ABSTRACT

Title of dissertation: INTERACTIVE VISUALIZATION TECHNIQUES FOR SEARCHING TEMPORAL CATEGORICAL DATA

Taowei David Wang, Doctor of Philosophy, 2010

Dissertation directed by: Professor Ben Shneiderman
Department of Computer Science

Temporal data has always captured people's imagination. Large databases of temporal data contain temporal patterns that can lead to the discovery of important cause-and-effect phenomena. Since discovering these patterns is a difficult task, there is a great opportunity to improve support for searching. Temporal analysis of, for example, medical records, web server logs, legal, academic, or criminal records can benefit from more effective search strategies.

This dissertation describes several interactive visualization techniques designed to enhance analysts' experience in performing search, exploration, and summarization of multiple sets of temporal categorical data. These techniques are implemented in the software *Lifelines2* (`http://www.cs.umd.edu/hcil/lifelines2/`). Lifelines2 is an interactive visualization system that enables analysts to dynamically change their focus in order to expose temporal ordering of event sequences and study the prevalence of such orderings.

This dissertation makes four main contributions. The first three are technical

contributions, and the last is a process model that generalizes user behavior. First, the *Align-Rank-Filter* framework is presented to help analysts perform visual search and exploration. It enables analysts to center their attention on temporal events that are the focus of their inquiry. Through a controlled experiment, *alignment* alone is shown to improve user performance speed by up to 60% in tasks that require understanding of temporal ordering of events. The initial successful exploration on the alignment operator led to its fuller exploitation. Further enhancements to filtering are presented to better incorporate alignment. Second, I designed and implemented the Temporal Pattern Search (TPS) algorithm for filtering to support the common, but difficult-to-specify *absence of* operator in a temporal pattern. TPS exploits the data structure of the visualization system, and it compares favorably to existing common approaches. Third, I present the *temporal summaries* technique as an overview to support grouping and comparison features in Lifelines2. They support higher-level tasks such as hypothesis generation. These features take advantage of alignment, and the entirety of the system is evaluated in several long-term case studies with domain experts working on their own problems. Fourth, from these long-term case studies, I generalize a process model that describes analyst behavior in searching and interacting with temporal categorical data. Gleaning from observations in the case studies, collaborators' interviews and commentaries, and logs of Lifelines2 usage, I recommend feature design guidelines for future visualization designers for temporal categorical data.

The enthusiasm of the domain experts who used Lifelines2, the changing strategies for problem-solving, and their initial successes suggest these interactive visual-

ization techniques are a valuable addition to search capabilities.

INTERACTIVE VISUALIZATION TECHNIQUES FOR SEARCHING TEMPORAL CATEGORICAL DATA

by

Taowei David Wang

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2010

Advisory Committee:
Professor Ben Shneiderman, Chair/Advisor
Dr. Catherine Plaisant
Professor Ben Bederson
Professor Amol Deshpande
Professor Galit Shmuéli

Dedication

To my parents 王勝雄 and 葉雪蘭

Acknowledgments

The path to a Ph.D. is never easy. I have had the fortune to be surrounded by incredibly capable and supportive people that made this difficult journey of growth and learning seem almost... enjoyable :)

I would like to first thank my advisor Ben Shneiderman, whose tireless enthusiasm and passion for all things HCI and, in particular, our work continue to inspire me. From writing papers with him, I learned how to be a researcher. From his communicating, engaging, and working with others, I learned how to be a better researcher. From the enthusiasm he displayed when communicating, engaging, and working with others, I learned what it means to be a great researcher. As I sit here and write these lessons, I realize that these principles do not just apply to research, but they apply in life in general. Ben, thank you for your lessons in leadership, your intellectual challenges, your inspiring enthusiasm, all of which have contributed to my growth.

I would like to also thank Catherine Plaisant, whose enthusiasm rivals only Ben's. I have learned a tremendous amount from her on the procedures of working with users and many design principles. She is not only a close collaborator of my publications, she has also helped a lot in shaping my dissertation. I thank you for all your teaching and suggestions to my work along the way. I also thank you for being a stout supporter for me.

My journey to Ph.D. had its highs and lows. There was a time when I was not sure if I would continue. In those most desperate days, if it were not for my

incredibly supportive friends, I would not be writing this lengthy document (believe it or not, there is *some* joy in writing it). I would especially like to thank first Jennifer Golbeck, for her guidance and encouragements in those times of uncertainly and despair. If it were not for Jen, I would not have gotten in contact with Ben and Catherine and continued on my journey this far. Secondly, I would like to thank Bijan Parsia. In those darkest hours, he had also tirelessly taken upon himself to advise and to encourage me to continue on the journey to Ph.D.

I would like to thank James Hendler, my previous advisor, for letting me become part of MINDSWAP, where I first began to learn how to do research and write papers, and where I met important friends. I also thank Jim for his generous financial support after he has left the university.

I thank the members of the HCIL for their support, challenges, and suggestions along the way. I have enjoyed the intellectual and the not-so intellectual discussions with the students, faculty, and staff. I like the collective open-mindedness to new and strange ideas, and I appreciate very much everyone's constructive and supportive attitudes toward one another. That made the lab such an idea-fostering and friendly environment. If "Standing on the shoulders of giants" characterizes the building of knowledge from the endeavors of others, I am pleased to say that I have been fortunate to be working with the giants directly in the HCIL day-in and day-out. I have learned a lot by listening and watching the seasoned researchers at work – with devices, with kids, with students. Finally, I thank the fellow HCIL students – the giants of tomorrow –, whose mutual support, critiques and suggestions have made our work that much better.

I thank Mark Smith and his team from the Washington Hospital Center and MedStar Health: David Roseman, Greg Marchand, Phuong Ho, and Vikramjit Mukherjee. This research would not have materialized without their generous financial support and close collaboration. I would also like to thank the following collaborators and supporters: Shawn Murphy from Harvard Medical School and Partners HealthCare, and Neil Spring from the Department of Computer Science, University of Maryland.

I thank my friends, particularly Margaret Tsai, James Phongsuwan, and Holly Martinson for putting up with my neglect and keeping me sane by reminding me that there is a world outside of computer science. I suppose I should also thank my friends on Facebook, but perhaps a status update would be appropriate.

Finally, I would like to thank my family – for every phone call in a holiday to remind me that I am in their thoughts, and for every phone call just because. I would like to especially thank my parents for supporting my decision of going to graduate school and live my adventure. Lastly, I would like to thank my 1993 Buick. Even though you finished before I did, I appreciate your faithful 16 years of service.

Table of Contents

List of Figures

List of Abbreviations

DFA	Deterministic Finite Automaton
EHR	Electronic Health Record
HCI	Human-Computer Interaction
HCIL	Human-Computer Interaction Lab
HIT	Heparin-Induced Thrombocytopenia
ICDL	International Children's Digital Library
ICU	Intensive Care Unit
IMC	Intermediate Medical Care
MI	Miocardial Infarction
NFA	Nondeterministic Finite Automaton
TPS	Temporal Pattern Search
WHC	Washington Hospital Center

Chapter 1

Introduction

As the amount of temporal categorical data grows, the need for effective ways to interact with them has never been stronger. Much research effort has gone into the storage, retrieval, and conversion for these data, but little had been done to increase the value of the stored data. Very few interactive techniques have been designed for humans to interact with these data. As a result much of the data in the storage fall into the unfortunate category coined by Powsner and Tufte, "write-once, read-never" [86]. As the U.S. national health care reform promotes interchangeable electronic health record formats, it is prime time to design and test these new interaction techniques for search.

1.1 Overview of the Dissertation

The goal of the dissertation is to design effective interaction techniques for humans to perform searches in temporal categorical data. I focus on support for three main tasks: visual search, temporal pattern search, and higher-level information seeking.

Categorical data are data points that are not numerical. The most common example are names. Categories are sometimes ordered, such as *Monday*, *Tuesday*, *Wednesday*. However, more often than not, they are unordered, just like a bag

of words: *horse*, *cat*, *bird*, *mouse*. There had been much work in visualizing and searching temporal numerical data, such as stock prices over time. However, support for temporal categorical data had been mostly left unaddressed.

Initial collaboration with physicians in the Washington Hospital Center and Harvard Medical School in affiliation with Partners HealthCare revealed that there is such a need in the real world. Physicians and hospital administrators interested in better understanding their data are often frustrated in searching for patients by temporal characteristics (as opposed to patient characteristics). Even state-of-the art systems in leading medical institutions do not have ideal support. While I have worked closely with real data and experts in the medical field, and my techniques are inspired by their needs, the applicability of these techniques are not restricted to medicine. These techniques are implemented in Lifelines2 (Figure 1.1), and I demonstrate the effectiveness of the techniques in a number of case studies beyond the medical domain.

