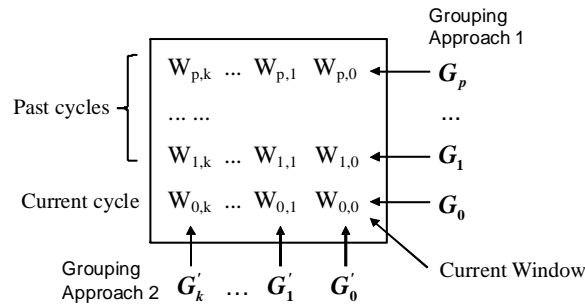


Figure 4.8: Two grouping approaches to helping users achieve various data analysis goals.

4.3.3 Integrating Time-series and Multivariate Data Visualizations

All of the above techniques assumed the use of extended multivariate visualizations to convey multi-dimensional correlations. Another normal data analysis requirement in exploring data streams is to observe the trends for each dimension. This can be achieved using traditional time-series data visualization techniques such as line charts and heatmaps.

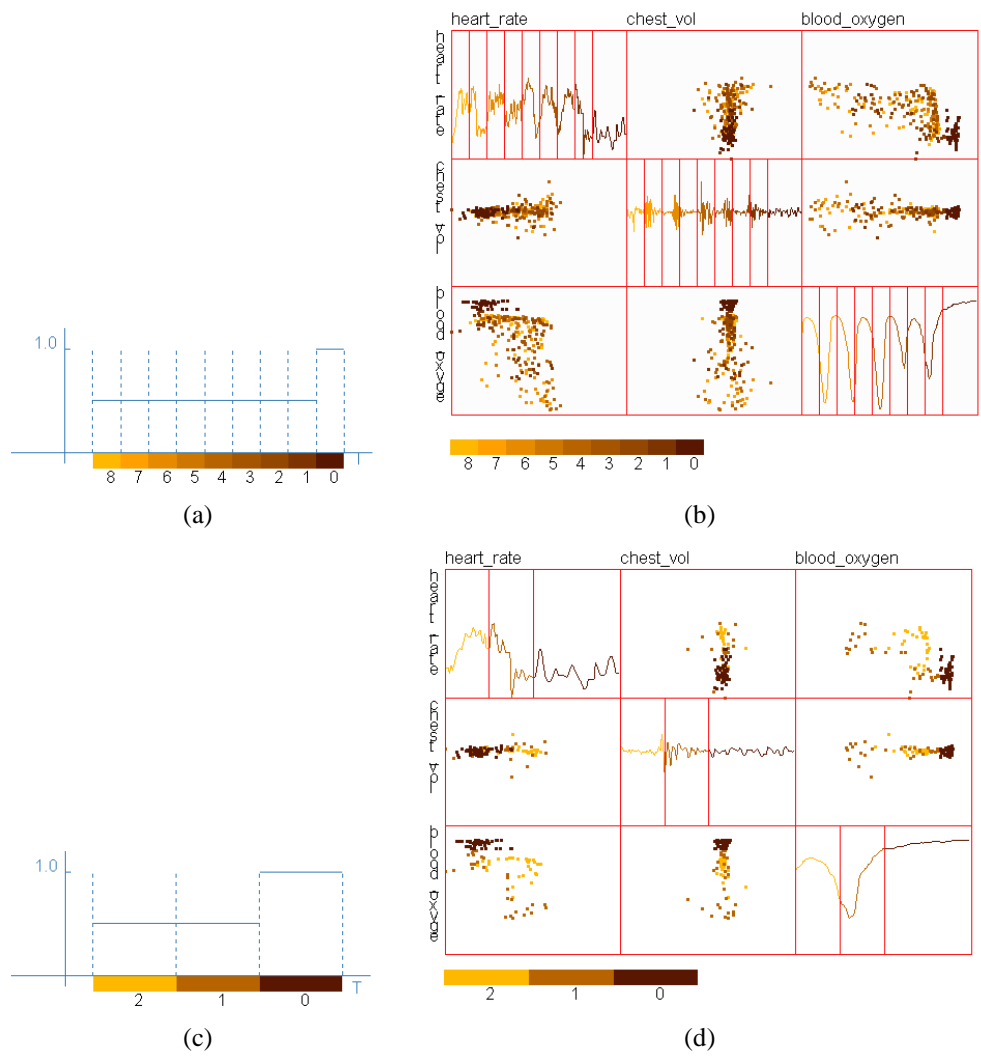


Figure 4.9: The embedded views for the sleep data stream. Users not only can see how clusters move over time in the scatterplots, but can also see the trends for each dimension via line charts in the diagonal plots. Figure (b) is generated using the DOI function shown in Figure (a), which chooses the recent 9 windows to display. After the user uses the DOI function interactive tool to adjust the function to Figure (c), a new view (Figure d) will be shown. Users can more clearly see the movement of clusters on Figure (d) than Figure (b).

It is true that one solution is just to put line charts and a scatterplot matrix side by side to convey both the trends for each dimension and multi-dimensional correlations. This requires decreasing the canvas size for each visualization since the total canvas size is normally fixed, e.g., the monitor size. To overcome this shortcoming, a novel technique, namely *embedded views*, is proposed to embed line charts into scatterplot matrices in order to save canvas space. This approach is adapted from the enhanced scatterplot matrices of Cui et al. [18], who introduced 0D, 1D and 2D visualizations, including histograms, line charts and images, into the diagonal plots. The difference is that I use the DOI function to partition each diagonal cell based on the number of windows being viewed.

Figures 4.9(b) and 4.9(d) show two embedded views using the sleep data stream. They use the DOI functions shown in Figures 4.9(a) and 4.9(c), respectively. From these two views, one can clearly see how a single cluster moves over time in the scatterplot. In addition, this can be used to study the trends for each dimension via line charts.

4.4 DOI Function Interaction Tool

Although two pre-defined types of DOI functions have been described in Section 4.2, it is necessary to enable users to define DOI functions by themselves to analyze data streams in different applications. Moreover, it will make the system much more useful to allow users to adjust the DOI functions interactively. Basically, visual analysis based on the DOI function is often a trial and error process. It is normal that analysts do not know the exact characteristics of the data patterns and how these patterns change prior to exploring the data streams. By allowing users to adjust the DOI functions, analysts can select a predefined DOI function first, and then adjust it to find useful data patterns while the system is running. Some possible adjustments to facilitate exploration include: (1) Increasing the sampling ratio to see more details or decreasing the ratio to avoid visual

clutter. (2) Changing some of the arguments for pre-defined types of DOI functions. For example, if the number of time windows to be displayed for the type RC DOI function is large, say 9, but most important changes occur within the recent two or three windows, the user can reduce it and observe the changes in more detail.

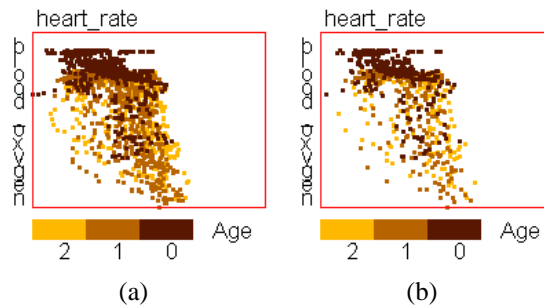


Figure 4.10: Using DOI functions to reduce visual clutter on a sleep data stream. (a) All datapoints are displayed; (b) Sampling is applied to each time window based on the DOI function after user adjustment.

I designed an interface to enable users to change the DOI function interactively. Using this tool, users can (1) drag the DOI function curve to change DOI values for a particular window; (2) save or load a DOI function to/from a file; (3) add/delete a window;(4) add/delete a cycle (only applicable for type PP function); and (5) reset the DOI function to the original state. Figures 4.9 and 4.10 show the effect of using this tool. Figure 4.10 shows the effect of reducing the number of windows. In Figure 4.10(a), the arguments for the DOI function are set as $r_0 = r_1 = r_2 = 1.0$, but they are changed to $r_0 = 0.5$ and $r_1 = r_2 = 0.33$ in Figure 4.10(b).

4.5 Evaluation 1: User Studies on Layout Strategies

For the four layout strategies introduced to utilize traditional multivariate visualizations for conveying the pattern changes in data streams, there are two important questions to be answered:

- Are the proposed techniques significantly better than traditional time-series data visualization techniques, such as line charts and heatmaps, in conveying the changes of multi-dimensional correlations?
- Which one among the proposed layout strategies is the best to convey pattern changes for particular datasets and patterns?

In order to answer these two questions, a user study was performed to observe participants' capabilities in detecting pattern changes. This experiment compared the effectiveness of the four proposed layout strategies with that of traditional time-series data visualizations, including line charts and heatmaps. The experimental results help validate the effectiveness of the proposed techniques, and helped in deriving a guide for choosing layout strategies based on the characteristics of data analysis tasks and datasets. These results can provide potential benefits to both data analysts as well as visualization system designers.

4.5.1 Experimental Design

The basic procedure used to design the experiment is as follows: (1) Choose some commonly-used data patterns that can be defined easily and clearly; (2) Construct streaming datasets with changes in selected data patterns between time windows; (3) Generate visualizations using the proposed techniques, as well as line charts and heatmaps; (4) Design questions for subjects in the experiment regarding the pattern changes in the generated visualizations.

In the experiment, users' response accuracy as well as response time were collected. It was assumed that higher accuracy and lower response time indicate an effective technique. Whether a proposed technique is good depends on many aspects, such as the selected data patterns and the magnitude of pattern change. In this experiment, many combinations

of these factors were tried, and how they affected users' responses was observed. It is impossible to try all combinations, thus the experiments aimed to test the most common ones to guide most data analysis tasks.

Choosing Data Patterns: Prior figures showed two types of data pattern change: the slope change of linear trends (Figures 4.4, 4.5), and the movement of a single cluster (4.9, Figures 4.10). They are both very common in real applications and are easy to explain to participants, even without any prior experience in visual data analysis. There are other types of changes, such as the offset change of fit lines representing linear trends, and the expansion or shrinking of clusters. Different types of data patterns might be similar to each other, e.g., the offset change and the slope change of a linear trend. Therefore, some results on evaluating slope changes may be borrowed when a system need to be designed to help users detect offset changes in linear trends. If a new data analysis task is totally different from the tested data pattern changes, a new experiment can be performed with this user study as a design guide.

Note that observing the change of line slope and movement of a single cluster are low-level tasks. In most real data analysis tasks, people normally do not know which low-level task to choose until after observing the overview of the data. In this experiment, participants do not need to decide which low-level tasks to choose. This is normally impossible in the real applications. For example, for the traffic data, users first investigated the data in different time windows, then determined that a linear trend existed between two variables, and finally decided to observe the line slopes for retrieving pattern changes over time. However, the experiment design will not impact the credibility of this user study. The reason is as follows: this user study is similar to a perception experiment. Its main goal is to check human capabilities to distinguish data pattern changes on four types of layout strategies. Thus, the performance of participants on these low-level tasks is just what I want to analyze in this experiments.

Constructing Datasets: The basic idea for constructing a dataset is as follows: (1) Pick a specific time window, namely W_0 , from a real dataset and regard it as the first window of the experimental data. (2) Construct several time windows, from W_1 to W_{n-1} , based on the initial window. Note that n is the number of windows shown to participants. The selected pattern is changing from W_i to W_{i+1} for any i that satisfies $0 < i < n - 1$. (3) Generate the final dataset by composing the windows from W_0 to W_{n-1} into one single stream. An example dataset is shown in Figure 4.11. It is generated from a snapshot of the traffic data with changes to the linear trend. Figure 4.11(a) corresponds to a subset of the real traffic data, while Figures 4.11(b) and 4.11(c) are generated using synthetic time windows adapted from the data in Figure 4.11(a).

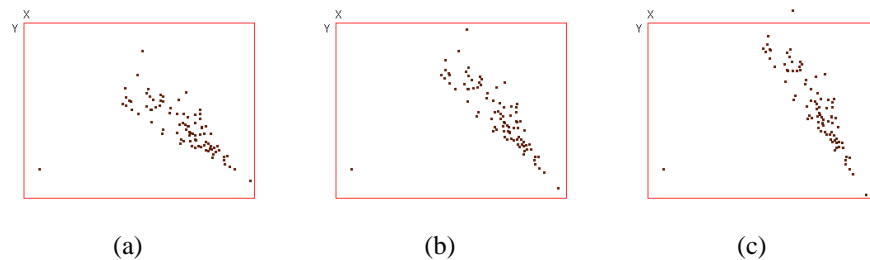


Figure 4.11: Figures (a) - (c) show three time windows of a streaming dataset. Figure (a) is extracted from a traffic data stream. The datapoints in Figures (b) and (c) are constructed from those in Figure (a) by changing the fit line slope as explained in Figure 4.12.

In the above step (2), multiple variations of the experimental datasets were generated by using different change magnitude and changing the number of windows. The reason why to choose these two variables is that adjusting these two factors can help distinguish which layout strategies are better than others for conveying changes for particular data patterns. In some preliminary results, if the change magnitude was big enough and there were only two time windows, all of four proposed layout strategies worked very well. However, small pattern changes make juxtaposition fail since users cannot distinguish the tiny difference between two similar figures, and too many time windows will produce too

much overlapping in superimposition.

One could argue that noise should be added when constructing datasets for the evaluation. This is not necessary. The reason is that the lack of noise will not impact the validity of these user studies. If the goal is to visualize the linear trends or clusters, noise has to be introduced to increase the credibility of the user studies. However, in this dissertation, I focus on how to convey the changes for the particular data patterns, e.g., the linear trend in this experiment. I want to observe how two factors, the change magnitude and the number of windows, affect users' capabilities to detect and estimate this change. Adding noise could make the tasks more difficult, but cannot add more credibility to this experiment.

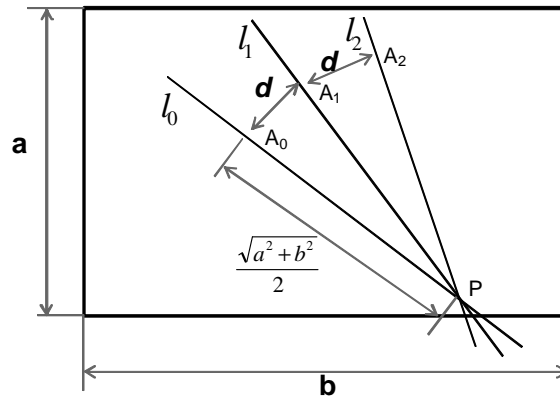


Figure 4.12: This figure shows the methodology applied to constructing a streaming dataset having three time-windows from a real dataset. The straight line PA_0 represents the linear trend with which the datapoints in a specific time window agree. PA_1 and PA_2 correspond to the linear trends of two synthetic time windows. Four or five windows were constructed for some questions.

Figure 4.12 shows how to determine the magnitude of the pattern change (in this case, the linear trend) between contiguous windows. The straight line PA_0 represents the fit line of the linear trend for the initial window. In this line, the point P is the intersection of fit lines for two contiguous time windows in the real traffic data, one of which is the initial window. For example, if one picks the second time window in Figure 4.4(d) as the initial

window, the point P is the intersection of lines A and B . The distance between P and A_0 is half of the diagonal line for the scatterplot. The fit lines for windows from W_1 to W_{n-1} were then constructed. Figure 4.12 contains two manually constructed fit lines, $l_1(PA_2)$ and $l_2(PA_2)$, for W_1 and W_2 . Note that $|PA_0| = |PA_1| = |PA_2|$ and $d_1 = |A_0A_1|$ and $d_2 = |A_1A_2|$ represent the change magnitude. In the part of this experiment for linear trend patterns, I used the combination of three types of change magnitude (1, 4 and 12 pixels), and three time window counts (3, 4, 5 windows).

The construction of datasets for cluster motion patterns is similar to the above process. Note that only one single cluster is shown in each window and the cluster size does not change across windows. Larger numbers of clusters and varying sizes will be tested as part of the future work.

Generating Visualizations and Questions: In order to make the comparison among the user responses for different techniques meaningful, I followed several rules:

(1) Color Scheme: In superimposition and step juxtaposition, the selection of the color scheme can significantly affect participants' capabilities to detect pattern changes. Thus the same color scheme was applied to all visualizations generated using superimposition and step juxtaposition. Specifically, I selected a color ramp, utilized the colors at the two ends in the step juxtaposition, and chose the linear interpolated colors based on RGB color space for the superimposition.

(2) Canvas Size: Because a small canvas size can lower response accuracy and increase the response time because of possible overlapping by the data of different windows, the total canvas size was fixed and a specific canvas size was assigned to each scatterplot based on the layout strategies. To be specific, the superimposition and animation use the total canvas size, but juxtaposition and step juxtaposition were in a grid while maintaining the ratio between width and height for each scatterplot. The total size of the grid is equal to the total canvas size. For example, if 5 windows are needed in generating a juxtaposi-

tion, the total canvas was split to a grid having 9 (3×3) cells (Figure 4.13). Although this wasted four cells, it maintains the shape of scatterplots.

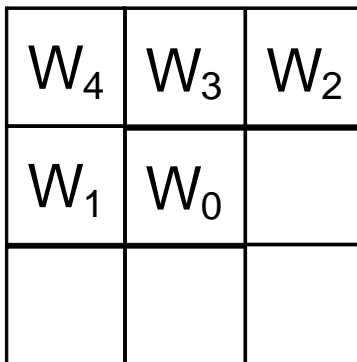


Figure 4.13: This figure shows how to split the whole canvas into multiple cells for juxtaposition and step juxtaposition. If five subfigures need to be shown, a grid having nine cells are constructed. Although four of them are wasted, this approach can maintain the shape of each scatterplot.

(3) Point Size: The point size must be appropriate to convey data patterns in scatterplots since small dots are difficult to distinguish and big dots could result in too much overlap. After some initial experiments, I chose suitable sizes for each of the proposed layout strategies. In particular, the superimposition and animation used 4×4 pixel points and the juxtaposition and step juxtaposition used 3×3 pixel points, because the latter ones occupy a smaller region of the display.

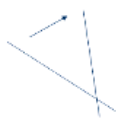
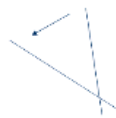
The design of questions for the user studies was straightforward. Participants were asked to answer only multiple choice questions to qualify the pattern changes instead of quantify them. This aims to make the user studies convenient and friendly. For example, for linear trends evaluation, Subject only needed to identify how the fit line slope changes (increasing or decreasing) between two specific contiguous windows. However, it is almost impossible to perceive a fit line or a cluster in line charts and heatmaps. Thus, instead, some equivalent questions were provided. For linear trends, participants were asked to estimate the rate of change for one variable with respect to the change of the

other, as this can be regarded as the fit line slope. Then users were asked to report how this rate changes. In the cluster movement questions, users were asked to estimate how the average value of one variable changes from one window to the next.

Figure 4.14 shows a real question used in this experiment.

Question 1

Assume all datapoints in each sliding window agree with a linear model via a fit line. Which of the following options best describes the change of slope for fit lines from window 1 to window 2?

- (a)  (b)  (c) Unchanged; (d) I do not know.

Submit

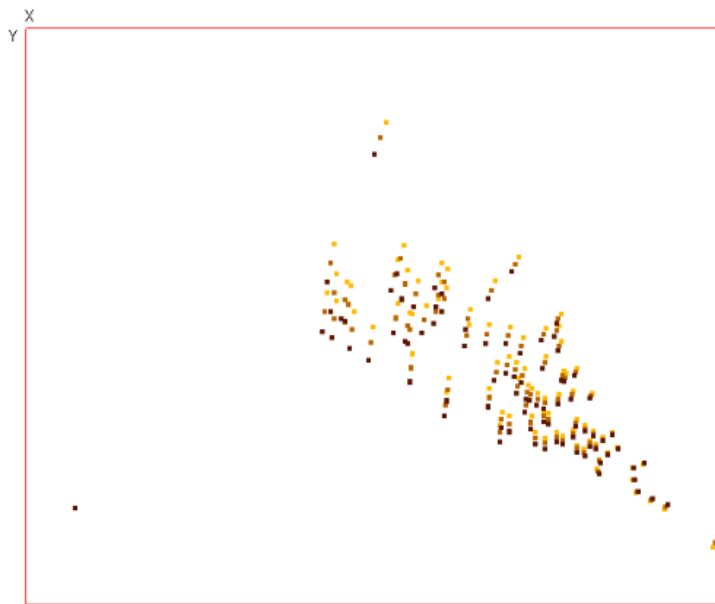
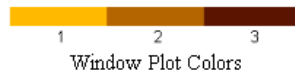


Figure 4.14: This is a question used in the user study for the user-driven framework. The figure is generated using the superimposition technique. Participants are asked to identify how the fit line changes from window 1 to window 2.

4.5.2 Experimental Settings

In total, 14 computer science students participated in the user studies. Two of them were undergraduate students, while the others were graduate students. I first gave a short introduction and showed some sample questions to each student, and then asked each to finish two groups of questions. Each group has 33 questions. One group was for linear trends and the other pertained to cluster movement. To avoid the side-effect of a learning curve, all questions in each group were shuffled for each participant. All questions were shown to the subjects on the same laptop.

4.5.3 Experimental Results

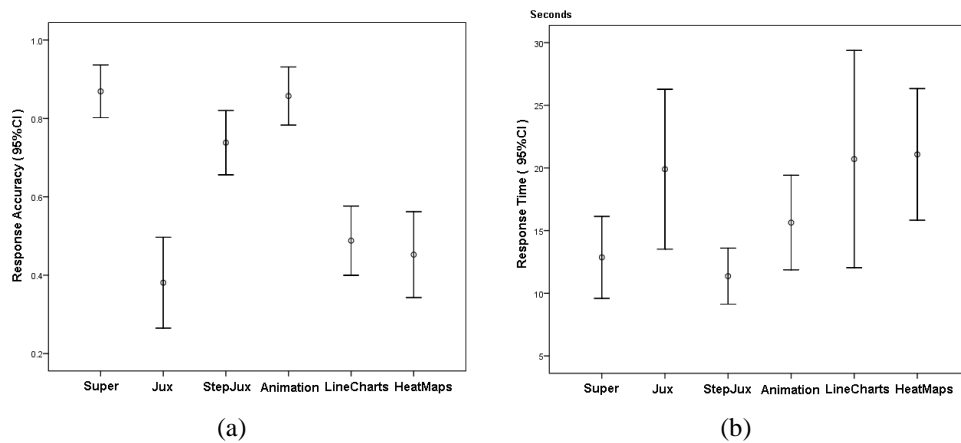


Figure 4.15: The experiment results for all participants and questions: (a) response accuracy; (b) response time.

Result 1: Figures 4.15(a) and 4.15(b) show the mean values with a 95% confidence interval of response accuracy (RA) and response time (RT) for all participants and questions. The paired samples t-test was applied to the experiment results to compare RA and RT values for different visualization techniques. The following conclusions were drawn: (1) From the aspect of RA, superimposition, step juxtaposition, and animation are all signifi-

cantly better than juxtaposition, line charts, and heatmaps ($p < 0.001$). Since every question has only three choices, the performance of juxtaposition, line charts, and heatmaps was deemed not acceptable within the experimental configuration because their RA mean values are less than 50%. Thus three of the four proposed techniques conveyed multi-dimensional correlations much better than line charts and heatmaps. (2) Superimposition and animation have a little bit higher RA values than step juxtaposition ($p = 0.02$ and 0.05). but superimposition is not significantly different from animation ($p = 0.67$). (3) For the RT values, participants spent less time on step juxtaposition than superimposition and animation. However, superimposition is not significantly different from animation ($p = 0.10$), while the difference between step juxtaposition and animation is significant ($p = 0.005$). Therefore, for tasks requiring users' quick response to pattern changes, step juxtaposition is a good option.

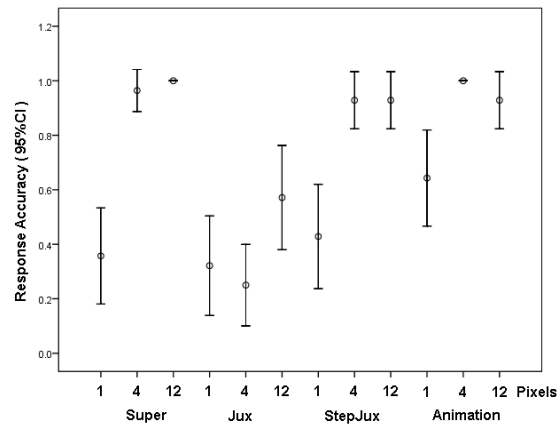


Figure 4.16: The response accuracy for all participants and datasets having only 3 time windows. The numbers on the horizontal axis mean the change magnitude in the unit of pixels.

Result 2: In order to see how the magnitude of pattern change affects participants' performance, I calculated the mean values with a 95% confidence interval of RA values grouped by the combination of layout strategies and change magnitude. The results are shown in Figure 4.16. The numbers, 1, 4, and 12 on the horizontal axis denote the change

magnitude in terms of the number of pixels. This demonstrates two facts:

1. Participants have improved performance when the change magnitude (the number of pixels) becomes bigger, except for juxtaposition. The difference between a 1 pixel change and a 4 (12) pixel change is significant for all layout strategies ($p < 0.02$) except juxtaposition. Moreover, the RA values for 4 and 12 pixel change are close to 100% for all layout strategies except juxtaposition. However, the RA values for a 4 pixel change is not significantly worse than those for a 12 pixel change.
2. For the 1 pixel change, animation has the highest RA values, and was significantly better than superimposition and step juxtaposition ($p = 0.04$ and 0.03). Subjects had a response accuracy of about 65%. Considering that the point size is 4×4 for animation, this is a very good result. The reason is obvious: when the change is very small, the similarity between the datapoints of two windows results in too much overlapping. Thus participants cannot perceive subtle changes from the figures generated using superimposition and step juxtaposition. However, animation can avoid the overlaps and still convey the pattern changes because of the short-term memory of the human brain.

Based on the above observations, the following conclusions can be drawn:

1. Under the experimental configuration, superimposition, step juxtaposition and animation can work very well for changes bigger than or equal to 4 pixels.
2. Animation can work relatively well even if the change is smaller than the point size, while superimposition and step juxtaposition cannot.

An interesting result of this experiment is that juxtaposition is not a good option for conveying pattern change. It was expected at least to be better than superimposition because it could relieve visual clutter and make the pattern changes obvious. However,

the experimental results reveal that it is not as good as the other three techniques. My guess is that human eyes cannot detect changes from one figure to the other without an appropriate reference if the change is very small. Recall that participants were asked to detect the slope change of a linear trend. One can observe one time window by treating the datapoints in the other time window as a reference in superimposition, step juxtaposition, and animation, because datapoints of two time windows are put in the same scatterplot. Note that when an animation shows the second frame, the first frame can still be used as a reference because of human short-term memory. This makes it easy to perceive the pattern changes. However, if using juxtaposition, it is difficult to use such a reference because two windows are separated from each other. One possible solution to improve juxtaposition is to add reference grid lines for each individual scatterplot. In this way, users can more easily estimate the parameters for the data patterns in each window, including the slope of fit lines and the distance between a cluster and the scatterplot border. This solution has an obvious disadvantage: grid lines can cause visual clutter and thus counteract their benefits. This should be tested in an experiment, which is planned as future work.

4.5.4 Evaluation Summary and Implications

For question 2 stated in the beginning of this section, I derived a set of guidelines to advise data analysts and visualization system designers to choose appropriate layout strategies with the main goal to increase the response accuracy. Table 4.1 shows this guideline.

The number of windows involved in pattern changes	The magnitude of the pattern change	
	Small	Large
Small	Animation	Superimposition & step juxtaposition
Large	Animation	Step juxtaposition

Table 4.1: A guideline to advise data analysts and visualization system designers on choosing appropriate layout strategies in terms of the characteristics of datasets and data analysis tasks.

I recommend the use of animation when the change is small, because animation is the only technique that appears to work relatively well in this case. Note that animation could cause a long response time when the number of windows is huge, so superimposition and step juxtaposition should be used when the change magnitude is big. Superimposition does not work well when users choose too many windows in the DOI functions, as it tends to cause serious visual clutter and humans cannot readily differentiate many colors at once. In this situation, step juxtaposition is a better choice. A key question is what this threshold may be at which superimposition becomes problematic. It is almost impossible to give such a number for all visual analysis tasks, because it depends on many factors, including the selection of color scheme, canvas size and the degree of visual clutter. For a specific use case, users or system designers would need to conduct an experiment to determine this number.

Table 4.1 divided the change magnitude into two types, namely, small and large. Based on the experimental results in Figure 4.16, only animation can work relatively well when the pattern change is smaller than the point size. So the recommendation is that if the change is smaller than the size of the visual items, it is regarded as small and the suggestion is to use animation to observe the pattern changes.

One might argue that small changes can be clearly observed just by zooming in all views. This is not feasible for certain circumstances because users might not have enough time to zoom in views when data arrival rates are relatively high.

4.6 Evaluation 2: Case Studies on Interaction Techniques and Other Multivariate Visualizations

In this section, I analyze the effectiveness of the proposed visualization and interaction techniques. Because the effectiveness of the proposed visualization techniques in the user

studies was shown in Section 4.5, I will mainly focus on demonstrating the usefulness of DOI functions and their interactions in three case studies. The first will use a type PP DOI function, while the second focuses on the trial-and-error exploration process using the DOI function interaction tool. Since examples so far used only scatterplots, the third case will demonstrate the integration of step juxtaposition into parallel coordinates to show that any of the proposed layout strategies can be applied to other traditional multivariate visualization techniques.

Case Study 1: Figure 4.5 uses the measures from sensor D191 (close to the intersection of I-35W and 35th Street) in the traffic data stream. A type PP DOI function is used as shown in Figure 4.5 (b). The length of a cycle is one day. The current window is 7:00PM-7:30PM on Tuesday, March 24, 2009. In the implemented system, scatterplot matrices were used to show this example. For the sake of saving space, I chose two interesting subplots from each scatterplot matrix to form Figure 4.5(a). It is not difficult to find that the traffic pattern on March 24 conveyed by the scatterplots at the third row is significantly different from patterns on March 22 and 23 as shown by the scatterplots in the first and second row. In the scatterplots formed by *Speed* and *Occupancy*, These two variables always have a negative relationship with each other. However, the fit lines in scatterplot 9 have a larger slope than those in scatterplots 1 and 5. Two fit lines (lines A and B) can be observed in scatterplot 11. The absolute value of the slope for line A is much larger than that for line B. Line B is formed by the points with dark color. It shows that the linear relationship between *Speed* and *Occupancy* was restored to normal after 7:00PM, as it has a similar slope to the fit lines in scatterplots 3 and 7. Thus, the conclusion is that traffic was very heavy (high *Occupancy* and low *Speed*) before 7:00PM and then went to normal. Such a change of data patterns did not happen on March 22 and 23, i.e, the traffic from 6:00PM to 7:30PM was not that heavy.

Some interesting changes of data patterns happened in the scatterplots formed by

Volume and *Occupancy*. In scatterplot 10, the negative linear relationship exists between *Volume* and *Occupancy*, but the relationships in scatterplots 2 and 6 are positive. From scatterplot 12, it is observed that this linear relationship changed after 7:00PM. In this scatterplot, points with dark color formed line C, whose position and shape are similar to the points in scatterplots 4 and 8. Since line C contains the data of the current window (7:00PM-7:30PM), it also shows the traffic became normal after 7:00PM. This change is different from the change in scatterplot 11, as the linear relationship changed from a negative one to a positive one. This inspired me to consider whether some abnormal things happened. My hypothesis is that this is a sign of a traffic jam caused by some special reasons, such as an accident or road construction. The reason is as follows. In scatterplots 2, 4, 6, and 8, some points have high occupancy, and others have lower occupancy. Thus the traffic is always oscillating from 6:00PM to 7:30PM on March 22 and March 23. However, the relationship between *Volume* and *Occupancy* are always positive. Recall the definition of *Volume* in section 3.1. This shows that vehicles still can run at a relatively high speed when the *Occupancy* is high, which results in more vehicles passing the sensor (high *volume*). However, in scatterplot 10 and line D in scatterplot 12, *Volume* became lower when the *Occupancy* was very high. This could happen during a traffic jam. For example, when police cleaned the highway after an accident, drivers could drive in only one lane, then the average speed of vehicles is very low, so the *Occupancy* is very high (close to 100%) and the *Volume* is close to 0. Incident records of Mn/DOT showed that flooding happened in the late afternoon on March 24, near the crossing of I35W and 42nd Street, because of 0.44 inch of precipitation on that day. This is very probably the reason for the interesting changes of data patterns as shown by Figure 4.5.

The analysis of this example confirms that the developed techniques can convey not only the multidimensional correlations for a particular time period, but also the change of data patterns, and even *the change of the changes of data patterns* when using the type PP

functions.

Case Study 2: This case study investigates Figure 4.9 using the sleep data stream. The current window is 36.5-37 minutes after the beginning of this sleeping experiment. Figures 4.9(b) and 4.9(d) use embedded views to convey not only trends for three dimensions but also multidimensional correlations. Figure 4.9(b) was generated using a DOI function to compare the recent 9 time windows as shown in Figure 4.9(a). From each scatterplot one can quickly find how the primary data in each window moves over time. For example, in the plot with the heart rate (*heart_rate*) as X and blood oxygen concentration (*blood_oxygen*) as Y, one interesting pattern is that the older data mainly falls into the bottom area, then slowly moves in the upper-left direction, and finally returns back to a middle position. The line charts in the diagonal plots show that the heart rate decreased to a minimum value and then slowly went up within the recent time windows. At the same time, the blood oxygen concentration reached a maximum value and then slowly went down. The findings from the line charts are good supplements to conclusions gained from the scatterplot display.

Figure 4.9(b), shows that this interesting change exists in the recent several windows. If users want to see more details of this change, the DOI function interactive tool can be used to adjust the DOI function in order to choose fewer time windows to display. Followed by this idea, the new DOI function shown in Figure 4.9(c) results in the new view shown in Figure 4.9(d). This new view clearly demonstrates how the positions of clusters change as compared to Figure 4.9(b).

Case Study 3:

This case uses a 5-minute slice of the sleep data stream. I split it into 10 time windows and generated Figure 4.17 using step juxtaposition and parallel coordinates. Let us first observe the changes of the correlation between the heart rate and blood oxygen concentration. Basically, Figure 4.17 shows two types of distribution: Type 1: low heart

rate and high blood oxygen concentration, such as in windows 54.0-54.5, 56.5-57.0, and 58.0-58.5; and Type 2: high heart rate and low blood oxygen concentration, e.g., windows 54.5-55.0 and 57.0-57.5. The datapoints in some windows are the mixture of two types of data, such as window 57.5-58.0. Each sub-figure can clearly show how the data is changing from one type to the other. For example, in Figure 4.17(a), the data changed from type 1 to type 2. After investigating the whole stream, I found that type 1 is the primary one while type 2 concentrates in some portions of the stream. Thus type 2 can be considered to be an outlier. Since the patient in this sleeping experiment shows sleep apnea (periods during which he takes a few quick breaths and then stops breathing for up to 45 seconds), type 2 data might be associated with this abnormality. The third dimension, the chest volume, that is the indicator of respiration, can tell us whether the hypothesis is accurate or not. During normal human breathing, the chest volume should change in an oscillating way. We can see that in most time windows, the chest volume values exist in a wide range, which is normal. However, the values of this dimension in four windows have a very narrow range, including windows 54.0-54.5, 55.5-56.0, 56.5-57.0, and 58.0-58.5, where the patient stopped breathing for a while. Moreover, just after each of these four windows, data changed from type 1 to type 2. For example, in Figure 4.17(c), the darker points correspond to window 55.5-56.0, where the patient might stop breathing. In addition, the data changed from type 1 to type 2 in Figure 4.17(d). Thus, a finding, with the help of visualizations, is that the patient will change to an abnormal condition of high heart rate and low blood oxygen concentration after the sleep apnea happened. This finding has not been confirmed with the medical expert, but this case study at least can show that the proposed layout strategies are useful in helping us find possible cause and effect in data streams, when data analysts apply them to traditional multivariate visualizations. This will help analysts promote hypotheses or confirm new findings.

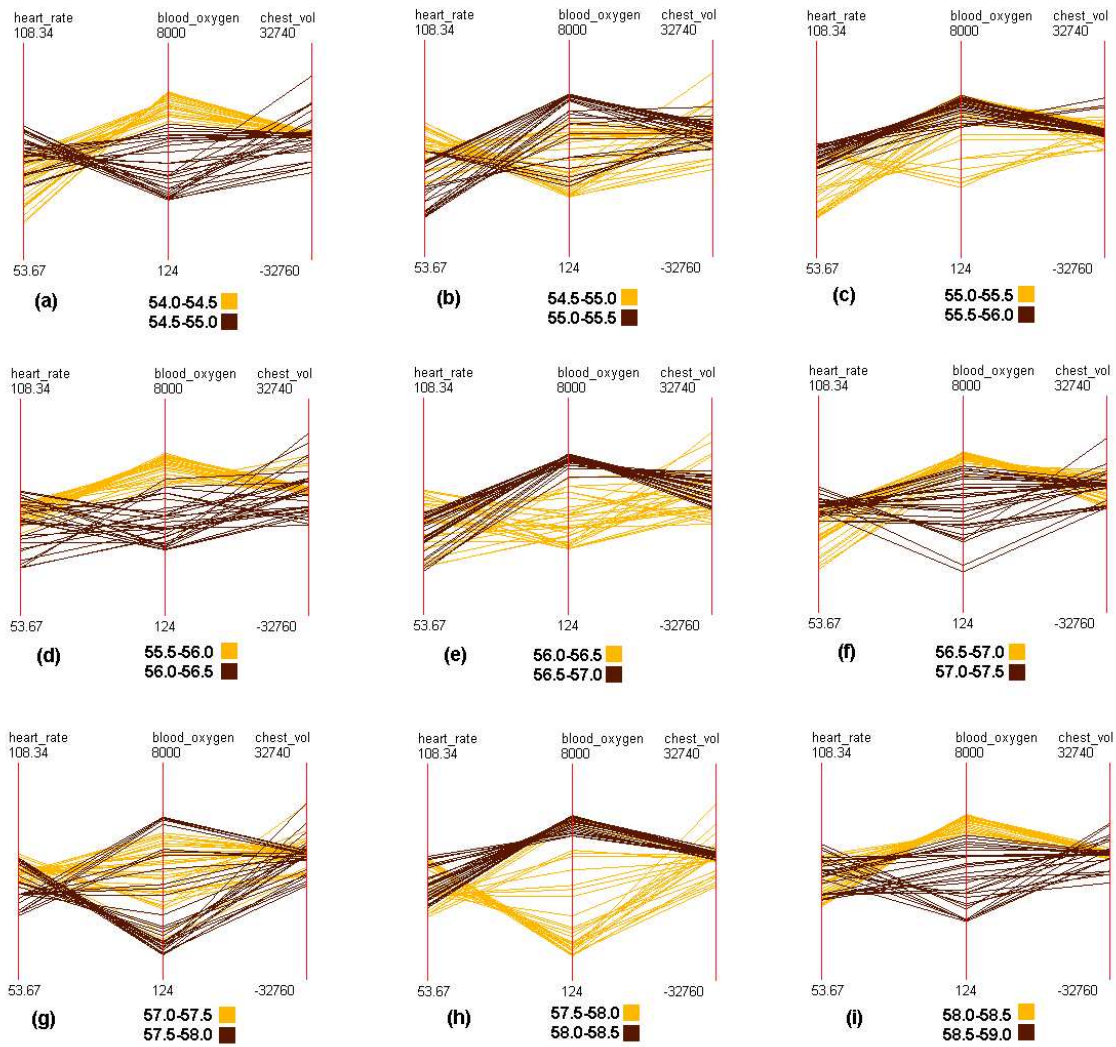


Figure 4.17: The visualization for a slice of sleep data stream generated by applying step juxtaposition to parallel coordinates. All time units in the figure are minutes. We can clearly see how the relationship between variables changes from one window to the next.

Chapter 5

A Data-driven Approach to Merging Windows

5.1 The Data-driven Framework

Before describing the framework and algorithms, I will give some terms and definitions.

Terms:

If windows $W_1, W_2, \dots,$ and W_k are merged to a window W' , W' is called the **parent (window)** of $W_1, W_2, \dots,$ and W_k , and $W_1, W_2, \dots,$ and W_k are defined as the **child (windows)** of W' . In other words, $W_1, W_2, \dots,$ and W_k are **original windows** and W' is the **merged window**. Many multivariate data patterns can be described using a vector (v_1, v_2, \dots, v_r) that is a **pattern vector**. An example is the vector $(-2, 5)$ that describes a linear trend $y = -2x + 5$.

Definitions:

W An original or merged window.

n_m The number of merged windows visualized.

n_o The number of original windows, including all child windows of all merged windows visualized.

V_p The pattern vector to describe the data pattern retrieved from a merged or original window.

$F_p(W)$ The pattern retrieving function to compute a pattern vector V_p from the data-points in a window W .

$d(V_{p_1}, V_{p_2})$ The function to measure the distance between two pattern vectors. For one specific pattern vector definition, this might not be unique, i.e., to reflect different users' interests.

$G_m(V_{p_1}, V_{p_2}, \dots, V_{p_k})$ The merge function to generate the pattern vector of a parent window from its child windows W_1, W_2, \dots, W_k that have pattern vectors $V_{p_1}, V_{p_2}, \dots, V_{p_k}$.

For the two data patterns investigated in this dissertation, linear trends and data range, I provide details about their pattern vectors V_p , and three functions, $d(V_{p_1}, V_{p_2})$, F_p , and G_m in Appendix .1 and .2.

Figure 5.1 shows the framework to generating the pattern and pattern change visualizations for the current view and history data. Here non-overlapped time windows are used. The time window is the minimal unit that users want to use to study data patterns. The current view contains n_o contiguous time windows, including the most recent time window. n_o is normally determined by users. n_o could be bigger than the maximal number of windows that the canvas can hold, which is defined as N_m ($=3$ in Figure 1.4). In other words, we allow at most N_m time windows to be displayed in the final output. The merge algorithm can reduce the number of windows in the current view to $n_m (< N_m)$ via merging adjacent windows having only small changes. If one window in the current view

becomes expired, it is sent to the history data pool. When the size of this pool exceeds the memory limit, a procedure is triggered to compress the history data via merging window pairs having small or no pattern changes. The pattern retrieval algorithm calculates the data patterns and pattern changes for the windows in both the current view and the history data pool. Finally, three types of views, juxtaposed views, pattern views and change views, are generated. The first type generates multiple traditional multivariate visualizations for each merged window, and juxtaposes them on the canvas. This can preserve all the details in the data. However, it needs a lot of canvas space and is not applicable to history data. The last two types regard the pattern vectors and their changes as time-series data, and generate traditional time-series visualizations to convey the data pattern change.