The contributions of this dissertation are listed below.

1.2 Contributions

1. The design and implementation of the novel *Alignment-Rank-Filter (ARF)* framework to support browsing and searching temporal categorical records. The idea is to rearrange the time line by "important events", rearrange records by event occurrence frequency, and filter by data characteristics. The dynamic, data-centric alignment is evaluated to confirm our hypothesis on the effective-

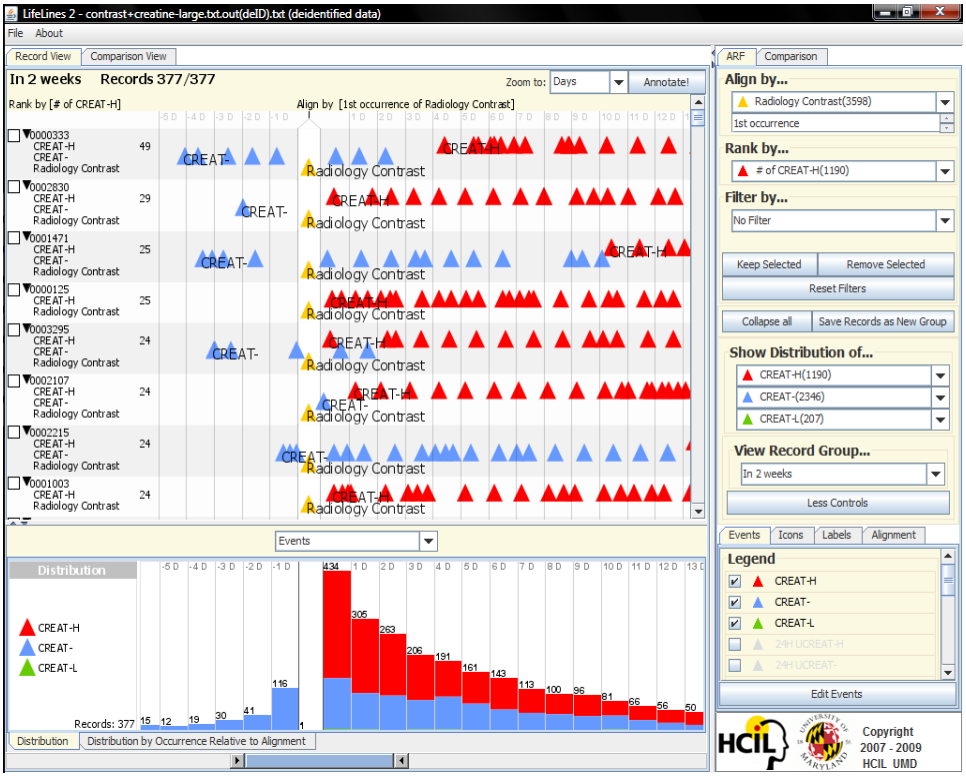

Figure 1.1: A screen shot of Lifelines showing a set of medical records on the top, and aggregation of medical events at the bottom.

ness of alignment in relative comparison tasks.

2. An algorithm designed for searching for temporal patterns in temporal categorical records. The idea is based on two main points. One, by separating each category in a separate array, a search can ignore many events not mentioned in it. Secondly, by using a sorted data structure, efficient binary search can be used. The internal data structure of Lifelines2 exhibits both of these characteristics, and an algorithm is designed to exploit them. A comparison of running time to common approaches shows its effectiveness and I discuss its extensibility.

3. The design and implementation of Temporal Summaries, Grouping, and Comparison to support higher-level tasks that involve managing multiple records of temporal categorical data. I show the usefulness and generalizability of these features through a number of case studies in several real world domains. As a product of experimenting with the variety of designs, I also present Lifelines2, as open source software that implemented all the contributions presented here.

4. By making observations of the case studies, analyzing logged Lifelines2 usage data, and through interviews with and commentaries of my collaborators, I generalized the case studies into a process model. It describes the steps analysts take in their exploratory process, and their behaviors. This process model serves as a guideline to future visualization designers, and I make several recommendations as a result. They outline tasks that are imperative to support, and describe possible design pitfalls.

1.3 Dissertation Organization

The dissertation is organized in the following chapters. I discuss background and related work first, and present each of the four main contributions in subsequent chapters. I generalize the case studies and present a process model for performing search in temporal categorical records. Finally, I give concluding remarks.

1. Chapter 1: Introduction

2. Chapter 2: Background and Related Work

3. Chapter 3: Supporting Visual Comparisons: Alignment and Intervals of Validity

4. Chapter 4: Supporting Temporal Pattern Search: An Algorithm

5. Chapter 5: Supporting Hypothesis Generation: Temporal Summaries, Grouping, and Comparison

6. Chapter 6: Solving Real Problems: Case Study Details

7. Chapter 7: Conclusions

Chapter 2

Background and Related Work

The intrinsic role time plays in our existence makes it an important perspective on how we view the world, and interpret its data. We humans often organize information temporally, whether it is in our minds, or in our digital devices. We reason about our physical world temporally. The fact that time is not reversible (currently!) makes it ideal to study causal links – causes always occur before effects. As a result, we often think of time as a linear entity, and we project time using a linear representation. However, depending on the tasks, other representations may be more appropriate. In this chapter, I present first, the designs of temporal data visualization systems, and, second, strategies of searching for temporal data as the two main bodies of related work. I break down each body of work into finer segments for detailed discussion. The background and related work discussed below reveal opportunities for improving search strategies for temporal categorical data.

2.1 Temporal Data Visualization

2.1.1 Temporal Categorical Data

2.1.1.1 Classical Designs

Most work in interactive visualization of temporal categorical data have predominantly utilized a linear timeline. This general technique has been applied to a variety of domains: authoring and management of video annotation [32, 38, 21], personal histories [84, 54], management of digital documents and alternative desktop management [63, 90], and analyses of user actions [57, 29], just to list a few. Timelines are used with the intention of allowing analysts to grasp when one event occurs in relation to others. This allows analysts to better understand the complex system in question.

For example, parallel programs are notoriously difficult to understand, debug, and optimize. In the late 1980's and early 1990's, a large number of research projects on debugging, performance profiling, and monitoring of parallel programs utilized timelines to visualize program execution [1, 33, 35, 60, 66, 105, 109]. These systems visualize the states of threads or processes. They focus on when processes are running, when they are switched out, and how messages pass among the threads. The systems enable analysts to identify anomalies and performance bottlenecks in a single parallel execution. Figure 2.1 shows one example from Harter *et al.*'s Interactive Distributed Debugger (IDD) system. IDD displays the message passing history among the different processes over the program's execution. Although it is

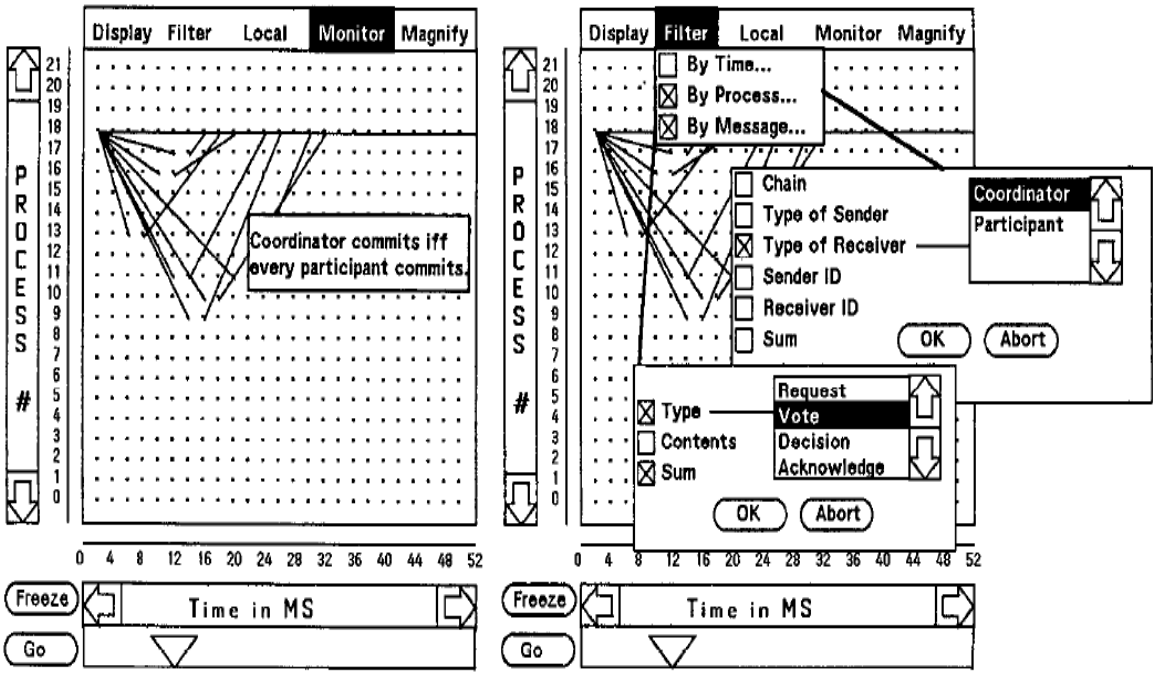

Figure 2.1: Two screen shots of the IDD system from Harter et al. [33]. All processes are listed on the y-axis, and time is on the x-axis. In the left screen, process 18 is focused, and the lines connecting the dots represent messages passed among the different processes. Analysts can select which process or time points to focus on. The right screen shows additional interactions analysts can perform: filtering by time, process, message and their attributes. A "Magnify" command allows for zooming.