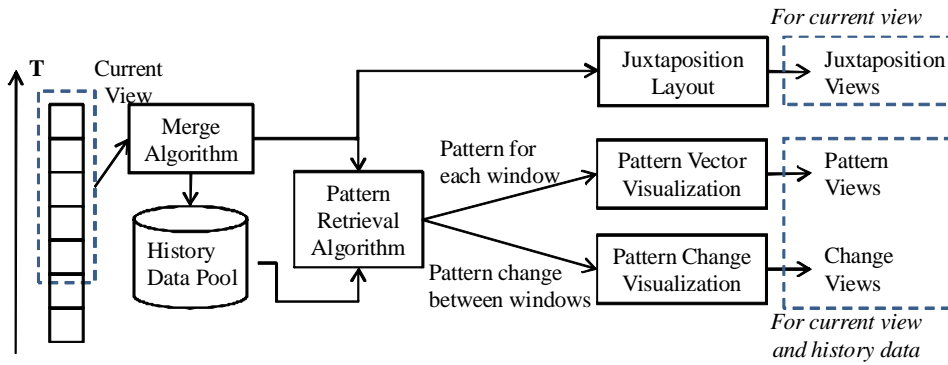


Figure 5.1: The framework to show how the windows are merged, history data is stored, and patterns or pattern changes are retrieved and visualized.

5.2 Merge Algorithm

As mentioned in the previous section, the basic idea of the proposed approach is to merge n_0 windows to $n_m (\leq N_m)$ windows, where N_m is the maximal number of windows that the canvas can hold. Basically, this is a problem of data compression. First, I will discuss some theory and algorithms for compressing data, and introduce a quality measure to describe how well a merge algorithm performs. I then propose two algorithms: *brute*

force and *heuristic*, and finally extend them to data streams.

5.2.1 The State of The Art in Data Compression

Data compression is based on information theory in which the primary goal is to minimize the amount of data to be transmitted [39, 54]. The basic method is to reduce the redundancy, leaving only the informational content. One of classical data compression algorithms is Huffman coding [30]. It assigns short codewords to those messages appearing frequently. Another compression technique for image compression is *difference mapping* [37] that is worthwhile mentioning. It represents an image as an array of differences between adjacent pixels rather than the pixel color coding. Since the adjacent pixels normally are similar, this technique can achieve a good compression ratio. I was inspired by this algorithm since my goal is to represent the difference between adjacent windows.

Data compression techniques can be categorized into lossless and lossy algorithms. The former technique means that we can get back all information after we decompress the compressed data, while the latter will discard some information. For example, Huffman coding and difference mapping both belong to lossless compression. Lossy compression is commonly used to compress audio, video or images, because the tiny difference in these areas is normally acceptable [17]. JPEG is a commonly used lossy compression technique for digital images. The name “JPEG” stands for Joint Photographic Experts Group, the name of the committee that created the JPEG standard and also other standards. In this technique, the images first are converted from RGB to YCbCr color space (Y: brightness, Cb: blue component, Cr: red component). Three components will be sent to separate channels to be compressed to achieve more efficient compression. Each channel is split into 8×8 blocks. A discrete cosine transform (DCT) is applied to each block. DCT is a Fourier-related transform that can separate the high-frequency and low-

frequency components. Human eyes are good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. So JPEG algorithm reduces the amount of information in the high frequency components after DCT transformation. Finally, the reduced information is quantized. For video compression, existing algorithms normally make use of the similarities between adjacent frames. In MPEG-I, pictures are categorized into four types: (a) I pictures, which are coded via the JPEG technique; (b) P pictures, which can be predicted from previous I or P pictures; (c) B (bidirectional) pictures, which are predicted from both past and/or future I or P pictures (for these data, reordering may be necessary); and (d) D pictures, which allow fast-forward mode with restricted quality.

Wavelets are another important technique for lossy compression. A wavelet is a kind of mathematical function used to divide a given function or continuous-time signal into different frequency components and study each component with a resolution that matches its scale. Its main advantages are the low time cost, multi-resolution features, and scalability [43]. Some visualization researchers used wavelets to visualize large-scale multivariate data at multiple resolutions [47, 51, 65]. For example, Miller et al. [47] applied wavelet transformations to the digital signal constructed from words within a document, and then used wavelet energy to analyze the thematic characteristics at varying degrees of detail, ranging from sections to words. Based on the analysis results, a visualization system, named TOPIC ISLANDS, was created to provide fuzzy document outlines at different levels of detail. Wong and Bergeron utilized wavelet transformations to display brushed data at a different resolution than the non-brushed data [65].

5.2.2 The Quality Measure for The Merge Algorithm

The basic idea for measuring the merge result quality is to compute how much the change information is preserved by the merge algorithm. To explain this, I first introduce an

intuitive merge algorithm, named *pattern-blind averaging*, which merges every n_0/N_m original time windows to one merged window. The pattern vector of the merged window is the average value of the pattern vectors of the original n_0/N_m time windows. Pattern-blind averaging is easy to understand and will be the competitor of the *brute force* and *heuristic merge* algorithms proposed later. Figure 5.2 shows the merge result of pattern-blind averaging and heuristic merge for the same input. Without loss of generality, in this example, the pattern of a time window is described by a real number, namely *pattern number*, and the change from one window to the next is defined by the difference between two numbers. One assumption in this example is that the pattern number of a merged window is the weighted average value of pattern numbers of all corresponding original windows computed via:

$$\bar{v} = \frac{v_1 n_1 + v_2 n_2}{n_1 + n_2}$$

Note that $v_1(v_2)$ and $n_1(n_2)$ are the pattern number and the number of datapoints for window $W_1(W_2)$.

Although a pattern vector could contain many scalar values in real applications, the distance between two pattern vectors are always represented by a real number $d(V_{p_1}, V_{p_2})$ (defined in Section 5.1). In the following definitions, I use only the distance measures between two time windows instead of pattern vectors themselves. Simplifying the pattern vector to a numerical value does not impact the discussion about the quality measures of the merge algorithm.

Figure 5.2 contains two subfigures. In each subfigure, the first row represents the original windows; the merged windows are shown in the second row. The second row of the left part shows the result of a heuristic merge algorithm that will be discussed in detail in Section 5.2.4. The right part is from an intuitive method, named pattern-blind averaging, which merges every n_0/N_m original time windows to one merged window.

In Figure 5.2, the heuristic algorithm merged W_1, W_2, W_3 and W_4 to one window.

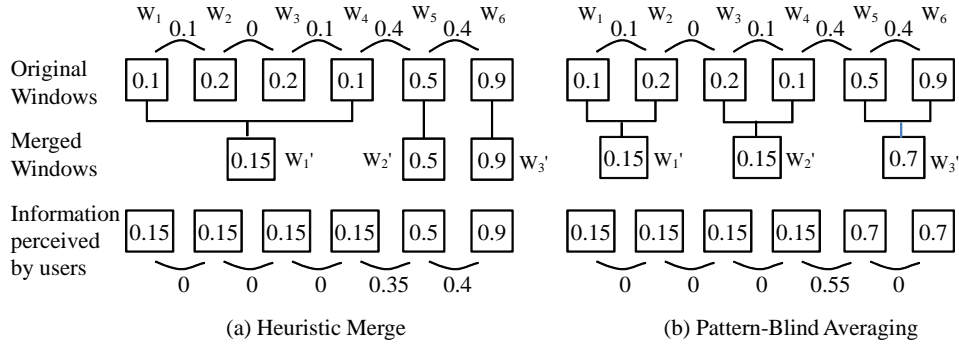


Figure 5.2: This figure shows how to measure result quality of the heuristic merge algorithm and pattern-blind averaging regarding the degree to which the change magnitude is preserved.

Therefore, from the user’s perspective, the pattern numbers of these four windows will be the same(0.15), although their actual pattern numbers are 0.1, 0.2, 0.2 and 0.1. This loss is shown in the third row of this figure. Hence, the distance between W_1 and W_2 will be 0 if try to retrieve it from the final visualization, while the actual change from W_1 to W_2 is 0.1. That is to say, the change information from W_1 to W_2 have been totally lost. Similarly, the original distance between W_4 and W_5 is 0.4, but this will change to 0.35 in the final visualization.

If d_i is used to represent the actual distance between original windows W_i and W_{i+1} , and d'_i is the perceived distance from the visualization after applying the merge algorithm to the data, then

$$D = \sum_{i=1}^{n-1} |d_i - d'_i| \tag{5.1}$$

is the total deviation for the change information after time windows are merged. Note that n is the number of original windows.

Obviously, a better merge algorithm should have a smaller deviation D . So Equation 5.1 is extended to

$$Q = \frac{1}{n-1} \sum_{i=1}^{n-1} \left(1 - \frac{|d_i - d'_i|}{d_{max}} \right) \tag{5.2}$$

to represent the merge result quality. Note that d_{max} is the maximal change. The above

definition guarantees that $0 \leq Q \leq 1$. A bigger Q value means a higher quality. If $d_{max} = 1.0$, then the quality measures for Figures 5.2(a) and 5.2(b) are 0.95 and 0.85 based on Equation 5.2, respectively.

In addition, normally people are more interested in bigger changes than small ones, so only those changes bigger than a threshold are counted, and Equation 5.2 is extended to

$$Q = \frac{1}{n' - 1} \sum_{d_i \geq d_T} \left(1 - \frac{|d_i - d'_i|}{d_{max}} \right) \quad (5.3)$$

where d_T is a distance threshold, and n' is the number of items in the set $\{d_i | d_i \geq d_T\}$. If $d_T = 0.2$, based on Equation 5.3, the quality measures for Figures 5.2(a) and 5.2(b) become 0.975 and 0.725, respectively .

5.2.3 Brute Force

The basic idea to brute force merge is intuitive. If the goal is to merge n_0 to N_m windows, the brute force algorithm will iterate all possible ways to split n_0 original windows to N_m subsets. Note that, in each subset, all windows should be contiguous. For each subset, the quality measure is calculated via Equation 5.2 or 5.3, and the result is that one having the best quality. Figure 5.3 shows a result using brute force algorithm ($n_0 = 12, N_m = 4$). Its quality measure is 0.85 from Equation 5.2 ($d_{max} = 1.0$);

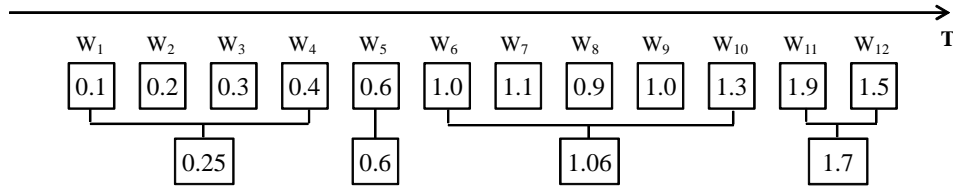


Figure 5.3: An example to show the result of brute force merge where 12 windows need to be merged to 4 windows. Its quality measure is 0.85 (Eq.5.2, $d_{max} = 1.0$).

To analyze the time complexity of this algorithm, I will first show a theorem:

Theorem. *There are $\binom{n_0-1}{N_m-1}$ ways to merge n_0 original windows to N_m merged windows.*

Proof. Imagine that we have $N_m - 1$ poles and put each of them between two contiguous windows, and two poles cannot be in the same position, then these $N_m - 1$ poles can split $n_0 - 1$ original windows to N_m subsets. Thus we can get N_m merged windows by merging all windows in each subset. Because there are in total $n_0 - 1$ positions where we can put poles, we can find $\binom{n_0-1}{N_m-1}$ ways to merge windows. \square

The process to find the best quality measure is as follows:

1. Compute the quality measure for $\binom{n_0-1}{N_m-1}$ combinations. For each,
 - 1.1 Use the merge function G_m to calculate the pattern vectors of N_m merged windows.
 - 1.2 Obtain the quality measure via Eq. 5.2.
2. Find the combination having the largest quality measures

For each combination, the time cost of Operations 1.1 and 1.2 is $C_1 n_0$ and $C_2 N_m$; in addition, the operation 2 takes $C_3 N_m$ (C_1 , C_2 and C_3 are constant real numbers). So, the whole algorithm needs to run:

$$\binom{n_0 - 1}{N_m - 1} (C_1 n_0 + C_2 N_m) + C_3 N_m \quad (5.4)$$

Normally, N_m is a constant in many applications, thus

$$\binom{n_0 - 1}{N_m - 1} = O(n_0^{N_m-1})$$

Therefore, the time cost of the brute force merge is $O(n_0^{N_m})$.

5.2.4 Heuristic Merge

The brute force merge is easy to implement but is time consuming. It is not acceptable for some real applications that need quick response. I now propose a heuristic algorithm that is much faster although its result might have a lower quality measure. The basic idea is to repeatedly scan the window list multiple times and merge contiguous windows having a change smaller than a threshold value, until the number of windows is smaller or equal to N_m .

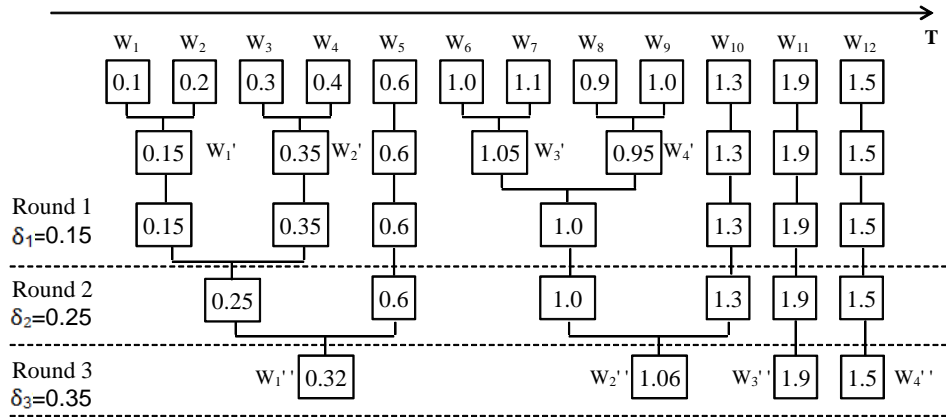


Figure 5.4: An example to show how to do multiple pass heuristic merge for the same input as Figure 5.3. The distance threshold sequence I used is $\{\delta_1 = 0.15, \delta_2 = 0.25$ and $\delta_3 = 0.35\}$. In each pass, The scan on the window list might be multiple times until all changes between contiguous windows are bigger than δ_i . The quality measure of the merge result is 0.838 (Eq.5.2, $d_{max} = 1.0$).

Figure 5.4 shows how to merge 12 windows to 4 windows using the heuristics merge. This example uses the same input as the brute force merge in Figure 5.3. The whole procedure is composed of three rounds. Each round i uses a different pattern distance threshold δ_i . The goal of this round is to merge as many windows as possible until the smallest pattern distance between contiguous windows is bigger than δ_i . Hence, two problems need to be solved: (1) how to choose δ_i ; and (2) how to merge.

In order to choose an appropriate δ_i , I first scan the whole window list, and then get the minimal pattern change between contiguous windows Δ_{min} , which is the minimal

values in $\{\Delta_j(1 < j < n')\}$. Note that n' is the current length of the window list. Then I choose $\delta = f(\Delta_{min})$ as the input of the one pass merge algorithm. The definition of function f depends on the application. In this example, $\delta = \Delta_{min} + 0.05$.

One may argue that $\delta = \Delta_{min}$ can be used to set the threshold in the merge algorithm. The reason why I do not set $\delta = \Delta_{min}$ is that probably only two windows will be merged in one round in that way. That could cause too many rounds in the merged-based hierarchical structure and high time costs.

In the first round, $\Delta_{min} = 0.1$, so $\delta_1 = 0.15$. The pattern distance measures between W_1 and W_2 , W_2 and W_3 , W_3 and W_4 , all are 0.1. Is it feasible to merge W_1 , W_2 , W_3 and W_4 to one window? Absolutely not. The reason is that the data pattern is increasing steadily from W_1 to W_4 . The aggregate change is 0.3, which is not small. If these four windows are merged to one window, this important change will be totally lost. Therefore, the heuristic merge algorithm only merges two windows at once. To explain this idea, assume the original window list is $\{W_1, W_2, \dots, W_{n_0}\}$, and the change threshold is δ . The algorithm will search the whole window list from the beginning, until finding two contiguous windows, say W_j and W_{j+1} having a change less than or equal to δ , and then merge them. After that, the same searching and merging will be repeated from W_{j+2} until W_{n_0} . For example, in the first scan, this algorithm only merges W_1 and W_2 , W_3 and W_4 , W_6 and W_7 , W_8 and W_9 , and keeps other windows unchanged. Note that a single scan is not enough because the change between a merged window and an original window, or two merged windows, can be less than or equal to the current δ , e.g., $W'_3(1.05)$ and $W'_4(0.95)$. Thus multiple scans are necessary for the window list. Actually, this algorithm did the full scan twice in Round 1. After Round 3, the length of the window list is 4, which is the goal, so the heuristic merge can stop.

Algorithm 1 shows the pseudocode to do one merge round given a pattern distance threshold δ . In Algorithm 2, multiple rounds are done by calling Algorithm 1.

Algorithm 1 Merge windows with pattern distance $\leq \delta$.

```
1: Input:  $\{W_i\}$  (The window list) ;  $\delta$  ( pattern distance threshold).
2: found  $\leftarrow$  false;
3: repeat
4:    $S \leftarrow \emptyset$  {S is the set to store merged windows.}
5:   while  $j < \text{Len}(\{W_i\})-1$  do
6:     while  $j < \text{Len}(\{W_i\}) - 1$  and  $d(W_j, W_{j+1}) > \delta$  do
7:        $j \leftarrow j+1$ ;
8:     end while
9:     if  $j < \text{Len}(\{W_i\})-1$  then
10:       $S \leftarrow S \cup \text{Merge}(W_j, W_{j+1})$ ; {Merge() merges two windows and returns the
      result.}
11:      found  $\leftarrow$  true;
12:       $j \leftarrow j+2$ ;
13:    end if
14:  end while
15:  Update the window list via replacing windows with their parent windows in  $S$  if
  they exist.
16: until found = false
```

Algorithm 2 Merge n_0 windows to $n_m (\leq N_m)$ windows.

```
1: Input:  $\{W_i\} = \{W_1, W_2, \dots, W_{n_0}\}$  (The original window list) ;
2:  $k \leftarrow 0$ ;
3: while  $\text{Len}(\{W_i\}) > N_m$  do
4:   { Len() returns the length of the window list.}
5:    $\Delta_{min} = \min(\{d(W_i, W_{i+1}) | 1 \leq i \leq n' - 1\})$ ,  $n'$  is the length of current window
   list;
6:    $\delta \leftarrow f(\Delta_{min})$ ;
7:   Call Algorithm 1 using  $\delta_k$ ;
8:    $k \leftarrow k + 1$ ;
9: end while
```

The heuristic merge might generate a result having a lower quality measure than the brute force method. For example, the result quality in Figure 5.4 is 0.838. Although it is a little lower than 0.85 generated by brute force merge (Figure 5.3), it is much higher than the result quality 0.730 generated by pattern-blind averaging (Figure 5.5).

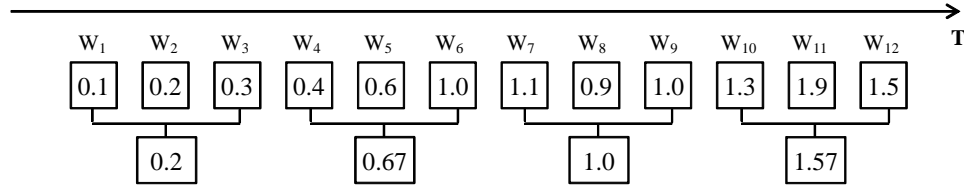


Figure 5.5: Pattern-blind averaging is applied to merge time windows for the same input as Figure 5.3. Its quality measure is 0.730 (Eq.5.2, $d_{max} = 1.0$), much lower than brute force and heuristic merge.

In the heuristic merge, every time two windows are merged, the length of the window list will be decreased by 1, so the time cost for merging windows is $C_1(n_0 - N_m)$. In addition, it takes time to find a smallest distance in each full scan on the window list. For the worst case, this algorithm needs to do $n_0 - N_m$ scans. so the time cost for this operation is $C_3(n_0 - N_m)^2$. Thus the total time cost is:

$$C_1(n_0 - N_m) + C_3(n_0 - N_m)^2 = O(n_0^2) \quad (5.5)$$

Note that the constants C_1 and C_3 have the same meaning as Eq. 5.4 in Section 5.2.3 used to compute the time cost for brute force merge. Compared to brute force merge, the time cost have been decreased from $O(n_0^{N_m})$ to $O(n_0^2)$ with a loss of result quality. It is a big savings when $N_m > 2$.

Section 5.4 will describe an experiment to compare the result quality of the two merge algorithms, and show that the loss in quality is worthwhile compared to the savings in computation cost.