an early system, IDD exhibits many design elements that persist in many modern systems today: First, the temporal dimension is laid out horizontally. Second, the different parts of the system (processes) are laid out vertically, and, third, the important events (messages) are represented as markers on the timeline. Some limited filtering and zooming features are available.

The system xtg [50] generalizes the designs of many of these parallel program analysis systems and allows user configuration to reproduce the visualization of these previous works. Symbols, vertical lines, and line elevations are used to

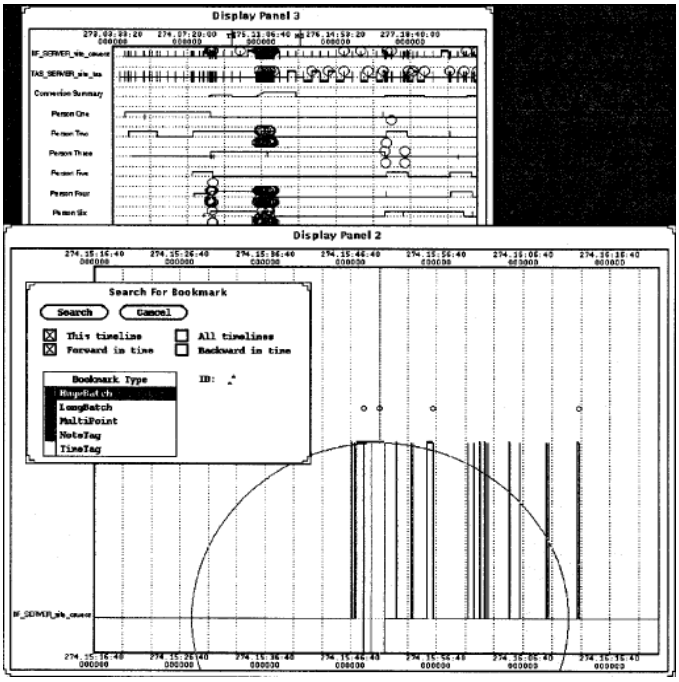

Figure 2.2: The xtg visualization system generalizes previous designs in visualizing parallel computation. In addition, it allows multiple view windows of the same temporal data. The search interface allows searching of events and bookmarks.

visually encode changes in the process states. In addition, analysts can create multiple, zoomable views to gain both an overview and detail or to perform comparison between executions (Figure 2.2).

The Time Line Browser was one of the earliest works in visualizing medical data [17, 18]. It also adopts a linear timeline to incorporate both event durations and values of medical tests, such as blood sugar reading. It is one of the first systems to mix both numeric time series data with duration events. However, up to this point, no evaluations has been performed to validate the value of timeline visualizations. Researchers have been relying on intuition that the timeline representation improves

human understanding of the temporal events. A landmark study on the advantage of the graphical representation of time is included as part of the study for the next system.

Plaisant *et al.* further extended the timeline designs in LifeLines [83, 84] (Figure 2.3). Duration events are color- and thickness-encoded on a timeline to represent categories and significance, so "high blood pressure" and "low blood pressure" can be distinguished. Events of similar categories can be grouped into facets. Analysts can collapse or expand different "facets" of the data. For example, a facet may be diagnoses, and another may be medications. Semantic zooming is incorporated in the visualization, so as analysts zoom in, for example, previously grouped "cardiovascular problems" may expand to "heart attack" and "hypertension". Mousing over events shows the details, and users can see supporting medical documents (such as a doctor's note or an x-ray image) on-demand.

An important contribution of the LifeLines study is an evaluation of graphical display of temporal data. The authors performed a controlled study comparing graphical layout of LifeLines against tabular display of the same personal history data. The authors concluded that participants can better temporally compare events and their durations on a graphical timeline display such as the ones in LifeLines [5]. This evaluation validated what visualization designers have long suspected: timeline-based visualization can help analysts understand the temporal dimension better. Many later approaches have since leveraged on the same visual designs of LifeLines, including Bade *et al.*'s MidGaard [9], Kumar *et al.*'s tmViewer [56], and Data Montage [20], a commercial product.

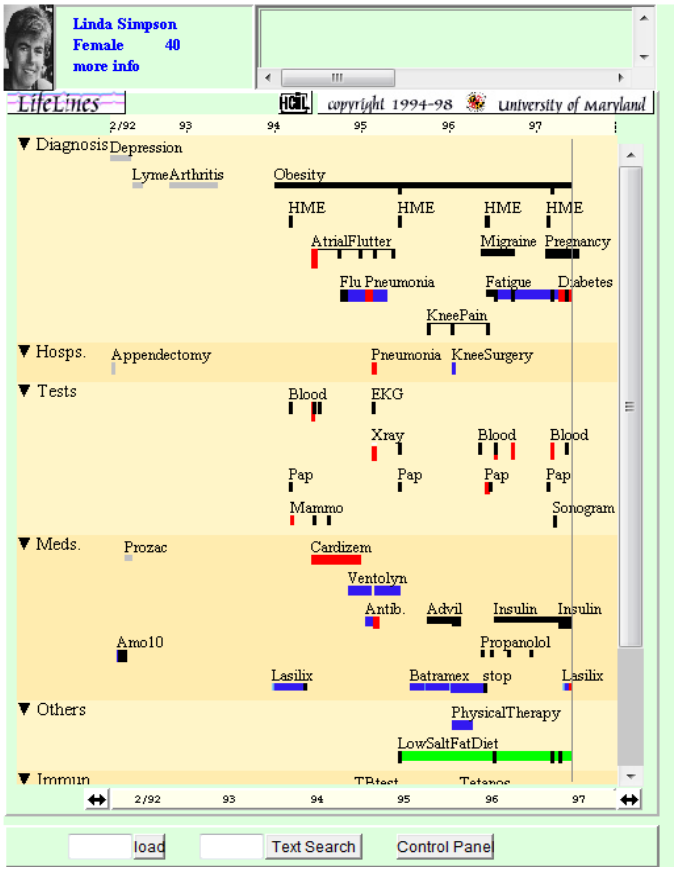

Figure 2.3: The screen shot of LifeLines shows a variety of events in a patient's medical record. Extending from previous research, LifeLines supports rich color encoding, semantic zoom, brushing, and collapseable facets.

2.1.1.2 Modern Innovations

Modern visualization research for temporal categorical data have focused on innovating different aspects of the classical designs. One extension is to incorporate more complex categorical hierarchies. Another is to focus on the explicit relationship among temporal events. However, they have largely ignored the importance of providing novel and effective search strategies.

A number of more modern research efforts have focused on extending the

small hierarchy of event types in LifeLines to incorporate deeper, richer hierarchies. These efforts are motivated by complex domains where taxonomy of domain jargon is readily available, for example, the medical field. Systems from Shahar *et al.* [96, 52], Post *et al.* [85], and Murphy *et al.* [70] deal with medical terminology by using standard or custom-built medical ontologies. Users can use the ontologies to search and navigate patient records. However, these systems typically treat temporal events as if they were attributes – whether a patient has been diagnosed with "asthma" and "pneumonia" as opposed to temporally ordered sequences such as "asthma" followed by "pneumonia".

While the organization of query terms enhances the usability of a search system, other modern visualization systems have instead focused on the known, explicit relationships among events. One example is historians using timelines to represent historical events and their connections in order to facilitate and enrich the narrative. The focus of these approaches is on a single interesting event (*e.g.*, Kennedy assassination) and its related events (those that lead up to the interesting event, and those that follow). This gives historians a way to convey the temporal relations of the events as well as a way to decompose a complex event into smaller events for analysis. The SIMILE timeline tool from MIT is one such example [88]. It relies on subtle visual cues to denote relationships of events by using color encodings.

Jensen's SemTime, by contrast, allows multiple explicit relationships among events (such as article citations) to be visualized as directed lines on the timeline [45]. To avoid obscuring the line paths, SemTime lets analysts truncate a linear timeline into stacked "time segments" of possibly different granularities (Figure 2.4 (left)).

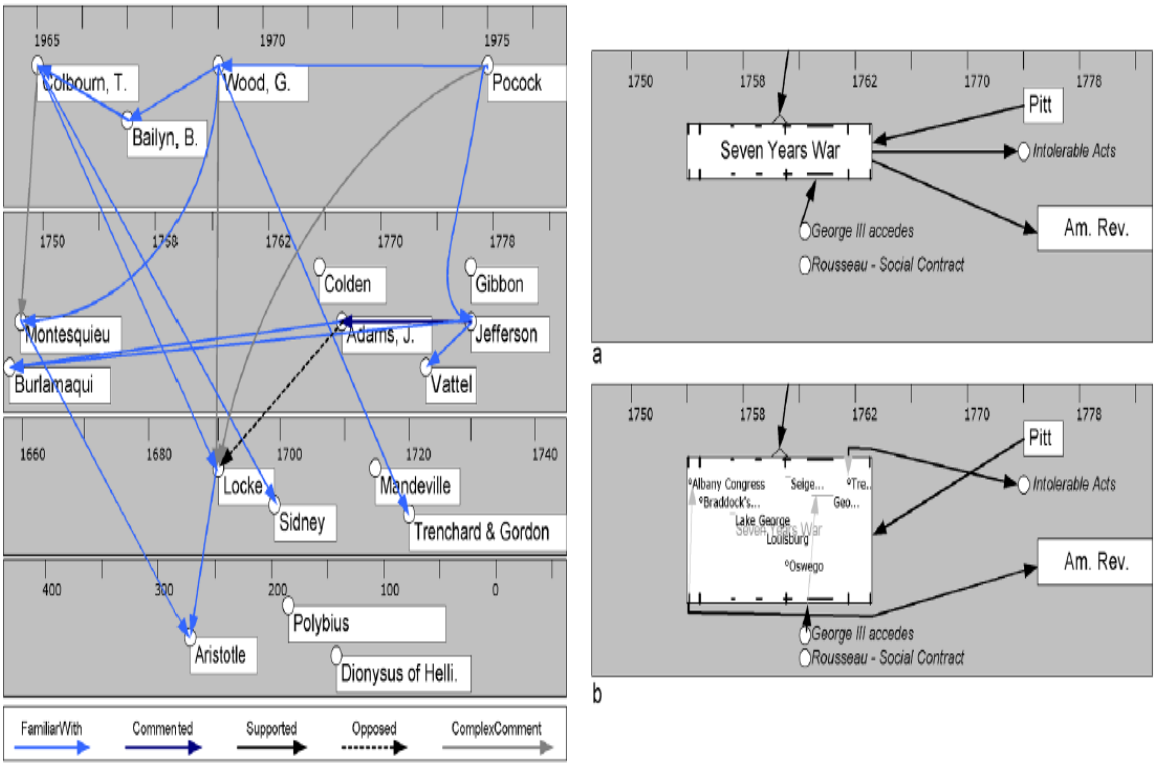

Figure 2.4: The two innovations of SemTime [45]. The left screen shows the relationships among events on a segmented timeline. The screen shots on the right show the expansion of a "sub-timeline" in-place.

This allows analysts to mitigate the overlapping of edges, as many arrows can now travel vertically. This idea effectively transforms a timeline visualization to a graph visualization that resembles the semantic substrate advocated by Shneiderman and Aris [98], where each substrate is a timeline segment. A second innovation is that SemTime allows a duration event to have its own "sub-timeline", which contains all "sub-events" specific to that event (such as battles in a war), and allows expansion of the event details in-place (Figure 2.4 (right)).

André *et al.*'s Continuum also makes use of the "sub-timeline" idea to organize hierarchical information on a timeline (Figure 2.5). However, Continuum's data

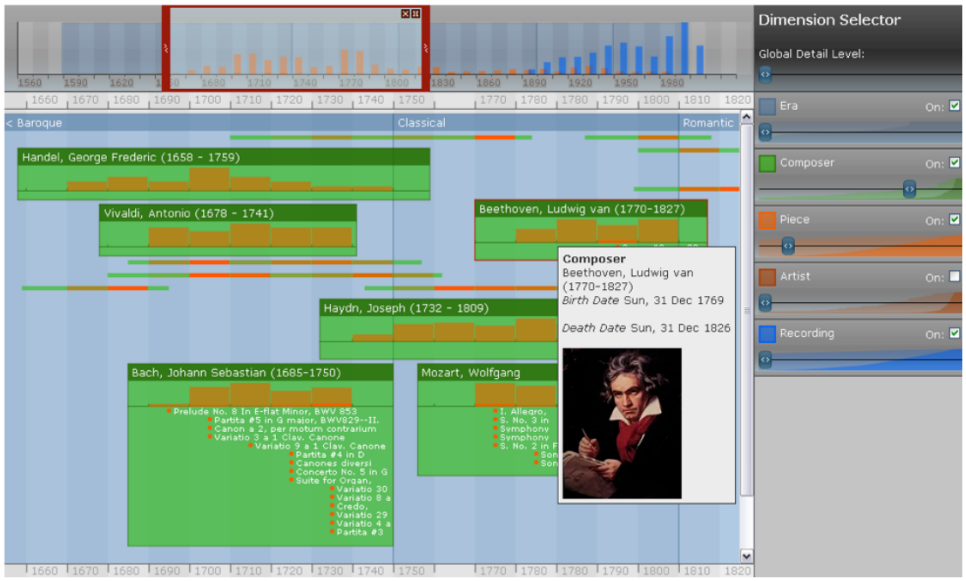

Figure 2.5: Continuum: temporal information to be organized by a dynamically-defined hierarchy [6].

structure is facet-based, and allows analysts to associate each facet to create an appropriate hierarchy for exploration. For example, in the domain of classical music composers, pieces, and recordings, an analyst can dynamically create the hierarchy `era -> composer -> piece` or `era -> piece` to suit a particular exploratory need. This dynamic hierarchy approach can be seen as a generalization of the multidimensional attribute browser FilmFinder [46], which plots a Starfield on time axis.

2.1.1.3 Multiple Record Analysis

Most of the previous work presented so far are for analyzing temporal data for a single entity, *i.e.,* single patient, single parallel program. There are also work on visualizing temporal data on categorical events over multiple records. Lexis diagrams

are an early example that is not interactive [61]. Demographers use them its more modern variations [87] to visualize multiple life histories at simultaneously. These approaches uses a Cartesian space, but both the x-axis and the y-axis are time. One would represent the calendar, and the other would be a time scale relative to a common, significant event. For example, the important event may be the start of pregnancy. Using these Lexis approaches, analysts can quickly find how many women are pregnant at time t_1 using the calendar time axis. Using the other axis, the analysts can quickly identify those who have been pregnant for more than x months. The representation of the personal record is simple. A line or a narrow strip usually suffices. These Lexis approaches are very useful when representing change of personal statuses in a population. Lexis pencil [27] takes the idea further to 3 dimensions, and displays different facets of a history on faces of a 3-D pencil to increase information density and allow for richer interaction.

These Lexis approaches use an important, but often overlooked strategy to deal with multiple records. The records are all aligned by the start of the significant event in their own respective timeline. The alignment allows for quick visual comparison among the individual records. However, this alignment is not dynamic, and users can only see the time progression *after* the alignment (no visualization of the statuses before the alignment). The idea of alignment is also adopted in other modern visualization tools. Experiscope [29] and Session Viewer [57] both have facilities to let analysts pick multiple records and align by a certain point in the record. Analysts can pick arbitrary points in Experiscope, while in SessionViewer, the time point selection is constrained by the presence of events. Analysts can only pick time

points where an event exists. The manual selection of alignment point provides great freedom to analysts to build their preferred view, but it is slow and does not scale to more than a handful of records.

There are yet other approaches that use the periodicity of time as points alignment to allow discovery of cyclic patterns in temporal data [118, 12]. For example, if every week represents one cycle, all the same days of the week in a month would be aligned (similar to a calendar). So if there are weekly cyclic patterns, the alignment can make them visually compelling. However, this kind of alignment is based on calendar designation, and not based on the data, and only cyclical patterns in data that is based on the calendar designation would be visually prominent.