5.2.5 Stream-based Merge

For a data stream, if one new window arrives, the oldest window, namely the expired window, has to be removed from the current view before the new window is added into the visualization (Figure 5.1). For example, in Figure 5.4, if W_{13} comes as a new window, then the current view becomes $\{W_2, W_3, \dots, W_{12}, W_{13}\}$, and W_1 is expired. Obviously, a re-merge is needed. The easiest approach to re-merge is to run the brute force or heuristic merge again on this new window list $\{W_2, W_3, \dots, W_{12}, W_{13}\}$. This is not efficient because the existing merge result for $\{W_1, W_2, \dots, W_{11}, W_{12}\}$ is not reused. To avoid this disadvantage and save the time cost for merging, the newly arrived window is handled by the following steps: (1) If the expired window has been merged into other windows, decompose the oldest merged window and put all its child windows back on the window list. (2) Remove the oldest window from the window list. (3) Add the new window to the window list. (4) Run the brute force or heuristic merge on the new window list. Therefore, after the new window W_{13} arrives, merge algorithm is run on $\{W_2, W_3, W_4, W_5, W_2'', W_3'', W_4'', W_{13}\}$ instead of $\{W_2, W_3, \dots, W_{12}, W_{13}\}$. Figure 5.6 shows the details of how to decompose W_1'' and do the re-merge starting from the existing result. The result is the same as what is obtained by doing heuristic merge directly on the original window list, but is obtained by running the merge algorithm starting from 8 windows instead of 12 windows.

Recall that the time cost of the brute force and heuristic merge is $O(n_0^{N_m})$ and $O(n_0^2)$ respectively, where n_0 is the number of original windows. For every n_0 input windows, on average, n_0/N_m original windows are merged to one merged window, so this stream-based merge algorithm needs to merge only $\frac{n_0}{N_m} + N_m$ windows. Normally, N_m is a small constant number, so, stream-based merge can reduce the time cost of brute force and heuristic merges by $\frac{1}{N_m^{N_m}}$ and $\frac{1}{N_m^2}$ respectively.

The above estimation does not consider the time cost increase to detach the expired

windows from the existing merge result. In Section 5.4, I describe experiments to investigate whether and how this optimization affects the result quality and time cost.

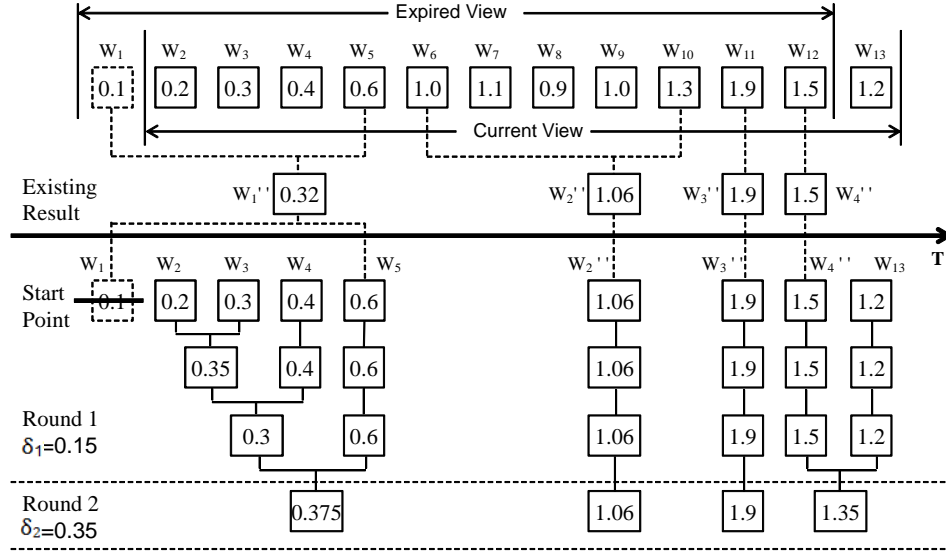


Figure 5.6: This figure shows how to do a stream-based heuristic merge when a new window arrives after 12 windows are merged to 4 windows in Figure 5.4. W_1'' is decomposed to make full use of existing result. In this way, the merge starts from 8 windows instead of 12 ones.

How to Merge Windows: There are two options to merge two windows: (1) doing a union set operation on two windows and then doing sampling to reduce the number of datapoints to the size of one window; or (2) utilizing the merge function G_m to calculate the pattern vector of the parent window. In the current view stage, the first approach is picked because it can save data details. For the history view and data storage, the selection depends on the users and the characteristics of the data pattern. For the data patterns having the merge function G_m , the second approach can be chosen to save memory space. If the merge function does not exist or is difficult to compute for a particular data pattern, e.g., clusters, only the first approach is viable.

Storage Policy: When one new window arrives, an old window must leave the current view. It will be stored in the history data pool for the purpose of generating history views. Because the data stream is potentially infinite in nature, all windows cannot be

stored. Even if only pattern vectors for time windows are stored, the memory will still be eventually full. To solve this problem, the merge approach is used: merging those windows with small changes to save memory space. In addition, old windows are merged earlier than newer windows because more recent data is more important than older data for most data analysis tasks. For example, if users want to know whether today's traffic is normal, they normally need to compare it with yesterday or last week, and rarely with last month or last year.

Specifically, users are allowed to provide two sequences $\{T_i\}_{i=0}^q$ ($T_0 < T_1 < \dots < T_q$, $T_0 = 0$, $T_q = \infty$), and $\{\delta_i\}_{i=0}^p$ ($\delta_0 < \delta_1 < \dots < \delta_p$), where T_i denotes a data age (the difference between the timestamp of this datapoint and the current time), and δ_i is a pattern change threshold. Note that δ_p is the maximal possible change of the data pattern. The sequence $\{T_i\}_{i=0}^q$ divides all the arriving windows into q sections, $[T_0(= 0), T_1]$, $[T_1, T_2]$, \dots , and $[T_{q-1}, T_q(= \infty)]$, in the order of the degree of users' interests from high to low. When the memory is full, the following procedure is triggered to merge windows in the history data pool. The goal is to reduce the history data pool size to a predefined size S_M , e.g. $0.9M$, where M is the maximal memory size assigned to the data pool. This process is done via calling Algorithm 1 up to $(p+1)q$ times. Each call is represented by $F(\delta_i, [T_j, T_{j+1}))$, which means merging all window pairs with change equal to or less than δ_i in the interval $[T_j, T_{j+1})$. All calls are placed in the following order:

$$\begin{aligned}
&F(\delta_0, [T_{q-1}, T_q)), F(\delta_0, [T_{q-2}, T_{q-1})), \dots, F(\delta_0, [T_0, T_1)), \\
&F(\delta_1, [T_{q-1}, T_q)), F(\delta_1, [T_{q-2}, T_{q-1})), \dots, F(\delta_1, [T_0, T_1)), \\
&\dots \\
&F(\delta_p, [T_{q-1}, T_q)), F(\delta_p, [T_{q-2}, T_{q-1})), \dots, F(\delta_p, [T_0, T_1)).
\end{aligned}$$

This order ensures that more important data is kept, which contains the most recent data and window pairs having significant pattern changes. After each call $F(\delta_i, [T_j, T_{j+1}))$, the data pool size is evaluated. If it is smaller than S_M , the merge process is stopped.

Otherwise, go to the next call. Since δ_p is the largest possible change and $T_0 = 0$, the memory space held by the data pool definitely will shrink to less than S_M after one call. The pseudocode of this procedure is shown in Algorithm 3.

Note that it is better to run Algorithm 3 offline than in parallel with online merging, because the latter way will increase the system complexity and causes more processing overhead.

Algorithm 3 Shrink history data pool.

```

1: Input:  $\{T_i\}_{i=0}^q$  (The data age sequence which satisfies  $T_0 < T_1 < \dots < T_q$ ,  $T_0 = 0$ , and  $T_q = \infty$ );  $\{\delta_i\}_{i=0}^p$  ( a stepped
change magnitude sequence which satisfies  $\delta_0 < \delta_1 < \dots < \delta_p$ );  $S_M$  ( The expected size held by history data pool after this
algorithm)
2: for  $i = 0$  to  $p$  do do
3:   for  $j = q-1$  downto  $0$  do do
4:      $\{W\} \leftarrow$  all windows within section  $[T_j, T_{j+1})$ .
5:     Call Algorithm 1 with parameters  $\delta_i$  and  $\{W\}$ ;
6:     if  $\text{DataPoolSize}() \leq S_M$  then
7:       Exit;
8:     end if
9:   end for
10: end for

```

5.3 Visualization of Patterns and Their Changes

This section will discuss three visualizations used in the data-driven approach: juxtaposed views, pattern vector views and pattern change views. Two particular data patterns, linear trends and data range, are chosen to show as examples.

5.3.1 Juxtaposed Views

This section presents visualization techniques based on *step juxtaposition* described in Section 4.3. Other techniques in Section 4.3 can also be applied to the techniques in the data-driven juxtaposed views.

In juxtaposed views, two types of visualization techniques are developed: (1) *juxtaposed full view* that uses traditional visualization techniques to show all datapoints in the

windows (Figure 1.4); and (2) *juxtaposed pattern outline view* that shows only the outline of the discovered pattern for each window. The pattern outline view is specific to each pattern. For example, it can be a line for linear trends.

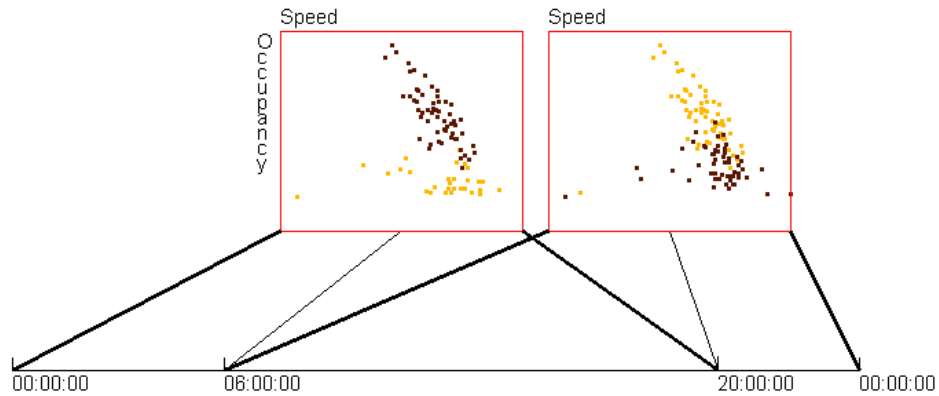


Figure 5.7: 48 windows, containing the traffic data in one day, are merged to 3 windows and then shown with 2 scatterplots. Each scatterplot contains two windows, and is linked to the time axis via three lines to delimit the time range for these two windows.

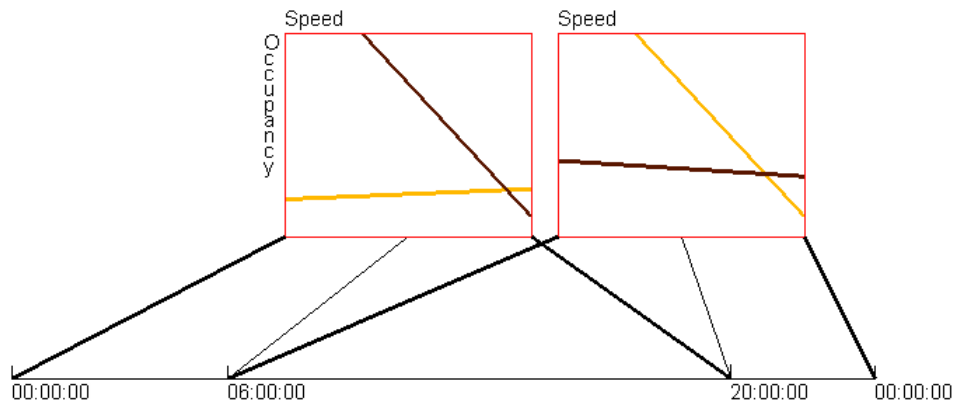


Figure 5.8: A pattern outline view to visualize the pattern change in traffic data slice used in Figure 5.7. Each line represents a linear model for a merged window. Note that three lines connecting each scatterplot to the time axis mark two corresponding time windows, which is similar to Figure 5.7.

Figure 5.7 shows a juxtaposed full view after merging 48 windows (traffic data in one day) to 3. Figure 5.8 uses the same dataset and merge algorithm as Figure 5.7 but contains the pattern outline view.

In Figures 5.7 and 5.8, all subfigures are placed on the canvas horizontally in the order of the timestamp. Because the time axis is evenly spaced and subfigures have different lengths of time range, Lines are used to connect subfigures to the time axis. This can help users understand where the change is fast and where the change is slow. I call this a *1D layout*.

The 1D layout is intuitive to interpret, but it does not make full use of the canvas when the number of merged windows is large, especially for those visualization techniques that generate output in a shape close to square, such as scatterplots or parallel coordinates. In order to avoid this drawback, a grid layout is proposed, in which all subfigures are laid out in a grid having n rows and n columns. If there are m subfigures, $n = \lfloor \sqrt{m-1} \rfloor + 1$. In grid views, the representation of the time axis is problematic. If the same method as the 1D layout is used to connect the subfigures to the time axis via lines, a lot of overlapping will occur. I solve this problem using an interaction technique: when the mouse hovers over a subfigure, the corresponding time range is highlighted on the time axis (Figure 5.9).

Figure 5.9 shows an example using the pattern outline view and grid layout. Each subfigure is a two-dimensional parallel coordinates. There are two bands in each subfigure. One band represents the data range in a time window. On dimension X , two corners of the rectangles correspond to $(\bar{X} + s)$ and $(\bar{X} - s)$ respectively, where \bar{X} and s represents the average value and standard deviation of all values within the corresponding time window. There are two types of range in this figure: Type 1 (low heart rate and high blood oxygen concentration, e.g., the yellow band in the highlighted subfigure) and Type 2 (high heart rate and low blood oxygen concentration, e.g., the dark band in the highlighted subfigure). The merge algorithm can automatically detect the shift between two types, as shown in Figure 5.9. From the time axis, one can find that Type 2 normally only exists in a short time range, so it can be treated as an outlier. This might be associated with sleep apnea,

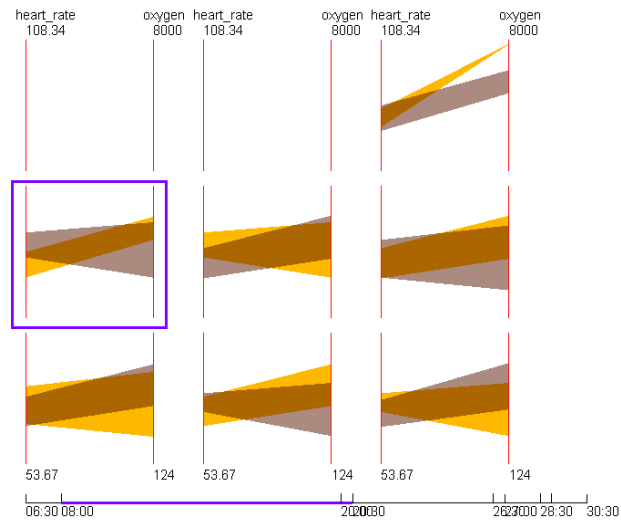


Figure 5.9: A pattern outline view in the grid layout to visualize the changes in data range for the sleep data. A subfigure is highlighted with a purple border when the mouse hovers over it. The corresponding part of the time axis is highlighted as well.

with which the subject in this sleeping experiment has been diagnosed [23].

5.3.2 Pattern Vector and Pattern Change Views

For data in the current view (see Figure 5.1), juxtaposed views can do very well in conveying the pattern change. But they perform worse for historical data. The main reason is that there will be many windows in the history. Imagine that there are 100 scatterplots on the canvas. Then each scatterplot will be very small. Even if zooming techniques and scrolled area are provided, it is still a tedious and difficult data analysis task to study how the linear model changes within these 100 windows. Therefore, *pattern vector views* and *pattern change views* are designed to visualize the history data. The basic idea is to utilize time-series visualization techniques to visualize the pattern vectors and pattern changes directly.

Pattern Vector Views: Assume that the pattern vector is an n -tuple, i.e., $V_p = (v_1, v_2, \dots, v_n)$.

Then, starting from n_m merged windows, a multivariate dataset having n_m datapoints can

be created. Each datapoint corresponds to a pattern vector of a merged window and has n columns. This is also a time-series dataset because each datapoint has a timestamp. Line charts, bar charts, heatmaps, or any other time-series visualization techniques can be used to visualize this dataset. The final output is named a *pattern vector view*.

Now, a problem similar to juxtaposed views arises: which option is better in the even and uneven time axis? Since most time-series visualization techniques can be distorted to be uneven, three approaches are proposed. To explain them better, assume that in one streaming dataset, the windows from 6AM to 9AM has been merged to 2 windows: one is from 6AM to 7AM (1 hour); the other is from 7AM to 9AM (2 hours). Bar charts are used to represent one dimension in the pattern vector.

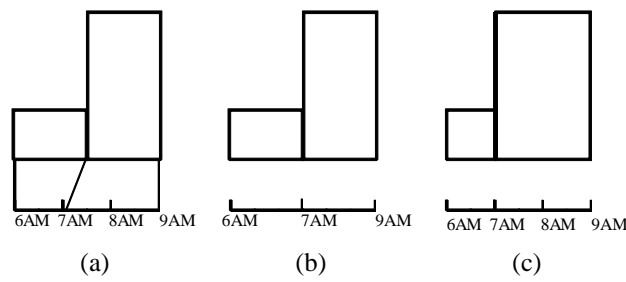


Figure 5.10: This figure shows three approaches to choosing an even or uneven time axis: (a) Even time axis, even windows; (b) Uneven time axis, even windows; (c) Even time axis, uneven windows.

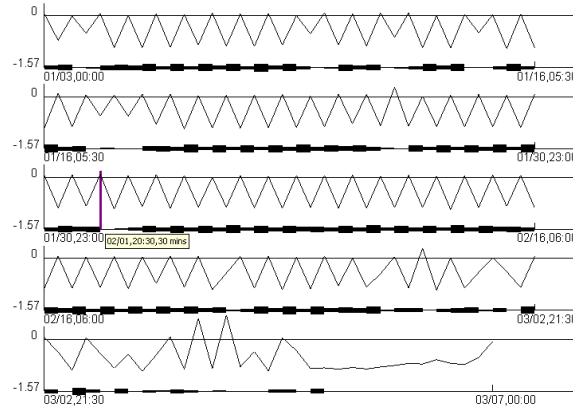
1. **Even time axis, even windows:** The visual elements (points or bars) of the time-series visualizations corresponding to each merged window have the same width, but the time axis is evenly spaced, so windows have to be connected to the time axis using straight lines (Figure 5.10(a)). This approach forces us to place the visualization on one row.
2. **Uneven time axis, even windows:** visual elements corresponding to each merged window are allowed to use the same width but the time axis is divided to n_m parts

with the same length. Each part of the time axis is just below the visual elements of the corresponding merged windows, so the time axis is not evenly spaced (Figure 5.10(b)).

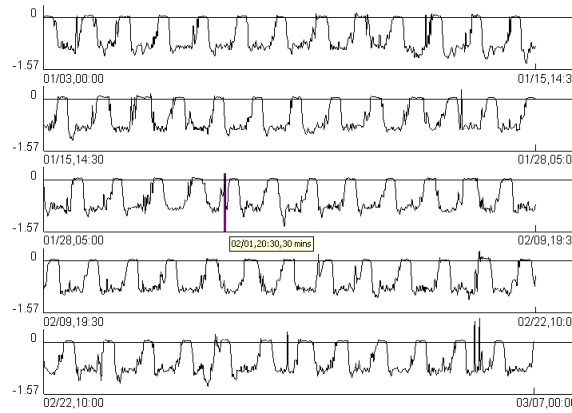
3. **Even time axis, uneven windows:** This approach uses an evenly spaced time axis, but distorts the visual elements corresponding to each merged window to force them to be just above the correct label on the time axis (Figure 5.10(c)).

In the above three approaches, the first and the second are primarily applicable to only the current view, but not for the history data. The reason is that approach 1 needs many straight lines to connect time-series visualizations with the time axis, and approach 2 has to provide the label at the border of all merged windows on the time axis. For history data potentially containing many merged windows, both these approaches will make the final visualizations too cluttered. Thus the suggestion is to use approaches 1 and 2 for the current view, but use the third approach on the historical data.

Figure 5.11(a) shows the pattern vector view via line charts for traffic data over 9 weeks (Jan. 3 - March 6, 2009). 3024 original windows are merged to 173 windows. This figure uses the even time axis and uneven windows. The incline angle $\arctan \beta$ is shown for each window in this figure, where β is the fit line slope for the linear model $Y = \alpha + \beta X$. Note that merged windows in Figure 5.11(a) normally have different time length. Labeling the time for each window is impossible because it can bring a lot of overlapping at the time axis. Instead, the thickness of the time axis segment represents the length of each merged window. A thicker time axis segment means a longer window, and thus indicates a slow pattern change, while a thin segment indicates a quick change. In order to show how the proposed techniques help users detect the pattern change, all the original windows are visualized in the historical data in Figure 5.11(b). One obvious observation is that the basic trend of incline angle is a wave style. However, sometimes there are



(a)



(b)

Figure 5.11: Pattern vector views using the traffic data over 9 weeks. The figures use line charts and only show the change of fit line slope for the linear model between *occupancy* and *speed*. The purple vertical line represents the beginning position of a window selected by users via moving the mouse to the specific place. (a) The merged windows in the historical data are used; (b) The original windows are visualized.

some quick fluctuations and vibrations, such as the place where the purple vertical line resides (20:30 at 02/01). It is difficult to perceive these quick changes because each original window is rendered in a very small region even if some pattern changes are very quick. The proposed visualization techniques based on the merge algorithm output can overcome this drawback. In Figure 5.11(a), the same window is highlighted via a purple vertical line. In this figure, changes can be easily observed because only the significant changes are shown. Therefore, the merge algorithm along with the proposed visualization techniques can pull out significant pattern changes and help users detect them.