One of the goals of my dissertation is to generalize these different alignment approaches. I propose and evaluate an alignment design that is dynamic, automatic, and data-centric to overcome the problems in these earlier systems (Chapter 3).

2.1.1.4 Aggregating Temporal Categorical Data

An aspect of dealing with temporal categorical data is to aggregate them in a meaningful way. The common approach is to aggregate them by type, and visualize the resultant numerical temporal data. Several algorithms and techniques have been proposed to efficiently aggregate temporal categorical data. Dubinko *et al.* propose an algorithm to rapidly aggregate photos tagged in different temporal granularities to create compelling photo slide shows [23]. Ribler *et al.* proposed several aggregation techniques for temporal categorical data, but mostly only focusing on temporal

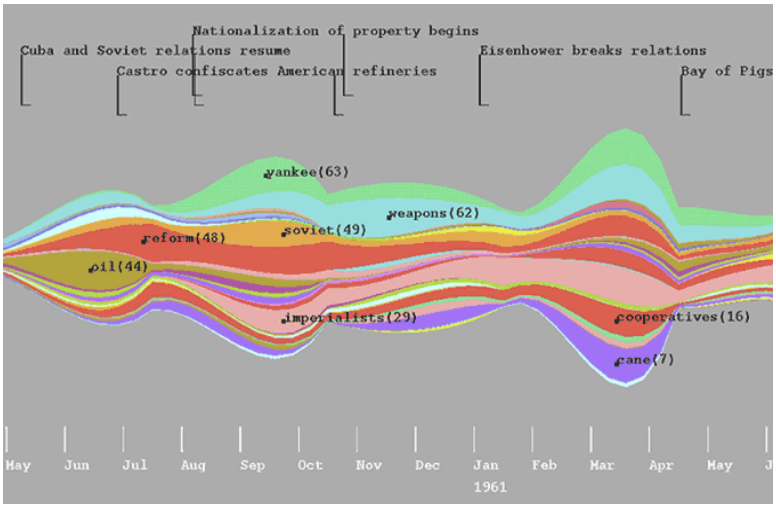

Figure 2.6: Screen shot of ThemeRiver. Categorical data (key words in news articles) are aggregated and displayed in an aesthetic way to highlight the rise and fall of news events.

periodicity [91]. Aggregation of temporal categorical events into numerical values in visualization systems is fairly common.

One often-cited example of aggregating categorical events into numerical data is ThemeRiver [34] (Figure 2.6). It visualizes keywords mentioned in news articles to track the rise and fall of importance of news events over time. Each keyword is represented as a color, and the thickness denotes its usage frequency. Many keywords can be tracked simultaneously. Analysts can easily tell how news topics rise and fall over time. However, ThemeRiver provides no facility to allow search.

Heer *et al.*'s sense.us and Wattenberg's Baby Name Voyager [117] use a similar strategy – animated stacked bar charts to visualize numerical data over time. Baby Name Voyager allows analysts to filter for specific subsets of baby names, thus adding an interactive search component. This kind of aggregation can be found in a variety

of domain applications. Phan *et al.* use bar charts to aggregate network event data to analyze network intrusions [80]. In traffic analysis, accidents can be aggregated by type and displayed in a temporal-geographical visualization in applications such as [28]. In all of these applications, however, the visualization of the numerical trends over time was the end goal. These applications do not support search functionalities or higher exploratory tasks. I present the design of *temporal summaries* (Chapter 5) and show they provide support for selecting temporal range filters relative to an alignment, and how the comparison of these summaries can be a useful feature.

2.1.1.5 Temporal Numerical Data

Although numerical time series visualizations fundamentally support very different tasks, many existing work present multiple time series in coordination. Among others, [86] and [25] present multiple numerical medical data of a single patient in the same view to facilitate visual tracking of a patient‘s condition, but offer none or very limited interactions. These approaches, however, do not have any search capabilities.

Innovative search methods may be hard to come by for temporal categorical data, but it is not the case for temporal numerical data. In an effort to search and query for patterns across many time series data, Hochheiser and Shneiderman's TimeSearcher [41] overlays multiple single-variable numeric time-series in one overview display, and provide novel query methods. The direct-manipulation interface allows users to focus on interesting points, and filter out uninteresting variables.

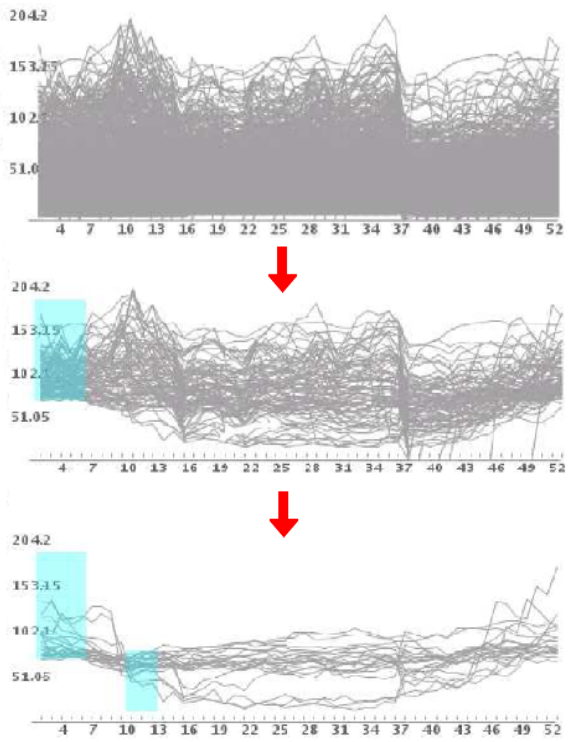

Figure 2.7: Composite screen shot of three screen shots of TimeSearcher's TimeBox query in action. By creating additional boxes in the space where multiple time series data is displayed, only the time series that intersect with the TimeBoxes remain. The others are filtered out.

Furthermore, novel features such as TimeBox (Figure 2.7) and Angular Query to specify the rise, fall, slope, and range of patterns in the visualization was advocated to alleviate the problem of constructing complex queries. Subsequent work on TimeSearcher2 expanded the earlier research by handling (1) unevenly-spaced time series data, (2) scaling to 10,000+ time points, (3) dealing with multiple variables, and (4) providing mechanisms for flexible pattern search [8]. The approach of visualizing multiple time series in the same space is adopted in many other systems, such as [81] from Pieczkiewicz *et al.*

Instead of overlaying the different numerical time series on the same 2-D space, Interactive Parallel Bar Charts (IPBC) uses 3 dimensions: the first for each time series, the second for the time dimension, and the third for the values of time series [14]. Analysts can select a portion of a time series to invoke a similarity search which returns segments in the visualization that is similar to the selected pattern. Analysts can also create a linear function that describes the behavior of a time series. IPBC can then use the function to find parts of the many time series that exceed the function.

Unlike TimeSearcher, VizTree [62] is a tool where pattern-finding is built-in from the internal data structure to the external representation. The range of a time-series is reduced to n bins. Each bin represents a symbol. A suffix tree of branching factor of n is built to represent the entire time series. Each branch corresponds to a symbol, and each path represents a string of symbols. The thickness of a path represents the frequency of that string in the time-series, hence the frequency of the pattern in the time series. The authors showed the effectiveness of this representation for matching patterns, detecting anomalies, and spotting motifs, and argued that it is a scalable solution for massive time series. Also unlike TimeSearcher, VizTree focused on recurring patterns from one single time series, as opposed to across many time series. It is worth noting that VizTree turns numerical time series into ordinal data (bins), which dictates a specific ordering, differing from unordered categorical data. Furthermore, the size of each discrete step in binning has dramatic effect on the ordinal sequence representation of the time series.

Shahar *et al.* present a temporal abstraction framework [96, 95, 52] to combine

the patterns of numerical time series to build higher-level abstractions (temporal abstractions) to assist decision making. While most of the applications have been in the medical domain, it has also been applied to network security data [95]. For example, by combining the values platelet counts, bone marrow toxicity readings and other numerical readings, these systems can identify different stages (ordinal data) of transplantation problems in bone marrow transplant patients. While the automatic detection and creation of higher level temporal abstractions (categorical data) is useful, these systems do not provide search mechanism on the abstractions. Other temporal abstraction approaches such as PROTEMPA have commented that the lack of search mechanism combined with the lack of effective representation of detailed patient data makes discovery of novel patterns difficult [85].