Figure 5.11(a) has five rows. For the first four rows, each of them contains around two weeks, but the last row contains only one week. In addition, the last part of the curve is smooth, which corresponds to the last day (March 6). This conforms to the principle of the merge algorithm for history data where I want to keep more details for the recent data.

Pattern Change Views: This technique aims to enable users to quickly identify how data patterns change via conveying the distance between data patterns directly. Assume that the current view or the history data pool has n_m merged windows. The pattern vectors for them are V_1, V_2, \dots, V_{n_m} . The pattern distance function can be used to get a distance sequence $\{d_i\}_{i=1}^{n_m-1}$, where $d_i = d(V_i, V_{i+1})$. This is a univariate time-series data. Now the same method as in the pattern vector views is used to visualize this distance sequence and to generate *pattern change views*. Compared to the *pattern vector views*, this technique enables users to perceive the change magnitude more quickly, but loses the pattern information itself.

5.3.3 A Guide to Choose Visualization Techniques

I used several real streaming datasets to study the strengths and weaknesses of the above three views and concluded that:

- Pattern outline, pattern vector, and pattern change views can help users quickly perceive the target data patterns in a data stream.
- Pattern outline, pattern vector, and pattern change views show only the target data pattern. Juxtaposed full views can help convey other information.
- Given a fixed size of canvas, juxtaposed full views can hold the least number of merged windows, while pattern outline views can show more windows. Pattern vector (change) views can show the most windows.

Therefore, I provide the following guide to advise data analysts in choosing appropriate views in terms of data analysis tasks:

- If users want to study only the target data pattern and its changes, the pattern vector and change views are the best options for both the current view and historical data.
- For the current view, if users want to study other data characteristics as well as the target data pattern, the juxtaposed full views are the best option. If the application has close to real-time requirements, the pattern vector or change views are the best options. Without these requirements, users can choose any technique.
- When visualizing historical data potentially containing many merged windows, the pattern vector and change views are the best options because each time window needs the least canvas space.

5.4 Evaluation

In this section, I evaluate two important issues: (1) how well does the heuristic merge algorithm perform on reducing running time and preserving the change information for

data patterns compared to pattern-blind averaging and brute force? (2) how much can the proposed techniques reduce users' response time?

5.4.1 Comparisons among Two Merge Algorithms

To the best of my knowledge, there are no existing algorithms designed and optimized for achieving the same goal as the proposed merge algorithms. Therefore, I chose the pattern-blind averaging as the competitor in this algorithm to evaluate the output quality.

The traffic data on Sensor D191 was used in the experiments. The target data pattern is the linear trend between *Occupancy* and *Speed*. Every 30 minutes (60 datapoints) are regarded as one original time window. The pattern change of interest was the slope difference between regression lines of two contiguous windows.

In the previous discussion about time complexity of the proposed merge algorithms, the number of original windows (n_0) and the number of merged windows (N_m) are two main factors to impact the running time of the proposed algorithms. Thus two groups of experiments are run: (1) Fix n_0 and change N_m (Figures 5.12, 5.14(a), and 5.15(a)); (2) Fix N_m and change n_0 (Figure 5.13). Once the developed application based on the proposed merge algorithm finished the process on n_0 original windows, it immediately shifted the current view by one window and started the merge algorithm again. The total running time for all input data is recorded. In real applications, it is not necessary to run the merge algorithms so soon, because the system can wait for the arrival of a new time window if the processing time for n_0 windows is shorter than the length of one time window. However, this difference does not impact the comparison for the time cost of the proposed algorithms. In addition, the computation of the result quality is based on Equation 5.3 in Section 5.2.2 ($d_{max} = \pi/3, d_T = \pi/24$). It means that the maximal change is $\pi/3$ and users are only interested in slope changes bigger than $\pi/24$.

In the first group of experiments, I also ran the stream-based versions for heuristic and

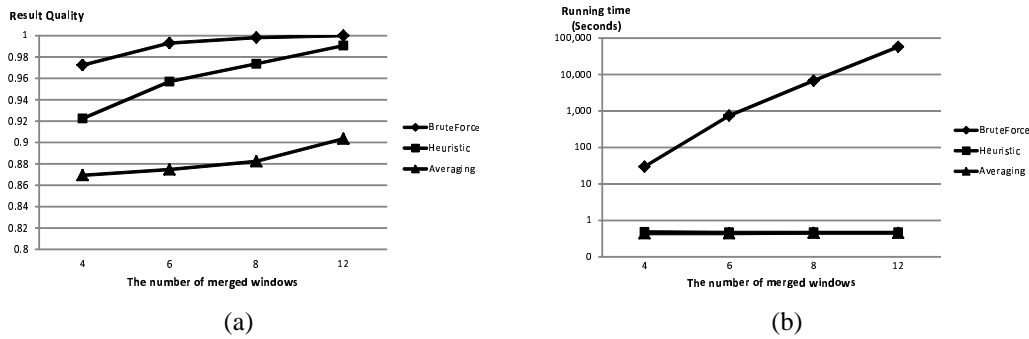


Figure 5.12: These two figures show the result of experiment 1 using the traffic data over 7 days (Jan. 1 - Jan. 7, 2008). Algorithm performance was measured when changing the number of merged windows N_m . The number of original windows n_0 in the current view is fixed at 24 windows. Note that the running time for heuristic and averaging is close to each other in Figure (b), so they overlapped a lot.

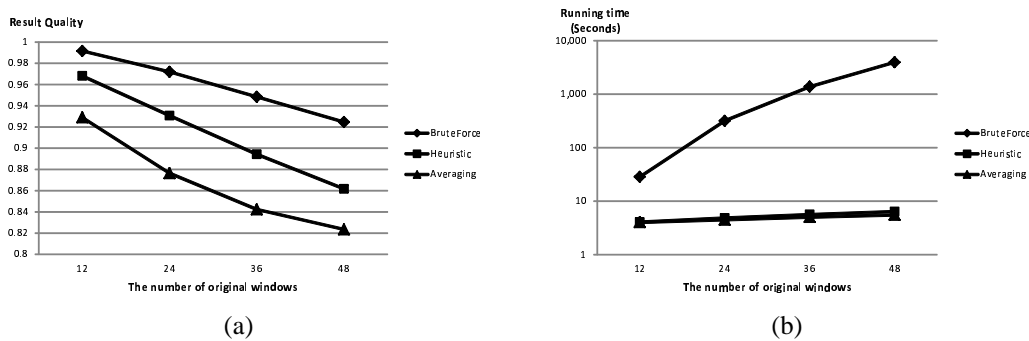


Figure 5.13: These two figures show the result of experiment 2 using the traffic data of 70 days (Jan. 1 - March 10, 2008). The algorithm performance was measured when changing the number of original windows n_0 in the current view. The number of merge windows N_m is fixed at 4.

brute force merge. This is to investigate how the stream-based optimization affects the time cost and result quality.

All experiments were run on a machine with Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz and 3.25G RAM. Its OS is Windows XP SP3.

The following observations can be made based on Figures 5.12, 5.13, 5.14, and 5.15:

- Regarding the result quality, the heuristic algorithm performs better than pattern-

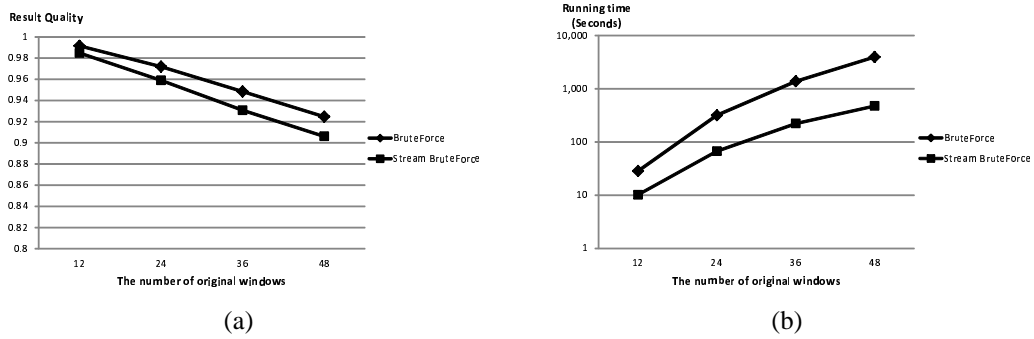


Figure 5.14: This is to compare regular brute force merge and the stream-based optimization version. The results are from the same group of experiments as Figure 5.12.

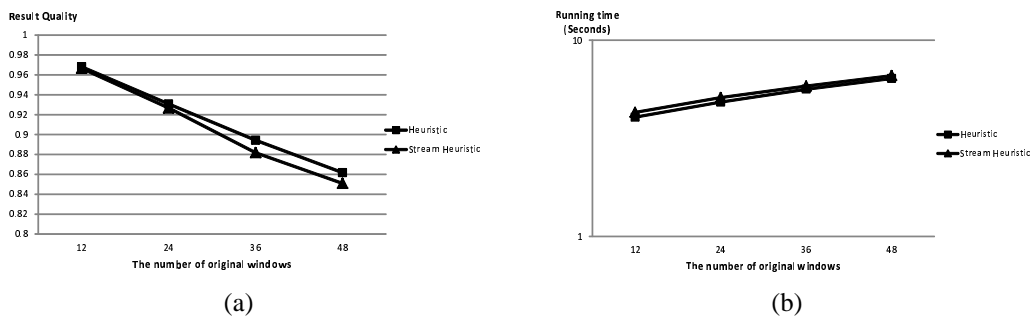


Figure 5.15: This is to compare regular heuristic merge and stream-based optimization version. The results are from the same group of experiments as Figure 5.12.

blind averaging and close to brute force, especially when the number of merged windows is big.

- The time cost of the heuristic merge is very close to pattern-blind averaging and much shorter than brute force.
- The scalability of the brute force algorithm is very bad. When n_0 or N_m become big, its running time is not acceptable.
- The stream-based optimization reduced the time cost for brute force merge by around $1/2$ to $1/8$, and does not reduce the result quality by at most 2%, which is very small.

- For heuristic merge, the stream-based optimization actually increased the time cost, probably because of the high cost to detach the expired time window.

Thus, the conclusion is that the heuristic merge is a very good improvement on brute force merge and can be applied to most cases. The brute force merge can be chosen only if n_0 and N_m are small and the experimental running time is within the real time requirement. For brute force merge, we can apply stream-based optimization to it to reduce the time cost. However, this optimization should not be applied to heuristic merge.

5.4.2 Comparing Proposed Techniques with Uniform Time Axis

One claim in Section 5.3 is that the proposed techniques can reduce users' response time for detecting pattern changes. This needs the support from an experiment. I conducted a user study to compare users' response accuracy (RA) and response time (RT) on different visualization techniques. The techniques to be tested included: (1) Juxtaposed views with the original windows; (2) Juxtaposed full views; (3) Juxtaposed pattern outline views; (4) Pattern vector views; and (5) Pattern change views. The first one is the competitor, and techniques 2, 3, 4, and 5 use the merged windows.

The experiments details are as follows:

Datasets and data patterns: In this experiment, I chose the traffic data and set the length of the current view to one day. The target data pattern was linear trends. The length of one time window was 30 minutes. The number of merged windows is set to 6. I picked 2 sensors and generated 2 figures for each technique, resulting in 10 figures.

Questions: Every participant was asked to observe each figure on a laptop monitor and answer: "When did the biggest change of the fit line slope happen?" Note that one figure using technique 1 contains 47 scatterplots, so users are allowed to apply zooming on figures when exploring them. 8 graduate students in computer science participated in this user study.

Figure 5.16 shows the screenshot of a question used in this experiment.

Question 1

Assume all datapoints in each sliding window agree with a linear model via a fit line. When did the biggest change happen for the fit line slope?

(a) 0:30; (b) 10:30; (c) 19:30; (d) I do not know.

Submit

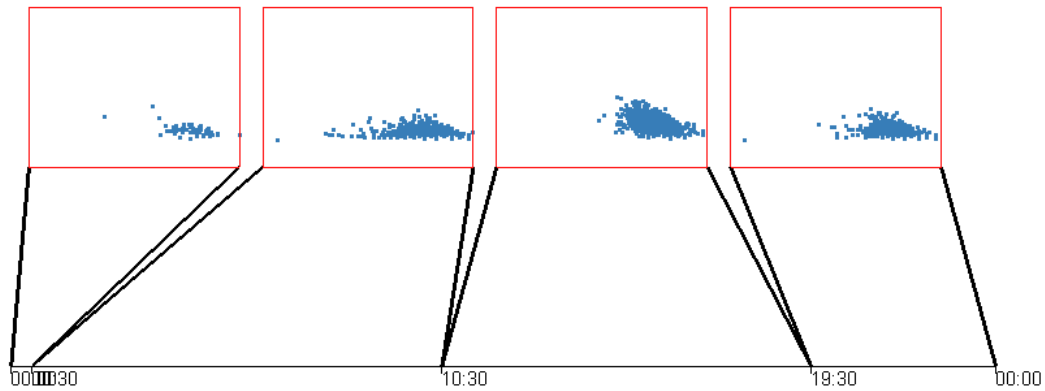


Figure 5.16: This is a question used in the user study for the data-driven framework. The figure is generated using juxtaposed full views along with superimposition technique. Participants need to identify when did the biggest change happen for the fit line slope.

Experiment Results: Since there was no significant difference for the RA using the five techniques, I only calculated the average RT shown in Figure 5.17 with 95% confidence interval, and compared the RT of different techniques using a paired samples t-test. The statistical result revealed that the proposed techniques (Techniques 2-5) have significantly shorter response time than the visualizations of the original windows ($p < 0.01$). The RT of pattern vector and pattern change views are significantly shorter than the full view using merged windows ($p = 0.011$ and $p = 0.006$) as expected. However, the difference between the RT of pattern vector (pattern change) views and pattern outline views using merged windows is not significant ($p = 0.115$ and $p = 0.053$). This might be because the sample size was small.

Based on the experiment result, the conclusion is that the proposed visualization tech-

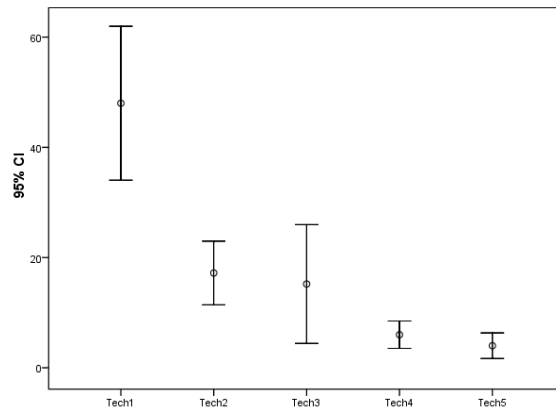


Figure 5.17: The response time for five techniques with 95% confidence interval. Tech 1: juxtaposed views with the original windows. Tech 2: juxtaposed views (full view). Tech 3: juxtaposed views (pattern outline). Tech 4: pattern vector views. Tech 5: pattern change views. Note that Techniques 3, 4, and 5 use the merged windows.

Techniques combined with the merge algorithm can significantly reduce users' response time when exploring the linear trend changes on streaming data. In the future, I plan to introduce other data patterns, such as data range, into this experiment. More participants will also be invited.

Chapter 6

History Views for History Data Using Nested Hierarchical Timelines

6.1 A Framework to Visualize History Data Using Nested Hierarchical Timelines

Figure 6.1 shows how to generate history views. This framework assumes that the streaming data can be defined using a hierarchical structure. At each level, users can define a time unit, and then the streaming data is split into many segments. One segment at one specific level could contain several segments at the lower level. Note that the segments at the bottom level are the time windows mentioned in the prior chapters. For example, traffic can be defined at five levels, including year, quarter, week, day and half hour. One year contains 4 quarters, each of which has 13 weeks, and so on. If the data does not have this structure, a hierarchical structure with only one level can be defined. This structure is shown at the left side of Figure 6.1. It has totally n levels. For the traffic data, $n = 5$. Each segment at levels L_0, L_1, L_2, L_3 and L_4 corresponds to half hour, day, week, quarter and year, respectively.

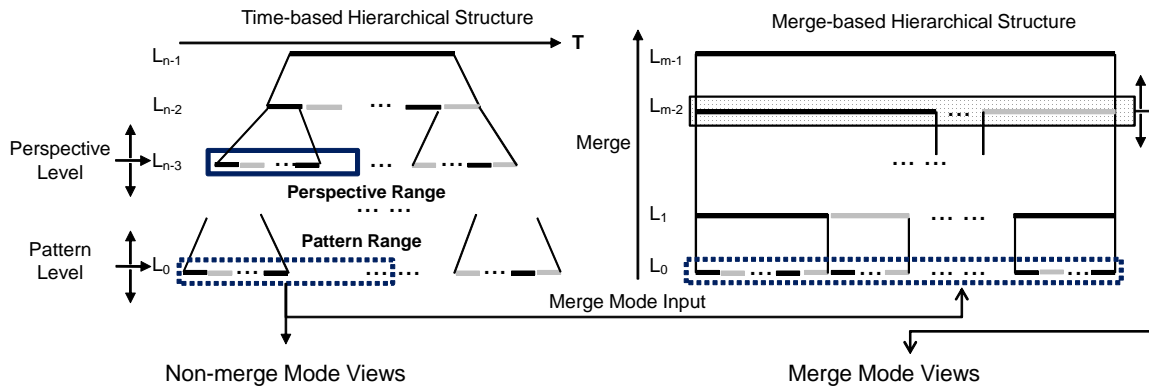


Figure 6.1: The framework to generate history views using nested hierarchical timelines. Left side shows hierarchical time units that contains two levels, perspective and pattern. At both levels, users can specify time ranges, named perspective and pattern ranges. All time windows in the pattern range can be directly output to a non-merge mode view, or the merge algorithm to generate a merge-based hierarchical structure. Users can select a specific level on this structure. All time windows on this level will be output to a merge mode view.

On this user-defined hierarchical structure, users can specify a *pattern level* and a *perspective level*. Each time window on two levels are called *pattern window* and *perspective window* respectively. The former indicates the time unit in which users want to observe the data patterns. On the latter level, users can define a time range that is called *perspective range* (highlighted by a blue solid line rectangle at the perspective level). The blue dashed line rectangle at the pattern level contains all time periods (*pattern range*) that users want to observe pattern changes. For instance, imagine that users move the perspective level to week, and the pattern level to half hour, and then select a specific week. Thus the pattern range should contain all time windows (half hours) within this week. In this case, users focus on investigating how data patterns change across these time windows within this week.

Now, the key task is to visually convey the pattern changes within the pattern range. To solve this problem, two approaches, named non-merge and merge modes, were designed. For the non-merge mode, I generate the visualizations for each time period and organize them on the history views called “non-merge mode views”, using layout strate-

gies proposed in prior chapters. In addition, some new approaches appropriate for history data will be proposed to form the final output. The above solution is straightforward and could cause long response time if there are too many time periods in the pattern range. The reason has been discussed in Section 1.2. Thus the merge modes were designed to display fewer visual elements via the merge algorithm (Section 5.1) while keeping the primary pattern changes. Slightly different from Algorithm 2 in Section 5.1, I set $N_m = 1$ (the number of merged windows). It means that all time periods will eventually be merged to one merged window. During the merge process, the intermediate results after each call to the single step merge (Algorithm 1) are recorded, thus producing a merge-based hierarchical structure (Figure 6.1). Users can choose a level in this structure, and the system will then form the visualizations using only the merged time windows on a selected level. A higher level can allow users to focus on the primary trends of the data, while a lower one conveys more details but with possibly more visual clutter. Note that sometimes in the merge process it is necessary to merge time periods at a level that is not at the bottom in the time-based hierarchical structure. For example, if users want to investigate how the patterns changed from one day to the next within one year, it is needed to merge adjacent days if the changes on traffic patterns are small, so the final output has enough space to show significant changes, e.g., from weekends to weekdays. This is different from the discussion about the merge algorithm in Section 5.1, where the requirement is to merge the time windows at the bottom level (the leaves of the tree at the left side in Figure 6.1). To solve this problem, it is necessary to define the distance between two time periods at a higher level, i.e., two days or weeks. This definition depends on the application area and users' interests. For example, one possible definition for the distance between two days of traffic is the difference between the average volumes of two days, while another one could be the summation of the slope difference of the regression lines (occupancy against speed), between the corresponding time windows.