2.1.2 Other Representation of Time

Several works have used alternative representation of time to support specific tasks. Weber *et al.* [118] and Carlis *et al.* [12] showed that by using spirals marked with time intervals (weeks, months, quarters, or years), periodicity of data can be easily noticed. Hewagamage *et al.* propose a way to visualize events on a spiral timeline in both 2D and 3D space, where events are represented as icons [37]. In Weber's tool, users can vary the length of the cycle and explore time series data to find repeated patterns. However, when data do not exhibit a cyclic nature (or that if the cyclicity is not in accordance with the temporal cycles), the temporal patterns can still be very difficult to spot. EventTunnel [106] uses a circle to place

temporal events, however, it does not align the temporal cycles on the same wedge of the circle. Instead, it uses the wedges of the circle to organize events that share the same attributes.

Temporal-spatio visualizations such as TimeMap [47], Pattern Browser [37], and (to an extent) COPLINK [15] allow individual events to be displayed both in geographical dimension and temporal dimension (through traversable animation). These visualization approaches are focused on enriching the historical narrative experience of the users on a single interesting event (*e.g.*, one timeline for a crime), and not finding patterns across many histories (*e.g.*, multiple timelines of similar crimes). The system Gravi++ uses a force-directed layout to represent patients and their answers to psychological surveys [40]. A group of patients can be simultaneously visualized. Similar answers tend to cluster and become visually prominent. By stepping through time via animation, the change of patient survey answers can be tracked. Similar to animation, small multiples have also been used to visualize multiple time series simultaneously [42].

2.2 Searching for Temporally-Ordered Data

There are two main characteristics in temporal data for search. The first is the ordering of temporal events, and the second is the duration between any two temporal events. There have been many proposed temporal query languages [102, 103, 16] and extensions of SQL to specific domains such as medicine [76, 49, 75]. Unfortunately these command line languages suffer from the same critical problems

SQL does (see Appendix B for a detailed example)). In this section, I present the more novel approaches to access temporal data. While these approaches are innovative, there are more left to be desired with respect to searching for the two characteristics of temporal data.

2.2.1 Sequential Approaches

Searching for temporal patterns is an important task for temporal data analysis. Unfortunately, many temporal visual analysis tools do not support such temporal pattern search [20, 24, 106, 57]. State-of-the-art production tools often omit this functionality in favor of simplicity, learnability, and familiarity [68, 70]. In making the visual analysis tool effective in its visual and interactive operations, often the data structure is indexed to suit those purposes. However, these indices may not be suitable for searching for temporal patterns. Temporal categorical data present its own constraints on how data needs to be represented faithfully. Many of the algorithms presented below rely on pre-processing of the pattern to ensure efficient run time.

While the preprocessing is generally light ($O(m)$, where m is the length of pattern) in approaches like nondeterministic finite automaton (NFA) and bit-parallelism approaches, it would be better to avoid any pre-processing. It begs the question that when data are read into an analysis system, could they be primed or otherwise indexed so that subsequent temporal pattern queries can be performed quickly? Chapter 4 presents an algorithm that is designed to overcome these constraints. However,

something would be amiss if I do not discuss the plethora of related topics.

The most obvious topic one is string matching. At a first glance, string matching is very similar to matching events over time – both share a sense of progression (event in time or character in position),both are concerned with finding ordered sequences of items. However, there are some fundamental differences. In classical string matching,(1) a pattern is wholly specified ("abc" as opposed to, in a temporal pattern search scenario, "a" followed by "b" followed by "c"), (2) no two characters can occur at the same position in a string, and, finally, (3) that negations are usually not considered. Nevertheless, it would be useful to examine these clever algorithms to examine these fast classical approaches to see they could be generalized for the purpose of temporal pattern search: Knuth-Morris-Pratt (KMP) [53], Boyer-Moore (BM) [11], and Rabin-Karp (RK) [51]. KMP pre-processes a given pattern to build a prefix table that tells the algorithm when and how far ahead (in number of characters) it can skip, avoiding unnecessary scans. Likewise, BM builds two tables based on the pattern which tells the algorithm how much to safely skip ahead. Finally, RK utilize the fact that a string can be viewed as a number, where each character is represented by a digit (in radix = size of the alphabet) and modulo equivalences to speed up checking for matches.

These string matching algorithms all exploit the fact that a string is contiguous. Every character (except for the first one and the last one) has only one predecessor and a successor, and the characters are indexable. This information, along with the known pattern, can be used to reduce the number of scans in the text. Sadri *et al.* use this fact to optimize sequential searches in SQL-TS over

data streams by using a variant of KMP [92]. The drawback is, or course, that the text and the pattern must be contiguous, limiting the expressiveness of their search patterns. In temporal data, events are not contiguous (so the pre-processing in the classical string match approaches cannot apply). Some events can occur at the same time. Finally, when searching for temporal patterns, both occurrence of events and the non-occurrence of events are important along with wild cards, optional, and repeatable events.

A more general way to deal with data that needs to be processed one-by-one is to utilizes either deterministic or non-deterministic finite automaton (DFA or NFA) [108]. These automaton-based approaches are more flexible than the classical string matching algorithms because they can handle regular useful operators such as the Kleene Closure (*), wild cards (.), negations(ˆ), and classes of characters, to name a few. Given a text of length n and a pattern of length m, a DFA can perform a search in $O(m)$, but building a DFA can take up to $O(2^m)$ space [89]. This makes the DFA approach useful for applications where building a DFA is infrequent or can be done off-line, such as network intrusion detection systems [121, 55, 26]. On the other hand, an NFA can compactly represent an equivalent automaton in space $O(m)$, but the search time may take up to $O(mn)$ to perform a search.

Given the trade-off between DFAs and NFAs, many systems choose to use NFA or extensions of NFA for their stream processing needs [2, 22]. These approaches tend to have an expressive pattern language where negations, Kleene Closures, and temporal constraints can be included. They are more expressive than regular expression. These systems are geared towards fast processing over continuous event

streams, where an event can be more complex, and contains additional attributes. The domain of data this dissertation focuses on is a simpler problem where events do not have these additional attributes, and thus allows the design for a simpler algorithm for searching for temporal patterns. For example, in [2], searching with negation of events is supported by first finding all positive events and then pruning off the results that contain negation events in the wrong temporal ordering. In contrast, my algorithm searches for negations in-place (Chapter 4).

Newer string match approaches such as the Shift-And, Shift-Or, or the Backward Nondeterministic Dawg Matching algorithm can be more easily adapted to deal with classes of characters, optional and repeatable characters that classical string match algorithms cannot handle [72, 73, 71]. The basic idea behind these algorithms is to use bits to represent the states of an NFA. When a symbol is read, all states can be updated in parallel by cleverly using bit-wise operators (and, or, shift, xor, etc). When there are w bits in a computer word, these algorithms are expected to perform w times faster than an equivalent NFA. Today when most consumer machines are either 32 or 64 bits, this can be a significant performance advantage. When more than w states are required to represent the pattern, multiple words can be used. In this case, the performance of these algorithms suffers because of the overhead of using an array of integers instead of a single integer to represent the states, and the best worst case time for Shift-And becomes $O(mn/w)$. However, unlike real NFA or DFA approaches, these algorithms are difficult to extend to handle additional data constraints. Finally, both the automaton and the bit-parallel approaches assume the inputs to be a single string, and cannot be directly applied to temporal categorical

records.

There are also some more ad-hoc temporal pattern search algorithms designed for specific purposes. For example, Harada *et al.*, assume a grouping over a column of data (*e.g.*, customer ID) and an ordering by a second column (*e.g.*, time stamp) and searches for temporal patterns (by the ordering column) within each group (in a customer's history) [30, 31]. They do not use an NFA approach to perform the search, instead, an algorithm that resembles building a topological graph is used. The expressiveness of their language allows the specification of only limited negation. For example, let a, b, c, d be event types, their approach can define a pattern that have the same semantics as the regular expressions $.*a.*b.*$, $.*a.*[\hat{}bd].*c.*$, but not $.*a[\hat{}bd]*c.*$, which is required for properly support negated events. The limited expressiveness also means that their algorithm never has to backtrack. There was no reported run time analysis for their algorithm.

Finally, Agrawal *et al.* propose a Shape Definition Language, aimed to describe the shapes formed by the values of a single numeric variable over time [3]. Users can use the simple definition language to specify increasing, decreasing patterns for search. The language is as expressive as regular expressions, but can discard non-maximal subsequences to reduce clutter. Though it is designed for time series data, the query method itself is categorical, and the capability to discard non-maximal subsequences is useful for finding "broken" data such as the event of checking into an emergency room without having been released first. This search approach is akin to the that of VizTree, and share similar weaknesses. Temporal data can not be generalized as a single sequence for search. Point events can occur at the same time,

and duration events can overlap.

2.2.2 Novel Query Interfaces

Tansel *et al.* present Time-by-Example, a "user-friendly" query language for historical relational databases [107]. The interface is akin to the Query By Example (QBE) language [122], but the language is relationally complete (for example, QBE cannot express nested sets [77]). The interface allows users to specify query constraints, including temporal constraints, in text. Although there were no formal studies, the authors note that their interfaces for the more advanced features of the query language are difficult for novice users.

Hibino and Rundensteiner compared the efficacy of a visual query interface and a form-based query interface for Temporal Visual Query Language (TVQL) [39]. The visual interface contains a set of range sliders to specify the temporal relationships among time intervals of interest. There are four sliders for each pair of intervals `(A, B)`, specifying the relationships between `(A.start, B.start)`, `(A.start, B.end)`, `(A.end, B.start)`, `(A.end, B.end)`. In addition to the sliders, a temporal diagram is dynamically generated as a feedback to the user. The controlled experiments showed, with statistical significance, that this visual interface is difficult to train, but allows participants to interpret the semantics of the queries and construct incremental queries more efficiently . While the results are promising, there are a few important points worth noticing. First, the number of sliders increases exponentially in the number of events. While every degree of freedom *can* be specified, it

is unclear whether they *need* to be in every case. Next, the visual representation of the intersection of intervals is not scalable to multiple, intersection intervals. Each of the range sliders has numeric values so the temporal displacement between two time points can be set. The numeric values do not have units. Furthermore, it is unclear whether it is possible to change units for multi-granularity query specification. Finally, the authors seem to have overlooked ways to specify absolute temporal constraints (*e.g.*, `event A` must happen before the absolute time `January 8, 2001`).

PatternFinder allows analysts to specify temporally ordered sequences in mixed temporal and categorical data sets [24] (Figure 2.8). It also presents the results as a chain-and-ball visualization to highlight all the matching events in personal records. PatternFinder is powerful, but the large number of widgets that allow analysts to specify the temporal ranges of events often overwhelmed analysts. The interface was later simplified and applied to the real system Amalga in a subsequent project [58], where relative comparison is built into the query interface. These approaches, however, do not allow the specification of the *absence of* events, which are important in many real world applications. For example, clinical researchers may want to find patients who have the sequence of events "with no prior history of heart problem, later diagnosed with hypertension, received no treatments, and finally experienced a heart attack". Without the absence operator, analysts cannot specify the lack of heart problem events and treatment events in the middle of the sequence.

HCI designers have also designed visualization and interaction strategies that guide analysts to search and filter. Multidimensional data that contain time stamps

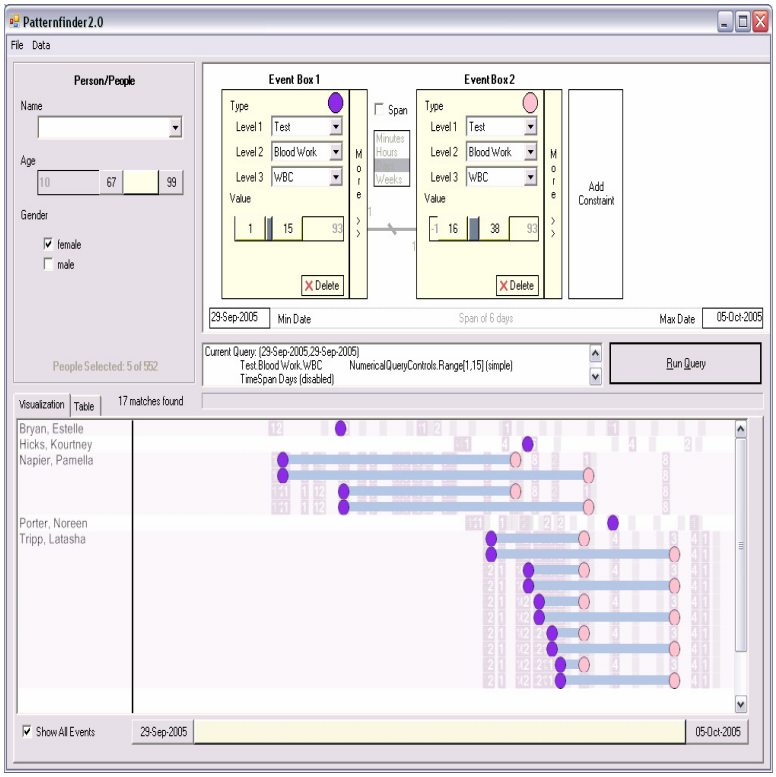

Figure 2.8: A screen shot of PatternFinder, its form-based query interface on top, and ball-and-chain result visualization at the bottom

can use interactive, direct-manipulation, and dynamic query techniques [4, 97, 110] to filter for data points that are within a temporal range [46, 119, 104, 6]. There have been fewer visual techniques for searching for temporal sequences. TimeSearcher [41, 8], VizTree [62], and SessionViewer [57] are rare examples.

More recently, ActiviTree [113] automatically aggregates personal activities over a population and uses a page-rank-like algorithm to suggest likely, and possible routes of exploration. One can think of ActiviTree as an aggregation of Lexis diagrams [61] in that it works for personal statuses well, but cannot handle simultaneous or overlapping events in general. Another interesting advance is searching

for similar sequences. Mannila *et al.* propose several ways to compute similarity among event sequences [65, 64]. Similarly (pun intended), Wongsuphasawat *et al.*'s interactive visualization Similan uses the idea of "edit distance" to compute a similarity measure for event sequences that works for relative time scales. Analysts can dynamically adjust weights on a variety of factors to fine-tune the way similarity measure is computed.

Even though these related work on search interfaces have some shortcomings, state-of-the art systems medical record systems have lagged even further. Real production systems have often sacrificed the functionality in favor of simplicity, learnability, and familiarity. As a result, queries on the temporal ordering of events are often sacrificed. For example, the i2b2 medical database allows only boolean queries to be specified in patient records from its interface by default [70] (Figure 2.9). Instead of "Pneumonia " after "Asthma", it can only handle "Pneumonia" and "Asthma". The Amalga system (formerly known as Azyxxi), our physician collaborators electronic health record system allows for canned search/filter options that allows value constraints, but not temporal constraints (Figure 2.10).

2.3 Summary

This related work section describes innovations in visualizing and searching temporal data. On the visualization side, many good existing designs can be leveraged. However, there also seems to be a niche for novel visual operators that support relative comparisons. In terms of search, on end of the one spectrum, non-visual,

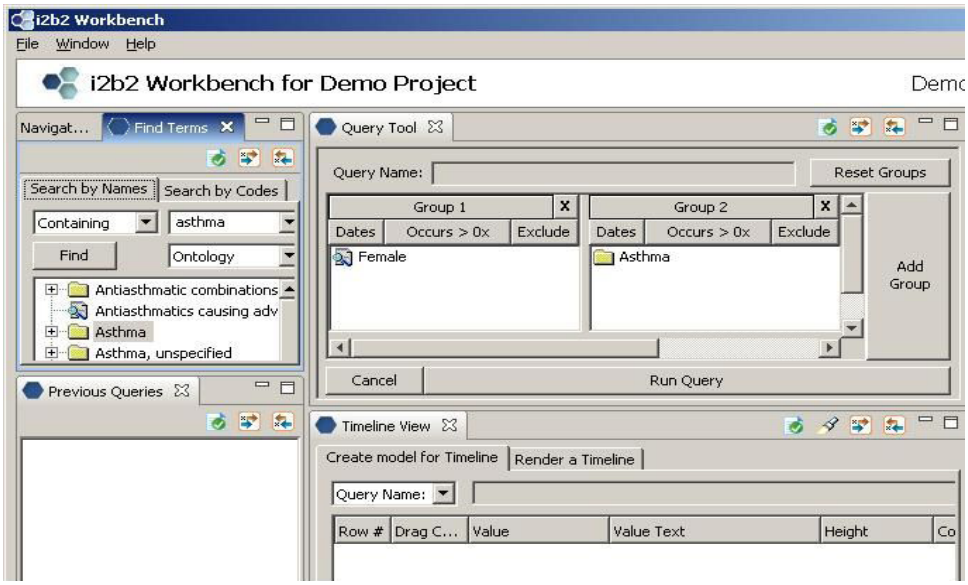

Figure 2.9: Partial screen shot of the i2b2 querying interface. The top right Query Tool panel allows Boolean queries over medical terms to be specified. A query is generated as a conjunctive normal form of the terms. Each Group in the interface is a list of OR's, and the groups of OR's are ANDed together.

command-line based languages are difficult to grasp, and adoption is limited to advanced users. On the other end of the spectrum, interfaces that allow complete specification of all constraints in a temporal search can easily overwhelm users. As a result, state-of-the-art interfaces shy away from providing the desired expressiveness required to perform temporal search of event patterns and favor simpler Boolean queries. This treatment of temporal data limits users' freedom to analyze it. These missing pieces suggest research opportunities, and I present my solutions to these problems in the following chapters.