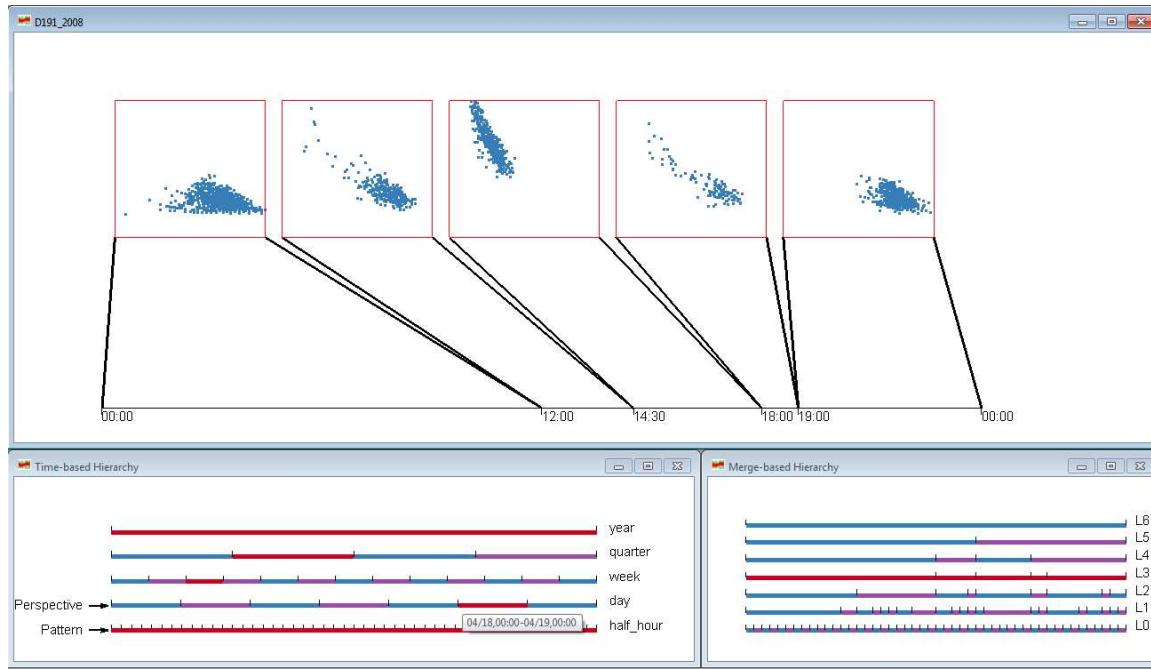


Figure 6.2: A snapshot of history views showing merged mode views.

Figure 6.2 shows a snapshot of the implemented visualization system based on the above proposed framework under the merged mode. This system is composed of three views: time-based hierarchy (bottom left), merge-based hierarchy (bottom right), and history views (upper section). The first two views correspond to the time-based hierarchical structure and the merge-based hierarchical view in Figure 6.1, respectively. The history views can be non-merged views or merged-views based on users' selection. A merged one is shown in Figure 6.2. This figure shows the traffic data from sensor D191 (close to the intersection of I-35W and 35th Street) during the period from Jan. 1, 2008 to Dec.31, 2008. In the time-based hierarchy, one timeline is shown for each level in this dataset. Thus this hierarchy has five timelines, corresponding to year, quarter, week, day and half hour. Users can use the mouse to drag the perspective and pattern level tag to change them. Note that the perspective level is set to the day and the pattern level is on the half hour. Since the data is only for one year, the top level has only one segment that is always selected and highlighted in dark red. Four segments in the second level corresponds to

four quarters in this year. For instance, the second segment is from April 1 to June 30. Users can click one segment to select this quarter and highlight it in dark red, then the timeline in the week level contains the thirteen weeks in this quarter. That means the first segment is the week from April 1 to April 7, and the second one corresponds to the following week (April 8 - 14). Users can continue to select one week on the week level timeline, and then do similar things on the following levels, until reaching the perspective level, the day timeline. If one day on this level is highlighted, all time windows on the pattern level (half hour) will be highlighted. In Figure 6.2, April 18 is selected at the perspective level. Then, the highlighted time windows (half hours, i.e., pattern level) in this day are highlighted and output to the merge algorithm for generating the merge-based hierarchy (the bottom right section of Figure 6.2). In this hierarchy, users can select and highlight a whole level instead of one segment, which is different from the time-based hierarchy. Then all merged windows on the selected level will be visualized in the history view at the upper section of Figure 6.2, named *history view*. Note that this figure uses the 1D layout discussed in Section 5.3. The grid layout is also implemented in this system for merge-mode views.

For non-merge mode, it is necessary to develop some new techniques, because a normal case is to visualize tens or hundreds of time windows in one view. Under such a situation, both 1D and grid layout will cause too much visual clutter and fail in conveying pattern changes. Figure 6.3 shows a grid view that presents the slope change for the regression lines across three months. This figure contains two views: a time-based hierarchy (bottom) and a grid view (top). The bottom view is the same as the merged mode, while users select the quarter as the perspective level, and the day as the pattern level. Since the second quarter (April 1 - July 1) was selected, all days within this period were highlighted at the day timeline (pattern level). This quarters contains 91 days, or $91 \times 48 = 4368$ time windows (half hours). Although it is possible to apply the merge algorithm to all 4368

time windows, Figure 6.3 shows a better solution, if users are only interested in the slope change of the regression line between variables *Speed* and *Occupancy*. In the top view of Figure 6.3, each glyph has a curve to show the slope change within one day. It is easy to observe that the curves in the first and last columns are relatively smooth compared to other grids. It shows that the traffic pattern changes within these days are slower than other days. Actually, these two columns correspond to Saturdays and Sundays.

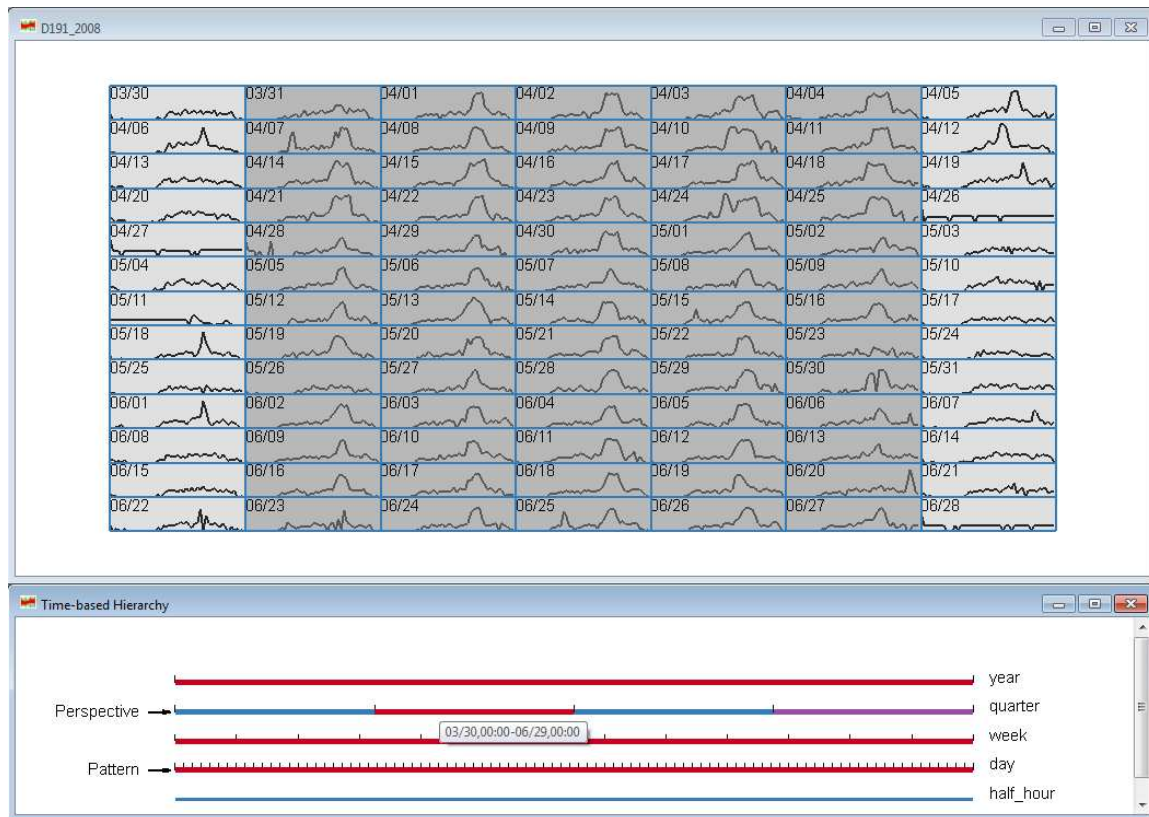


Figure 6.3: This figure shows a history view (top) with hierarchical time structure (bottom) defined by users. They are interested in the changes across contiguous windows on the pattern level (days) in this figure. The selected quarter (March 30 - June 29) on the perspective level is highlighted in red color and indicates the time periods of interest. The red color on the week and day level means all segments in the selected quarter are selected. In the history view, each glyph corresponds to one day (pattern level) and contains a curve to represent the slope change of regression lines within 48 time windows for each day. Grey background is applied to all weekdays to help readers observe data patterns.

Definitions

Some terms used in this framework are given below:

pattern level: A level in the time hierarchy on which a window is a basic unit for users to observe data patterns during pattern evolution. For example, if users want to investigate how traffic patterns change from one day to another, the “day” is the pattern level.

pattern window: A time window on the pattern level.

pattern range: The time range on the pattern level containing pattern windows among which users want to explore the traffic pattern changes.

perspective level: The highest level on which users can select one or more time windows to define the time range containing pattern windows of interest. For example, if users are interested in the traffic pattern changes across days within one quarter, the perspective level is “quarter”.

perspective window: A time window on the perspective level.

perspective range: The selected time range on the perspective level.

6.2 Visualization Techniques for Merged Mode

This section will discuss more details about visualizations in merged mode. One advantage of merged mode views is that the height of the merged-based hierarchical tree can reflect the intensity of the pattern change. More levels indicate a quicker pattern change. In addition, it enables users to choose an appropriate number of subfigures based on canvas size and their requirements. For example, Figure 6.4 contains only 5 levels in the merge-based hierarchy for the data on April 20 (Sunday). However, in Figure 6.2, the merge algorithm generated a hierarchy having 7 levels for the data on April 18 (Friday).

It shows that the traffic pattern changes more frequently on April 20 than April 18. This finding can be confirmed through Figure 6.3.

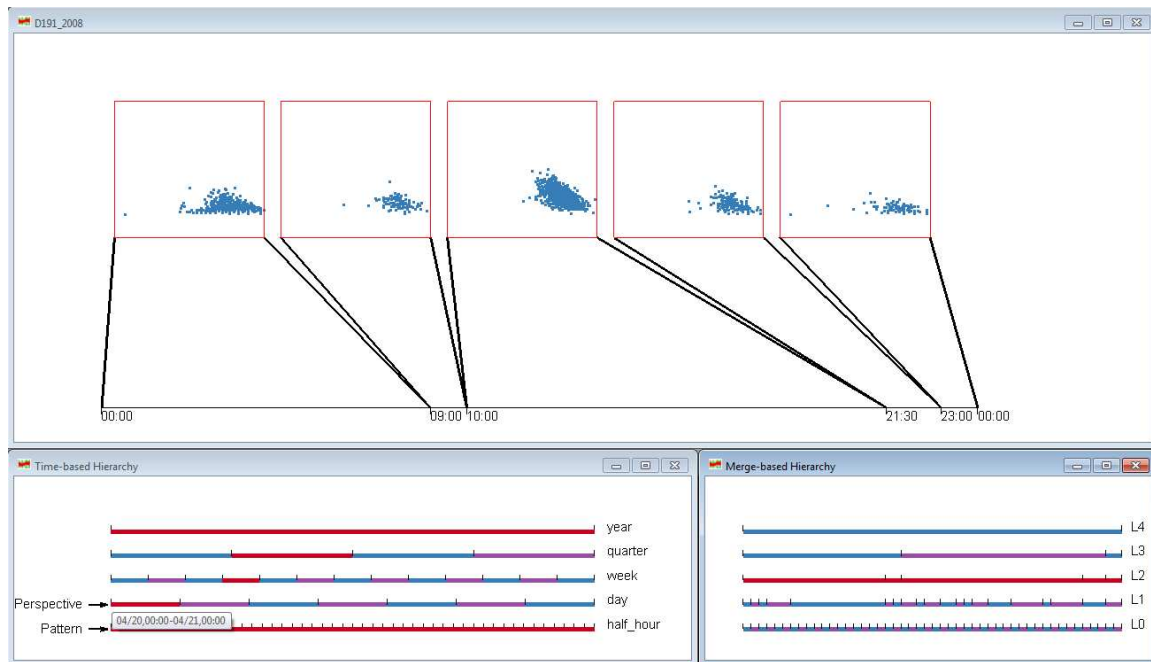


Figure 6.4: A snapshot of history views showing merged mode views with less levels in merge-based hierarchy. The selected day is April 20, 2008.

To help users understand how time windows are merged from one level to the next, an approach, named the *two levels view*, was designed to display the merged windows on two levels together. An example is shown in Figure 6.5. In this figure, two levels, “L2” and “L3”, are selected in the merge-based hierarchy. The corresponding time windows in these two levels are displayed in the history views simultaneously. Two levels both are connected to the same time axis. From this figure, one can clearly see how time windows are merged from one level to the other.

In conclusion, merged mode views can help users perform the following data analysis tasks in the history data:

- Observe the data pattern changes on the pattern level. Users can adjust the number of displayed windows in terms of canvas size and the degree of visual clutter.

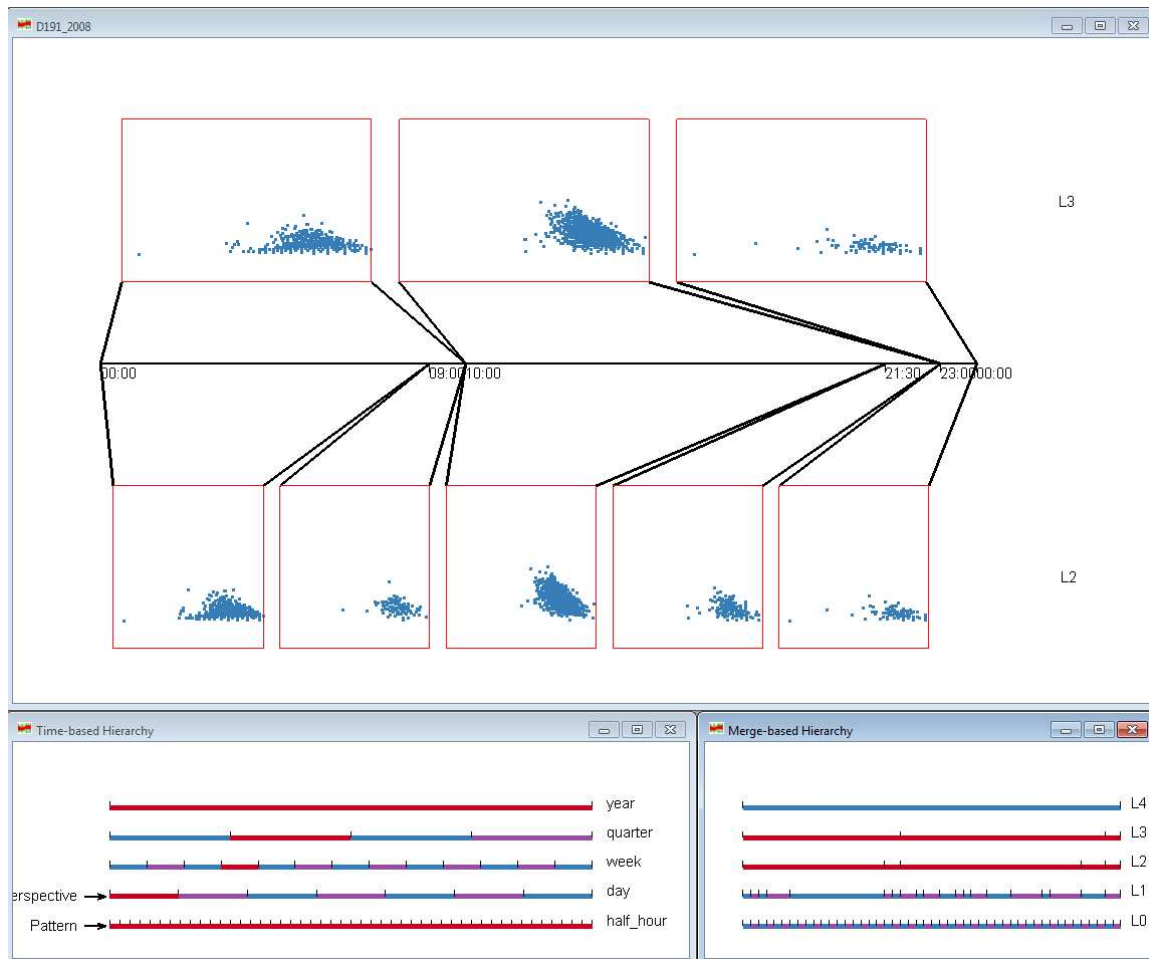


Figure 6.5: This figure shows the merged mode with the *two levels view*. Two levels on April 20, 2008 are selected to show how windows are merged from a lower level to a higher one.

- Investigate how time windows are merged from one level to another via the *two levels view*.

6.3 Visualization Techniques for Non-merged Mode

In Section 6.1, Figure 6.3 shows a non-merged grid view. It has some obvious disadvantages: (1) This technique cannot work for the case where the difference between the perspective level and pattern level is bigger than 2; (2) It does not explicitly convey the

pattern change from one day to the next. In this section, extensions will be applied to the proposed approach for solving the above problems. Section 6.3.1 focuses on issue 1; Section 6.3.2 proposes visualization and interaction techniques for explicitly representing the pattern changes.

6.3.1 Virtual Calendar View

In many cases, users might need to observe the pattern changes within a bigger time range. For example, in the traffic data, a common analysis task is to observe how patterns for each day change across one year. Since its hierarchical structure has five levels (year, quarter, week, day and half hour), the perspective level should be year, and the pattern level is day. The difference between these two levels is 3, so the visualization technique proposed in Section 6.1 does not work unless a certain extension is applied to it. For this data analysis task, the solution is to render one grid view for each quarter, and then generate the final visualization by laying out four views horizontally (Figure 6.6). Note that this is similar to a calendar, thus it can be called a *virtual calendar view*. The reason why to call it *virtual* is that this approach can be used on the streaming data having a hierarchical structure not based on natural time units (year, quarter, week and so on). In such a case, the view is not a real calendar.

The above approach was inspired by the Wijk and Selow’s calendar view [64] and *MulteeSum* [44] developed by Meyer et al. Wijk and Selow’s work used a real calendar to visualize the numbers of employees present at a research center that were encoded by colors. Meyer et al. visually represented the gene expression profiles of cells via a small-multiple matrix of line charts. Each row corresponds to a cell while each column is a gene. Then one line chart can convey the time series data for the expression of one gene in a specific cell. My non-merge mode views are very similar to *MulteeSum*, but rows, columns, and glyphs all are representing different time units in a hierarchical way.

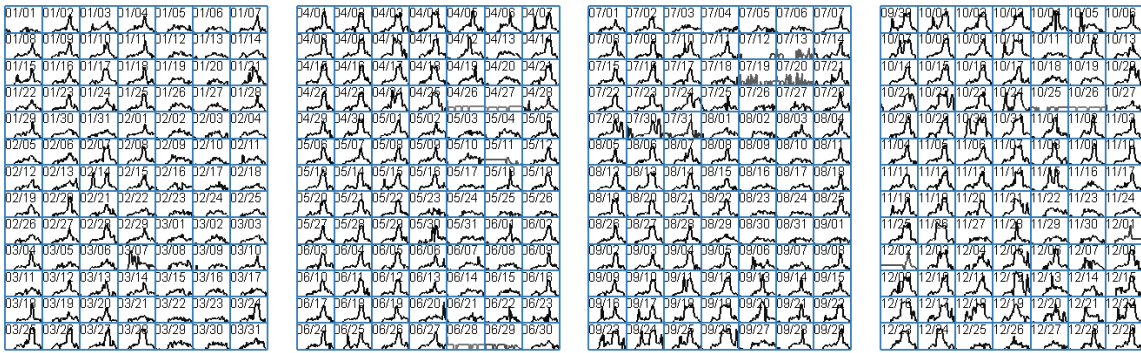


Figure 6.6: 2D grid view is extended to create a *virtual calendar view*.

Figure 6.7 shows how to generate the virtual calendar view. Compared to Figure 6.1, a new level, grid level, was added. It is just two more layers higher than the pattern level. If users selected a range in the perspective level, it will contain some continuous segments on the grid level, which can be called the *calendar range*. For each segment in this range, a grid view is generated. In Figure 6.6, each segment on the grid level corresponds to one quarter. All grid views are then organized horizontally, vertically, in a bigger grid, or via other layout strategies to obtain the final visualizations. In theory, this approach can work regardless of the number of segments in the calendar range. However, if this number is too big, the quality of the final output will be very low because of too much visual clutter. Thus, in real applications, developers should not allow users to select too many segments on the grid level to avoid low quality output.

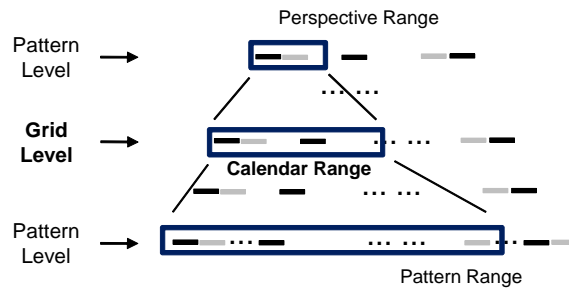


Figure 6.7: To generate the *virtual calendar views*, a grid level is added to the framework in 6.1. Each segment on the grid level corresponds to a grid view in the final visualization.

6.3.2 Explicitly Conveying Pattern Changes

In all the above visualizations, the patterns for each segment in the pattern level were conveyed to the users. It is true that data analysts can investigate how patterns change across the selected time range by observing the whole figure and comparing glyph shapes. However, this is time consuming especially if there are hundreds of glyphs on the final output. In this section, two visualization techniques that can explicitly convey the pattern changes will be discussed: *MDS pattern starfield* and *distance map*. Then some interaction techniques based on them will be introduced.

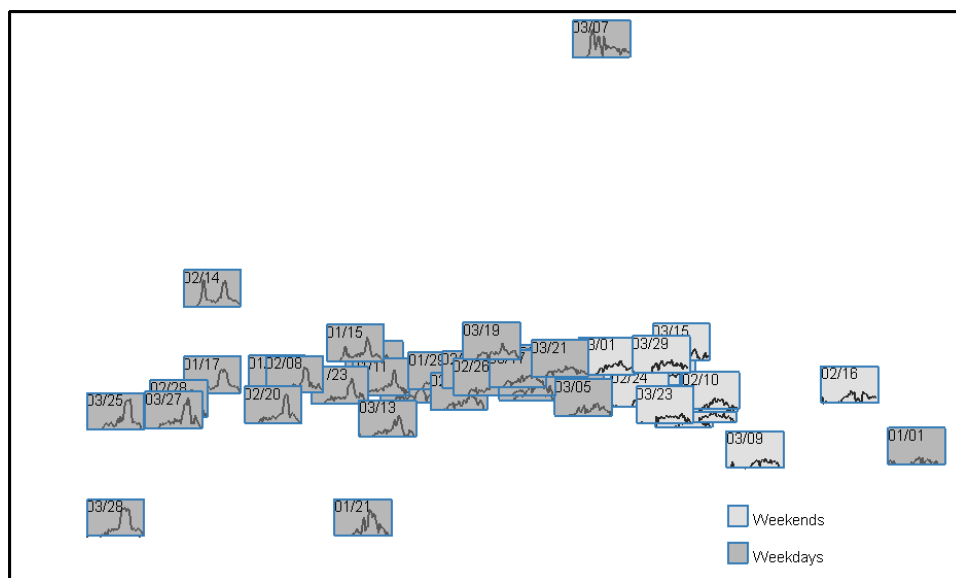


Figure 6.8: MDS algorithm is used to generate positions for cells. The distance between grids represents the distance between corresponding cells.

MDS Pattern Starfield

MDS is a commonly used approach in data visualization to convey the distance among multiple objects. For example, Yang et al. developed the MDS VaR display to visually represent the distances among multiple dimensions in a large-scale multivariate dataset [68, 67]. Figure 6.8 shows an MDS pattern starfield to present a pattern space for the first quarter in the same dataset as what is used in Section 6.1. Similar to Figure 6.3, each

glyph corresponds to one day, i.e., one pattern window. This layout was generated using an MDS algorithm [35]. The proximity among glyph positions reflects pattern distances. Assume that users want to observe N pattern windows, the procedure to get such a starfield is as follows: (1) Distances among these pattern windows are calculated and recorded in an $N \times N$ matrix. (2) This matrix is regarded as the input to an MDS algorithm [35], which generates a position for each pattern window. (3) Each pattern window is rendered as a glyph in the position obtained from the MDS algorithm.

The advantage of this approach is obvious. First, users can easily observe the distribution of pattern windows and clusters in pattern space, since this layout conveys the distance among pattern windows. Then different actions, such as manual clustering and outlier detection, can be easily applied to some pattern windows. For example, in Figure 6.8, one can see that there are several outliers: the glyphs corresponding to the pattern windows on Jan. 1, Jan. 21, Feb. 14, Feb. 16 and March 7. In addition, one interesting phenomena is that weekends mainly occupy the right part of the figure while weekdays are in the middle and left sections. This is easy to explain because weekend traffic patterns are significantly different from those in weekdays. Actually, the expectation is to have two clusters shown on the output, one representing weekdays, and the other being weekends. Probably because there are some outliers, these two clusters are not clearly separated.

In order to avoid the impact of outliers and observe whether two clusters (weekends and weekdays) exist in this dataset, I introduced an interaction technique to allow data analysts to remove some glyphs from the figures. When users move the mouse to a specific glyph, they can right click this glyph then this glyph will be removed from the input of the MDS algorithm and it will not be rendered in the final output. Users can repeat this action multiple times to remove more than one glyph. Using this technique, two glyphs, March 7 and Jan. 1, were removed from Figure 6.8, producing Figure 6.9.

Now, two clusters are clearly shown.

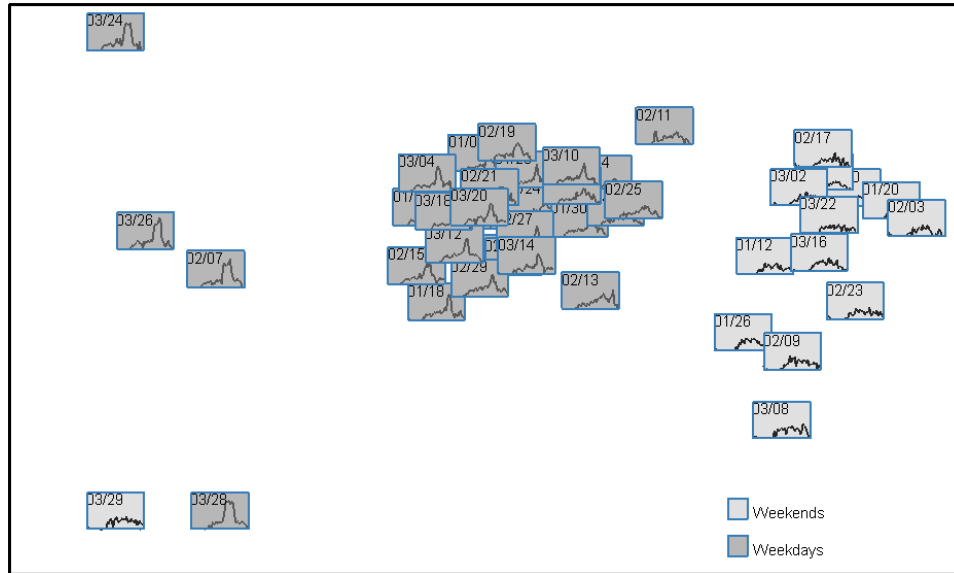


Figure 6.9: Two days, March 7 and Jan. 1, were removed from Figure 6.8, since they are obvious outliers. Now one can clearly see two clusters: weekdays and weekends.

Distance Map

The main goal of the distance map is to convey the pattern distance among pattern windows. Assume users selected N pattern windows in the *Pattern Range*, there are $N \times N$ possible distance measures to be represented. It is not practical to show all these measures in one figure. For example, there are $364 \times 364 = 132496$ distance measures in Figure 6.6. Actually, in most data analysis tasks, users probably are only interested in the distance between two specific pattern windows, or one target window and all others. A possible scenario is as follows: the data analyst finds one interesting pattern window, and then wants to investigate how this window is different from others, or how patterns change around this window. Thus what needs to be shown is the distance measures between this target window and its neighbors. Therefore, I allow users to specify a target pattern window, and then use the distance map to show the pattern distance measures between this target and others.

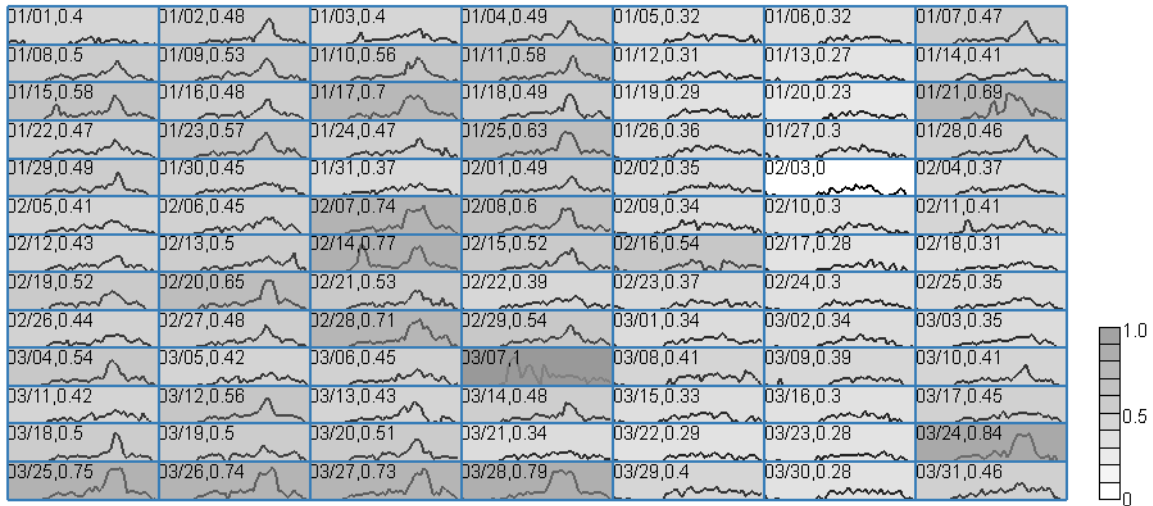


Figure 6.10: This figure shows an example of the *distance map*. One day (Feb. 3) is selected as the target pattern window. The distance measures between other pattern windows and this day are printed in each grid and explicitly represented by the background color. The legend shows how the distance measures are mapped to colors

The other problem is how to visualize the distance measures. Since the color is a visual variable that has a high degree of preattentive processing [61], an encoding technique is used to map distance measures to glyph background colors. The generated output is called a *distance map*. One example is shown in Figure 6.10. In this figure, Feb. 3 is selected as the target pattern window. The implemented visualization system calculates the distance measures between this target windows and all others, and then shows all measures via the glyph background color. All distance measures are normalized before visualizing via the following formula:

$$d' = \frac{d - d_{min}}{d_{max} - d_{min}}$$

where d_{max} and d_{min} are the maximal and minimal distance, d is the real distance and d' is the normalized value. For the examples in this sections, $d_{min} = 0$, since the distance between one pattern window and itself is 0, and all distance measures are positive values. Obviously, the biggest distance measure will be normalized to 1. This distance is also

printed in each glyph. For example, in the glyph for Jan. 1, the number 0.4 is the normalized distance measure between Jan. 1 and Feb. 3. One can also find that March 7 has a normalized distance equal to 1 (maximal possible value), and its glyph has the darkest color.

In Figure 6.10, one finding is that most glyphs in the first column (Sundays) and the last column (Saturdays) have a smaller distance to Feb. 3 (Sunday) since their background colors are brighter than other columns. That is to say, the weekday columns (the second to the sixth) have a smaller similarity to Feb. 3 than the Sunday and Saturday columns. This finding is consistent with common sense that normally the traffic patterns in weekends are different from those in weekdays.

One disadvantage of the distance map is that it might be difficult to distinguish different colors when the difference measures among pattern windows are slight. Actually, the difference between background colors of weekday and weekend columns is not obvious in Figure 6.10. It might require a long response time to draw the above conclusion. In order to shorten the user's response time, an interactive technique, namely a *pattern brush*, is introduced. Its main idea is to use two colors to distinguish a small difference and a bigger one. Thus users can easily investigate the pattern changes across pattern windows.

Pattern Brush

Brushing is a commonly used interaction technique to allow users to select a subset of data via a query [5]. In order to use brushing, all pattern windows are regarded as a set $S = \{W_i | 1 \leq i \leq n\}$ and then are divided into two subsets:

$$S_1 = \{W_i | d(W_i, W') < \delta, 1 \leq i \leq n\}$$

and

$$S_2 = \{W_i | d(W_i, W') \geq \delta, 1 \leq i \leq n\}$$

where W' is the target pattern window, δ is a distance threshold. Thus, S_1 contains those pattern windows closer to the target window than S_2 . In the existing literature, there are a lot of techniques [14] to highlight a subset of interest. Here I propose to use the fog technique used by lots of visualization systems, such as Tableau [57]. If fog is applied to a subset (S_1 or S_2), the other one will be highlighted. For example, in Figure 6.11, the subset S_1 is highlighted by applying fog to S_2 ($\delta = 0.4$). It is obvious that most glyphs having a distance measure smaller than 0.4 are in the weekend columns. This conclusion is the same as what one draws via Figure 6.10. However, users can more easily draw this conclusion from Figure 6.11 than Figure 6.10. This shows that this approach achieved the goal of shortening user response time.

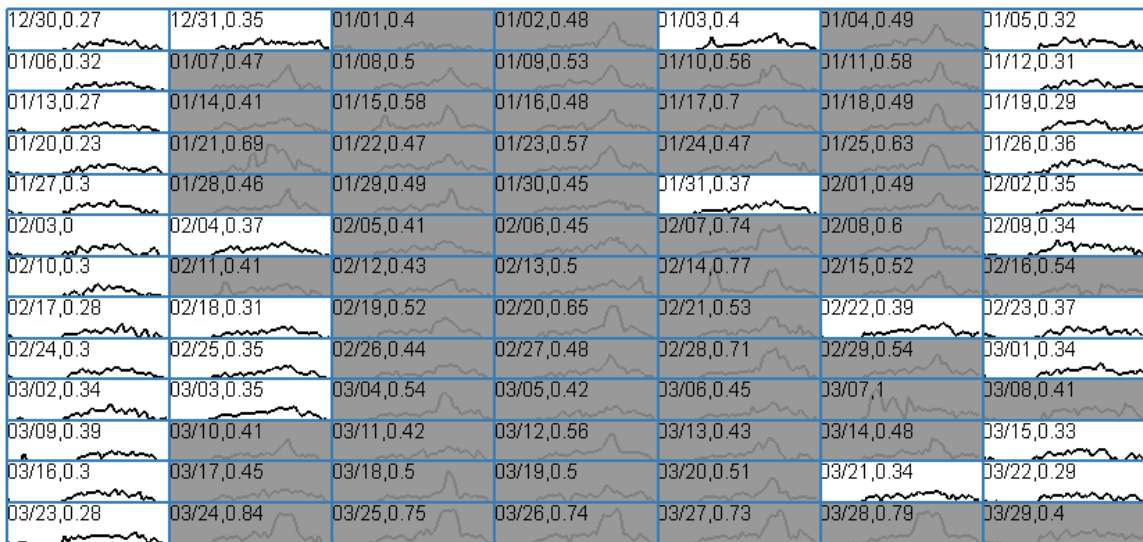


Figure 6.11: A fog effect is applied to the glyph whose distance to Feb. 3 is bigger than $\delta = 0.4$. This can make those pattern windows close to the target pattern window more obvious than Figure 6.10.

In order to enable users to adjust the distance threshold, I provide a slider in the configuration dialog. If users adjust the distance threshold in Figure 6.11 from 0.4 to 0.5, Figure 6.12 is generated. Compared to Figure 6.11, more pattern windows are highlighted. All weekend pattern windows are highlighted except Feb. 16. Another observation is that

some weekday pattern windows have a similar curve to the target pattern window Feb. 3. This leads to some more interesting findings: (1) Feb. 16 is different from regular weekends, even though it is a Saturday. If observing its curve shape carefully, the finding is that it is also different from most weekdays. Actually, it is an outlier (recall Figure 6.8). (2) Some weekdays have similar traffic patterns to Feb. 3(Sunday), such as Dec.31, Feb. 4, Feb. 18, and so on. It means that the traffic in these days is not heavy. Some of them are easy to interpret. for example, Dec. 31 is New Year Eve and Feb. 18 is the Washington’s Birthday. Most people did not go to work on these two days.

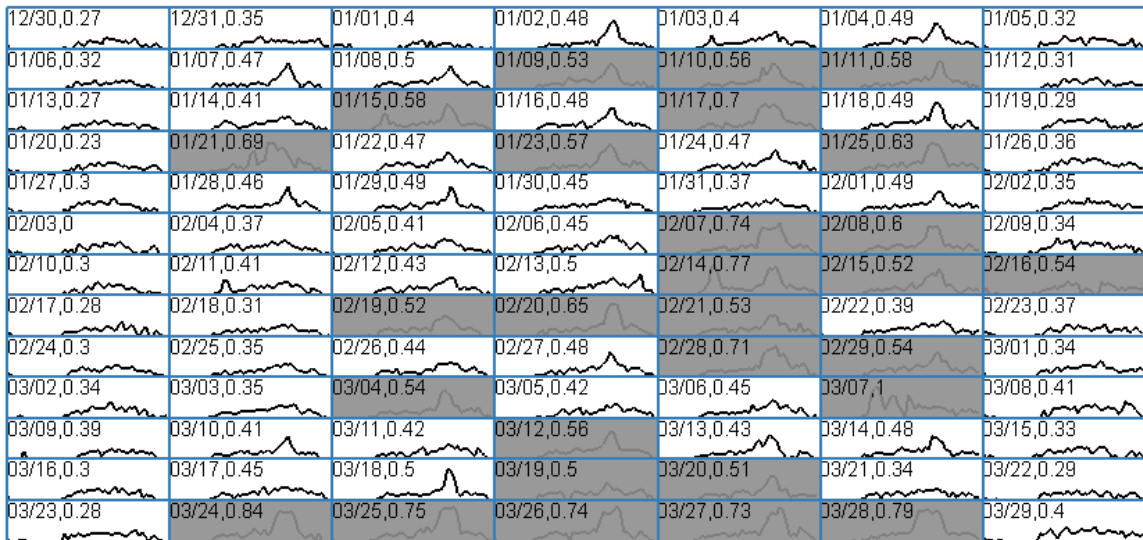


Figure 6.12: This figure uses the same dataset and technique to Figure 6.11 but with a bigger threshold $\delta = 0.5$. Fewer glyphs are dimmed than Figure 6.11.

6.4 Usability Evaluation

In the prior sections, I showed the proposed history views under merged and non-merged mode views along with their associated interaction techniques. Many examples have demonstrated that the proposed techniques can help data analysts efficiently discover how data patterns change across pattern windows in different hierarchical levels over a

very long time period. However, the discussion and examples in the prior sections is not enough to show the usability of the implemented system based on the proposed framework. A pending question is: can users really understand concepts about the proposed data model and visualization/interaction techniques and learn to use this system? To answer this question, I designed a usability evaluation and invited some people to use the implemented system for performing some specific data analysis tasks. Below is a list of questions I wanted to answer in this usability experiment:

- **Timelines** (Navigating Timelines): Can users navigate two timelines to find information of interest? For example, can they correctly select one specific day in the user-defined hierarchical timelines? Can they choose one appropriate level in the merge-based hierarchy for observing the data pattern change with a pattern window?
- **Merge Mode** (Investigating Merge Mode Views): Can merge mode views enable users to easily observe the pattern change within a time period?
- **Grid Views** (Browsing Grid and Virtual Calendar Views): Can data analysts find the pattern changes of interest, such as periodic phenomena and pattern trends, without difficulty via grid and virtual calendar views?
- **MDS** (Understanding MDS Pattern Starfield): Is the MDS pattern field an efficient way to help users find clusters and outliers among pattern windows?
- **Distance Map** (Mastering Distance Map and Pattern Brush): Is it easy for people to use the pattern brush to understand the similarity among pattern windows?

The basic idea to designing this experiment is as follows: (1) design some data analysis tasks on streaming data related to the above questions; (2) invite participants to per-

form these tasks; (3) record the average response time and response accuracy for each task, based on which the above questions can be answered.

In this experiment, I continue to use the traffic data used in Sections 6.1 and 6.3, since it has all features, such as cyclic changes and outliers, I wanted to ask participants to look for. Eleven software engineers from Microsoft attended this experiment. All of them have experience with simple visualizations, including line charts, bar charts and scatterplots, but had not worked with some advanced ones, such as the MDS layout. Participants first received training to familiarize themselves with my system, and then performed 24 tasks belonging to 12 categories.

The list below contains 12 types of tasks this experiment asked subjects to perform:

Timelines (Navigating Timelines)

1. Find a specific day in the time-based hierarchy.
2. Find a specific half hour in the time-based hierarchy.

Merge Mode (Investigating Merge Mode Views)

1. Find where the biggest change of the regression line slope is in a specific day.
2. Find up to three biggest changes between adjacent time windows in the regression line slope in a specific day.

Grid Views (Browsing Grid and Virtual Calendar Views):

1. Observe the traffic pattern trends within a specific quarter using the grid views and choose the correct description from multiple choices.
2. Observe the traffic pattern trends within one year using the virtual calendar views and choose the correct description from multiple choices.

MDS (Understanding MDS Pattern Starfield)

Task Type	Task No	Response Time (Seconds)		Response Accuracy	
		Avg	Std Dev	Avg	Std Dev
Timelines	1	8.3	3.8	1.0	0.0
	2	11.3	3.6	1.0	0.0
Merge Mode	1	130.8	14.9	0.86	0.23
	2	160.5	29.7	0.82	0.25
Grid Views	1	197.0	45.9	1.0	0.0
	2	168.0	54.2	0.95	0.15
MDS	1	21.7	10.1	0.95	0.15
	2	33.0	15.7	0.91	0.20
Distance Map	1	113.0	41.0	0.73	0.34
	2	121.5	41.4	0.77	0.26
	3	41.1	17.0	0.91	0.20

Table 6.1: The response time of usability experiment for history views.

1. Look for the outliers or clusters, if any, in a MDS pattern starfield.
2. Remove one or more outliers until they can clearly see the clusters.

Distance Map (Mastering Distance Map and Pattern Brush)

1. Find the top 10 glyphs closest to a specific day regarding the traffic pattern trends using the distance map. If users can find more than 7 correct glyphs, the answer was treated as correct.
2. Find the top 10 glyphs farthest from a specific day regarding the traffic pattern trends using the distance map. The standard for correct answers was same as the previous task.
3. Set a specific distance threshold to highlight only 10 glyphs closest to a specific day regarding the traffic pattern trends using the pattern brush. If 8-12 glyphs are finally highlighted, the answer was treated as a correct one.

Table 6.1 lists the response time and response accuracy for these 12 tasks.

From these experiments, the conclusion are:

- Users can easily understand the time-based and merge-based hierarchy.
- The merge-based hierarchy can be effective in helping users browse a short time period with uneven pattern change rates. Most users can locate the pattern changes correctly.
- Users can retrieve pattern trends from the grid views and the virtual calendar views with relatively ease. Although the tasks normally took them about three minutes on average, the response accuracy is very high.
- MDS pattern starfield conveyed the clusters and outliers in pattern windows very well. Users quickly learned how to remove outliers from the view and make the clusters separate.
- The first two types of tasks in “Distance Map” have a low but acceptable response accuracy, while most users performed very well on the last type of task. It shows that the distance map achieved the goal to represent the pattern distance among time windows, but needs more improvement. Most users complained that it is difficult for them to distinguish the background color for glyphs, so a better color scheme is necessary. In addition, outliers can be removed to make color difference more obvious.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this dissertation, I have proposed a group of visualization techniques to visually convey the data pattern changes in multivariate data streams.

User driven approach

The basic idea of the user driven approach is to display both the current data and abstractions of past data to show how changes occur over time. The whole stream is split into non-overlapped time windows and pattern-blind averaging is applied to each window. The sampling ratio for a particular window is determined by a DOI (degree of interest) function to reflect users' interests. A larger DOI value results in a larger sampling ratio for the specified window, meaning more details shown in this view. Two types of DOI functions are provided to satisfy common data analysis tasks, as well as a DOI function interactive tool to allow users to adjust the DOI function when exploring data streams. In order to show how data patterns change, I have proposed four layout strategies, namely, superimposition, juxtaposition, step juxtaposition, and animation, to place time windows in the final views. The evaluation showed that three of these four visualization techniques

can effectively convey the multivariate pattern change compared to the traditional time-series data visualization techniques. A guide was derived to advise data analysts and visualization system developers in choosing appropriate layout strategies based on the characteristics of datasets and data analysis tasks. In addition, users are allowed to use multiple views in the final visualization and use linked brushing to highlight a subset of interest in all views when defining a query in one view.

Data driven approach

This approach addresses the problem of how to efficiently visualize pattern changes on a data stream given the fact that the pattern change rate is not constant. Distorting the time axis can partially solve this problem, but most existing techniques are user-driven. This is not applicable to data streams that normally need quick responses. A data-driven approach is proposed to automatically merge adjacent time windows with few or no changes in the current view. A group of experiments show that the proposed merge algorithm can preserve more change information than pattern-blind averaging. I proposed two types of visualization techniques: juxtaposed full views and outline views. The former keeps the data details while the latter aims to convey only the data patterns users want to observe. A user study was conducted to confirm that the proposed visualization techniques together with the merge algorithm can significantly reduce the time cost to detect pattern changes over data streams.

History views

The history views are used to help users explore the data pattern changes within a relatively long time period. The data analysis tasks normally are off-line and do not need urgent response. The proposed history views work under two modes: non-merged and merged. In the former mode, users need to define a hierarchical structure to represent timelines. The definition of the hierarchy can be from natural time units, such as year, quarter, and month, or domain specific units. When users select one time period on the

first timeline hierarchy, I generate the history view containing all time windows in this time duration. In each glyph, the abstraction of data patterns are drawn instead of the original datapoints. All glyphs comprise a grid or virtual calendar. Other approaches, such as an MDS pattern starfield, distance map, and pattern brush, are designed to assist users in discovering the similarities of time windows regarding the target data patterns. For the merge mode, time windows selected by users are sent to the merge algorithm and a merge-based hierarchy is generated. The contiguous time windows having similar patterns will be merged first. If users want to know more details about the data, they can choose one lower level in this hierarchy, but with more visual clutter. Otherwise, they can choose a higher level to avoid visual clutter with random sampling, which could lose some details in the data. The usability evaluation demonstrated that most users can understand the concepts in history views and finish assigned tasks, including navigating timelines, finding significant pattern changes, and investigating similarity among time windows.

7.2 Future Works

Although these approaches can effectively help data analysts observe, understand, and retrieve how data patterns change in multivariate streaming data, these techniques can be improved from the following aspects:

- **Target patterns:** When I designed visualization techniques for conveying pattern changes, I always tried to make the proposed solutions be as general as possible for most data patterns. It has been verified that they are feasible for linear trends and data distribution. However, this is not enough. A plan is to apply the proposed techniques to more data patterns, such as data density, clusters, and outliers.
- **Application domains:** All of the proposed approaches are general solutions for streaming data, and do not target at any specific application. The next step is

to apply them to a variety of real domains, such as financial data analysis and video monitoring. It might be necessary to modify the proposed framework and design more visualization and interaction techniques for these applications. The algorithms might need further optimization because of possible strict requirements for response time.

- **More data types:** In all of proposed techniques, an assumption is that streaming data contains only numerical values. However, a lot of application data is not numerical, and includes text, audio, video, networks, and other types. There are two possible ways to handle them: (1) convert them to multivariate data and apply the approaches developed in this dissertation; or (2) directly design new approaches for them. Either is an interesting direction for continued research.
- **More Evaluation:** All user studies in this dissertation invited only participants who did not have expertise in the application domain related to the dataset. A more efficient way is to find domain experts to assess the implemented system using the dataset of their interest. For example, I can visit medical researchers to ask them to observe sleep datasets in the implemented system and see whether and how this can help them better understand their data.
- **Distribute the Code:** Following the tradition of the Xmdv group, releasing the code of this system is an efficient way to get feedback.

Appendices

.1 Pattern Vectors for Linear Models

In this dissertation, *least square* [40], the most commonly-used regression method, is used to estimate the linear model between two variables X and Y in a specific window. Assume that the paired data in this window is (x_1, y_1) , (x_2, y_2) , ..., and (x_n, y_n) . The linear model can be represented by:

$$Y = \alpha + \beta X$$

where

$$\alpha = \frac{R_2 \cdot S_1 - R_1 \cdot T}{n \cdot R_2 - R_1^2}, \beta = \frac{n \cdot T - R_1 \cdot S_1}{n \cdot R_2 - R_1^2}$$

Note that $R_1 = \sum_{i=1}^n x_i$, $R_2 = \sum_{i=1}^n x_i^2$, $S_1 = \sum_{i=1}^n y_i$, and $T = \sum_{i=1}^n x_i y_i$

If the linear model pattern vector of two specific windows, W' and W'' , are represented as

$$V'_p = (\alpha', \beta', n', R'_1, R'_2, S'_1, T')$$

and

$$V''_p = (\alpha'', \beta'', n'', R''_1, R''_2, S''_1, T'')$$

then the pattern vector of the parent window of W_1 and W_2 is:

$$(\alpha, \beta, n' + n'', R'_1 + R''_1, R'_2 + R''_2, S'_1 + S''_1, T' + T'')$$

where

$$\alpha = \frac{(R'_2 + R''_2) \cdot (S'_1 + S''_1) - (S'_1 + S''_1) \cdot (T' + T'')}{(n' + n'') \cdot (R'_2 + R''_2) - (R'_1 + R''_1)^2}$$

$$\beta = \frac{(n' + n'') \cdot (T' + T'') - (R'_2 + R''_2) \cdot (S'_1 + S''_1)}{(n' + n'') \cdot (R'_2 + R''_2) - (R'_1 + R''_1)^2}$$

Note that, in the pattern vectors, only α and β are used to describe the linear model, while

other items serves the operation to merge windows.

The distance between V'_p and V''_{p2} can be computed via:

$$|\arctan \beta' - \arctan \beta''|$$

where I am interested in only the change of fit line slope for the linear model. Other definitions for distance functions can be used if users have different requirements.

.2 Pattern Vectors for Data Range

In this dissertation, I use mean value and standard deviation for each dimension to represent the data range. Other measures can be easily added.

Assume that variable X in a specific window is x_1, x_2, \dots, x_n , its mean \bar{X} and standard deviation s can be represented by

$$\bar{X} = \frac{\sum_{i=1}^n x_i}{n}$$

and

$$s = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{X})^2}{n - 1}} = \sqrt{\frac{R_2 - 2 \cdot \bar{X} \cdot R_1 + n \cdot \bar{X}^2}{n - 1}}$$

where $R_1 = \sum_{i=1}^n x_i$ and $R_2 = \sum_{i=1}^n x_i^2$.

If the data range pattern vector of two specific windows, W_1 and W_2 , are represented as

$$V'_p = (\bar{X}', s', n', R'_1, R'_2)$$

and

$$V''_p = (\bar{X}'', s'', n'', R''_1, R''_2)$$

then the pattern vector of the parent window of W_1 and W_2 is:

$$(\bar{X}, s, n' + n'', R'_1 + R''_1, R'_2 + R''_2)$$

where

$$\bar{X} = \frac{n'\bar{X}' + n''\bar{X}''}{n' + n''}$$

$$s = \sqrt{\frac{(R'_2 + R''_2) - 2 \cdot \bar{X} \cdot (R'_1 + R''_1) + (n' + n'') \cdot \bar{X}^2}{n' + n'' - 1}}$$

The distance between W_1 and W_2 can be defined as:

$$\sqrt{(\bar{X}' - \bar{X}'')^2 + (\bar{Y}' - \bar{Y}'')^2}$$

To make this distance function usable, two variables both should be normalized to $[0, 1]$ first. Other distance functions are possible in terms of specific applications.

Bibliography

- [1] D. Andrews. Plots of high dimensional data. *Biometrics*, 28:125–136, 1972.
- [2] A. Aris, B. Shneiderman, C. Plaisant, G. Shmueli, and W. Jank. Representing unevenly-spaced time series data for visualization and interactive exploration. *Proc. Intl. Conf. Human-Computer Interaction - INTERACT*, pages 835–846, 2005.
- [3] R. Bade, S. Schlechtweg, and S. Miksch. Connecting time-oriented data and information to a coherent interactive visualization. *CHI*, pages 105–112, 2004.
- [4] M. F. Barnsley. *Fractals Everywhere*. Morgan Kaufmann, San Francisco, CA, USA, 1993.
- [5] A. Becker and S. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.
- [6] L. Berry and T. Munzner. Binx: Dynamic exploration of time series datasets across aggregation levels. *IEEE Symp. Information Visualization Poster*, page 215.2, 2004.
- [7] E. Bertini and G. Santucci. Quality metrics for 2d scatterplot graphics: Automatically reducing visual clutter. In *Proc. 4th International Symposium on SmartGraphics*, pages 77–89, 2004.
- [8] E. Bertini and G. Santucci. Give chance a chance- modeling density to enhance scatter plot quality through random data sampling. *Information Visualization*, 5(2):95–110, 2006.
- [9] N. Boukhelifa, F. Chevalier, and J.-D. Fekete. Real-time aggregation of wikipedia data for visual analytics. In *Proc. IEEE Symp. Visual Analytics Science and Technology*, pages 147–154, 2010.
- [10] A candlestick chart to reflect the ratio change between USD and JPY. <http://forex.easy-forex.com.au/images/candlestick-chart2-lrg.jpg>, accessed on Apr. 2, 2008.
- [11] S. Card, J. Mackinlay, and B. Shneiderman. *Readings in Information Visualization*. Morgan Kaufmann, San Francisco, CA, 1999.

- [12] J. V. Carlis and J. A. Konstan. Interactive visualization of serial periodic data. *Proc. Symp. User Interface Software and Technology*, pages 29–38, 1998.
- [13] R. Chang, M. Ghoniem, R. Kosara, W. Ribarsky, J. Yang, E. A. Suma, C. Ziemkiewicz, D. A. Kern, and A. Sudjianto. Wirevis: Visualization of categorical, time-varying data from financial transactions. In *Proc. IEEE Symp. Visual Analytics Science and Technology*, pages 155–162, 2007.
- [14] H. Chen. Compound brushing. *Proc. IEEE Symposium on Information Visualization*, pages 181–188, 2003.
- [15] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. In *VLDB*, pages 323–334, 2002.
- [16] E. H. Chi. *A framework for information visualization spreadsheets*. PhD thesis, University of Minnesota, 1999.
- [17] R. J. Clarke. Image and video compression: a survey. *International Journal of Imaging Systems and Technology*, 10:20–32, 1999.
- [18] Q. Cui, M. Ward, and E. Rundensteiner. Enhancing scatterplot matrices for data with ordering or spatial attributes. *Visualization and Data Analysis, Part of IS&T/SPIE Symposium on Electronic Imaging*, pages 0R1–0R11, 2006.
- [19] A. Dix and G. Ellis. By chance - enhancing interaction with large data sets through statistical sampling. *Proc. Advanced Visual Interfaces*, pages 167–176, 2002.
- [20] G. Ellis and A. Dix. A taxonomy of clutter reduction for information visualisation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1216–1223, 2007.
- [21] G. Furnas. Generalized fisheye views. *Proc. ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 16–23, 1986.
- [22] L. Golab and M. T. Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.
- [23] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000.
- [24] M. C. Hao, U. Dayal, D. A. Keim, D. Morent, and J. Schneidewind. Intelligent visual analytics queries. *Proc. IEEE Symp. Visual Analytics Science and Technology*, pages 91–98, 2007.

- [25] M. C. Hao, U. Dayal, D. A. Keim, and T. Schreck. Importance-driven visualization layouts for large time series data. *Proc. IEEE Symp. Information Visualization*, pages 203–210, 2005.
- [26] M. C. Hao, U. Dayal, D. A. Keim, and T. Schreck. Multi-resolution techniques for visual exploration of large time-series data. *EuroVis07: Joint Eurographics - IEEE VGTC Symp. on Visualization*, pages 27–34, 2007.
- [27] S. Havre, E. Hetzler, P. Whitney, and L. Nowell. ThemeRiver: Visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):9–20, January 2002.
- [28] K. P. Hewagamage, M. Hirakawa, and T. Ichikawa. Interactive visualization of spatiotemporal patterns using spirals on a geographical map. *Proc. Symp. Visual languages*, pages 296–303, 1999.
- [29] H. Hochheiser and B. Shneiderman. Dynamic query tools for time series data sets: Timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, 2004.
- [30] D. A. Huffman. A method for construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
- [31] D. F. Huynh. SIMILE – timeline [<http://simile.mit.edu/timeline/>]. *Massachusetts Institute of Technology*, 2006.
- [32] A. Inselberg. The plane with parallel coordinates. *Special Issue on Computational Geometry, The Visual Computer*, 1:69–97, 1985.
- [33] D. Keim. Designing pixel-oriented visualization techniques: Theory and applications. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):1–20, January-March 2000.
- [34] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 organizers’ report: Peeling the onion. *SIGKDD Explorations*, 2(2):86–98, 2000.
- [35] J. Kruskal and M. Wish. *Multidimensional Scaling*. Sage Publications, Thousand Oaks, CA, USA, 1978.
- [36] N. Kumar, V. N. Lolla, E. J. Keogh, S. Lonardi, and C. A. Ratanamahatana. Time-series bitmaps: a practical visualization tool for working with large time series databases. In *SIAM International Data Mining Conference, Poster*, 2005.
- [37] R. P. Laeser, W. I. Mclaughlin, and D. M. Wolff. Engineering voyager 2s encounter with uranus. *Scientific American*, 255(5):36–45, 1986.
- [38] J. LeBlanc, M. Ward, and N. Wittels. Exploring n-dimensional databases. *Proc. IEEE Visualization*, pages 230–237, 1990.

- [39] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Comput. Surv.*, 19:261–296, September 1987.
- [40] S. J. Leon. *Linear Algebra with Applications (7th Edition)*. Prentice Hall, Upper Saddle River, NJ, USA, 2005.
- [41] B. Lichtenbelt, R. Crane, and S. Naqui. *Introduction to Volume Rendering*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.
- [42] J. Lin, E. J. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 15(2):107–144, 2007.
- [43] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, Maryland Heights, MO, USA, 1999.
- [44] M. D. Meyer, T. Munzner, A. H. DePace, and H. Pfister. Multeesum: A tool for comparative spatial and temporal gene expression data. *IEEE Trans. Vis. Comput. Graph.*, 16(6):908–917, 2010.
- [45] S. Miksch, W. Horn, C. Popow, and F. Paky. Utilizing temporal data abstraction for data validation and therapy planning for artificially ventilated newborn infants. *Artificial Intelligence in Medicine*, 8(6):543–576, 1996.
- [46] S. Miksch, A. Seyfang, W. Horn, and C. Popow. Abstracting steady qualitative descriptions over time from noisy, high-frequency data. *Proc. Artificial Intelligence in Medicine. Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making*, pages 281–290, 1999.
- [47] N. Miller, P. Wong, M. Brewster, and H. Foote. Topic islands: A wavelet-based text visualization system. *Proc. IEEE Visualization*, pages 189–196, 1998.
- [48] Minnesota Department of Transportation. Mn/DOT traveler information. <http://www.dot.state.mn.us/tmc/trafficinfo/>, accessed on Feb. 25, 2009.
- [49] A. V. Moere. Time-varying data visualization using information flocking boids. *Proc. IEEE Symp. Information Visualization*, pages 97–104, 2004.
- [50] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases [<http://www.ics.uci.edu/~mllearn/mlrepository.html>]. *University of California, Irvine, Dept. of Information and Computer Sciences*, 1998.
- [51] G. Nielson, I.-H. Jung, and J. Sung. Wavelets over curvilinear grids. *Proc. IEEE Visualization*, pages 313–317, 1998.
- [52] P. Ren, J. Kristoff, and B. Gooch. Visualizing dns traffic. In *Proc. 3rd Workshop on Visualization for Computer Security*, pages 23–30, 2006.

- [53] J. F. Roddick and M. Spiliopoulou. A survey of temporal knowledge discovery paradigms and methods. *IEEE Trans. Knowl. Data Eng.*, 14(4):750–767, 2002.
- [54] K. Sayood. *Introduction to Data Compression, Third Edition*. Morgan Kaufmann Pub, San Francisco, CA, USA, 2005.
- [55] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualization. *Proc. IEEE Symposium on Visual Languages*, pages 336–343, 1996.
- [56] J. Siegel, E. Farrell, R. Goldwyn, and H. Friedman. The surgical implication of physiologic patterns in myocardial infarction shock. *Surgery*, 72:126–141, 1972.
- [57] C. R. Stolte. Visual interfaces to data. In *SIGMOD Conference*, pages 1067–1068, 2010.
- [58] C. Tominski, J. Abello, and H. Schumann. Axes-based visualizations with radial layouts. *Proc. ACM Symp.on Applied Computing*, pages 1242–1247, 2004.
- [59] E. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, USA, 1983.
- [60] M. Ward and B. Lipchak. A visualization tool for exploratory analysis of cyclic multivariate data. *Metrika*, 51(1):27–37, 2000.
- [61] M. O. Ward, G. Grinstein, and D. Keim. *Interactive Data Visualization: Foundations, Techniques, and Applications*. A K Peters Ltd, Natick, MA, USA, 2010.
- [62] M. Wattenberg. Baby names, visualization, and social data analysis. *Proc. IEEE Symp. Information Visualization*, pages 1–7, 2005.
- [63] M. Weber, M. Alexa, and W. Müller. Visualizing time-series on spirals. *Proc. IEEE Symp. Information Visualization*, pages 7–14, 2001.
- [64] J. V. Wijk and E. V. Selow. Cluster and calendar based visualization of time series data. *Proc. IEEE Symposium on Information Visualization*, pages 4–9, 1999.
- [65] P. Wong and R. Bergeron. Multi-resolution multidimensional wavelet brushing. *Proc. IEEE Visualization*, pages 141–148, 1996.
- [66] Z. Xie, S. Huang, M. O. Ward, and E. A. Rundensteiner. Exploratory visualization of multivariate data with variable quality. *Proc. IEEE Symposium on Visual Analytics Science and Technology*, pages 183–190, 2006.
- [67] J. Yang, D. Hubball, M. Ward, E. Rundensteiner, and W. Ribarsky. Value and relation display: Interactive visual exploration of large data sets with hundreds of dimensions. *IEEE Trans. Visualization and Computer Graphics*, 13(3):494–507, 2007.

- [68] J. Yang, A. Patro, S. Huang, N. Mehta, M. Ward, and E. Rundensteiner. Value and relation display for interactive exploration of high dimensional datasets. *Proc. IEEE Symposium on Information Visualization*, pages 73–80, 2004.
- [69] C. Yu, Y. Zhong, T. Smith, I. Park, and W. Huang. Visual mining of multimedia data for social and behavioral studies. In *Proc. IEEE Symposium on Visual Analytics Science and Technology*, pages 155–162, 2008.