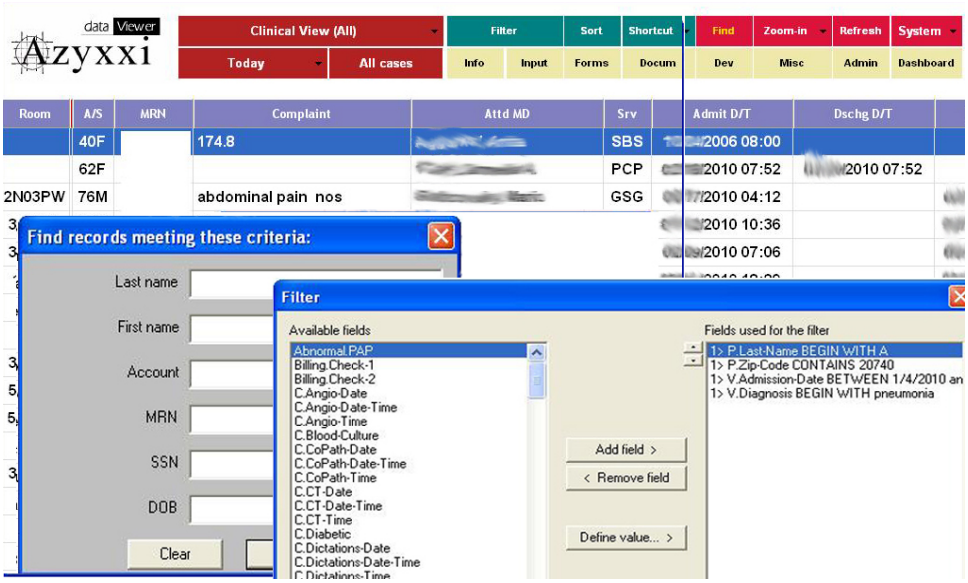

Figure 2.10: Composite screen shot of the query interface in Amalga (formerly known as Azyxxi). Analysts can search for patient records by values in cells as in a spreadsheet (left dialog box). Further filtering can be accomplished by restricting the values to be matched (right dialog box).

Chapter 3

Supporting Visual Comparisons: Alignment and Intervals of Validity

3.1 Overview

An appropriate visual arrangement of data can allow analysts to build a more easily maintainable mental model, or see patterns that otherwise would be difficult to discern. For example, in graph visualization literature, TreePlus by Lee *et al.* uses a tree to visually represent a graph [59]. It effectively reduces the high visual complexity typically present in a node-link diagram. As a result analysts can perform faster and with more reliability in some graph browsing tasks. A different technique by Aris *et al.* shows that by arranging the nodes in a legal decision citation network by *semantic substrates*, analysts can better detect prominent patterns (*e.g.*, lower court citing higher court decisions) and outliers (*e.g.*, higher court citing lower court decisions) [7]. Modern computation expands the power of visual graphics in that modern systems empower analysts to dynamically and iteratively rearrange the visualization to suit their purposes. The question then, is what is an appropriate visual arrangement for analyzing multiple temporal categorical data? And subsequently, can the arrangement be dynamic?

In this chapter, I describe *alignment*, a novel interaction technique that is suitable for browsing records of temporal categorical data. Alignment is implemented in the visualization tool *Lifelines2*. I also describe a controlled experiment that val-

idates the usefulness of alignment. Finally, I summarize the impact of my findings to the field of temporal categorical data visualization. The content of this chapter corresponds to one of my earlier publications [115].

3.2 Background and Motivation

Lifelines2 was initially motivated by our physician and clinician friends from Harvard Medical School and Partners HealthCare. They highlighted two scenarios where querying large databases of clinical data and reviewing results are not well addressed by currently available tools.

1. **Observational research using existing data (instead of a clinical trial)**. In this scenario, researchers use de-identified data collected for the purpose of other (existing) studies or medical practice data to better understand health problems or study the effect of treatments.

2. **Clinical trial patient recruitment**. Harvard Medical School and Partners HealthCare receives over 600 requests a year from researchers everywhere to find suitable participants for clinical trials. The query terms typically contain diagnoses, treatments, and chief complaints. Analysts need to make a decision on whether each of the patients is a possible candidate for the trial based on manual reviews of the results.

Among the many challenges posed by these two scenarios, are the issues of design and user interaction. First, temporal comparison among patients is challenging,

subsequently making finding patterns difficult. Secondly, even experts who know the underlying temporal data semantics and provenance can misinterpret the data presented. I illustrate these two challenges in the following motivating example:

Researchers who study asthma may be interested in the relationship between patients' first pneumonia and their asthma attacks and treatments. A query is issued with those two diagnoses. Researchers then review the frequency and the temporal placement of the asthma events in relation to the first pneumonia in a large set of patient records. The reference event (*i.e.*, the 1st occurrence of pneumonia) is called the *sentinel event*, and I use that term in the rest of the dissertation. Reviewing results on a timeline display is helpful but requires comparing events spread over, potentially, a large time span. In addition, pneumonia and asthma are often related, and diagnoses often occur very closely, researchers may spend a significant amount of time zooming and panning to switch between a global overview and a detailed inspection. The constant interaction with the display is disruptive to a researcher's visual memory, making discovery of patterns difficult. Proving the existence of a phenomenon will still require statistical analysis, but could interactive techniques assist analysts in the initial review of the data to find problems, perceive patterns and formulate hypotheses about possible phenomena?

A second problem reported by our colleagues is that even highly trained medical professionals can forget the implicit but uncertain duration of medical events when interpreting temporal events, making the review process error-prone. To reuse the asthma example, researchers might be looking for patients who have been given steroids for their asthma condition. Because clinical data do not encode what condi-

tion a drug is prescribed for, users have to rely on the data at hand to estimate why the drug was prescribed. Steroids are prescribed for asthma but also for rheumatoid arthritis or other long lasting medical conditions. Even though highly trained analysts should be well aware that rheumatism is a long-lasting condition, they are still susceptible to interpreting a rheumatism diagnosis, which appears as a point event, as a short-term condition, and make the wrong decision. Would showing the *interval of validity* of a diagnosis be helpful to remind users of the likely duration of the condition?

3.3 Lifelines2

Lifelines2 is a prototype visualization system that focuses on supporting analysis across multiple temporal categorical records. The original Lifelines was designed to summarize the entirety of a single personal history record (*e.g.*, a medical record). In contrast, Lifelines2 displays multiple records. It is a platform in which I develop and experiment with interaction and visualization techniques. The first two issues I experimented with were *alignment* and *intervals of validity*. Here I give a description of the interface, and describe the design of the two features.

3.3.1 Interface Description

In Lifelines2, Each record is vertically stacked on alternating background color and identified by its ID on the left (Figure 3.1). Asthma and pneumonia diagnosis events appear as color-coded triangle icons on a horizontal timeline. By default all

records share the same absolute time scale (with the appropriate temporal granularity such as years or month labels displayed at the top) and the display is fitted so that the entire date/time range fits in the screen. As in Lifelines, zooming on the horizontal axis and panning is possible. Tool tips provide details, and records can be collapsed (one by one or all at a time) into a compact silhouette using smaller icons and less space. Left-clicking onto the visualization centers and zooms in. Right-clicking zooms out. Any click onto the record ID area resets the display to the initial, fitted overview. In Figure 3.1, it is easy to see the co-occurrence of *Pneumonia and influenza* events and *Asthma* events . However, it is not clear whether *Asthma* occurred before or after each patient's first *Pneumonia and influenza*. Users are forced to zoom in to each patient's first occurrence of *Pneumonia and influenza* for details, but each zoom can only reveal the details around a particular *Pneumonia and influenza* event on the absolute timeline.

On the right side a control panel provides access to align, rank, and filter the display. Menus are data-driven. Users can choose any event category to align all of the records. For example, Figure 3.2 shows setting the *1st pneumonia-or-influenza* as the *sentinel event*, and all records are aligned on a vertical line by that event. It is easily verifiable that the first 3 patients were diagnosed with asthma within a month prior to or at the same time their pneumonia was diagnosed, while the other 2 patients in view were not. When alignment is in effect, the time scale becomes relative and labels indicate time relative to the alignment, *e.g.*, "+1 month" and "-1 month". By default the *1st* occurrence is used, but a menu allows users to switch to the n^{th} event, for example, the *2nd*, the *3rd*, the *last*, or *2nd from the last*

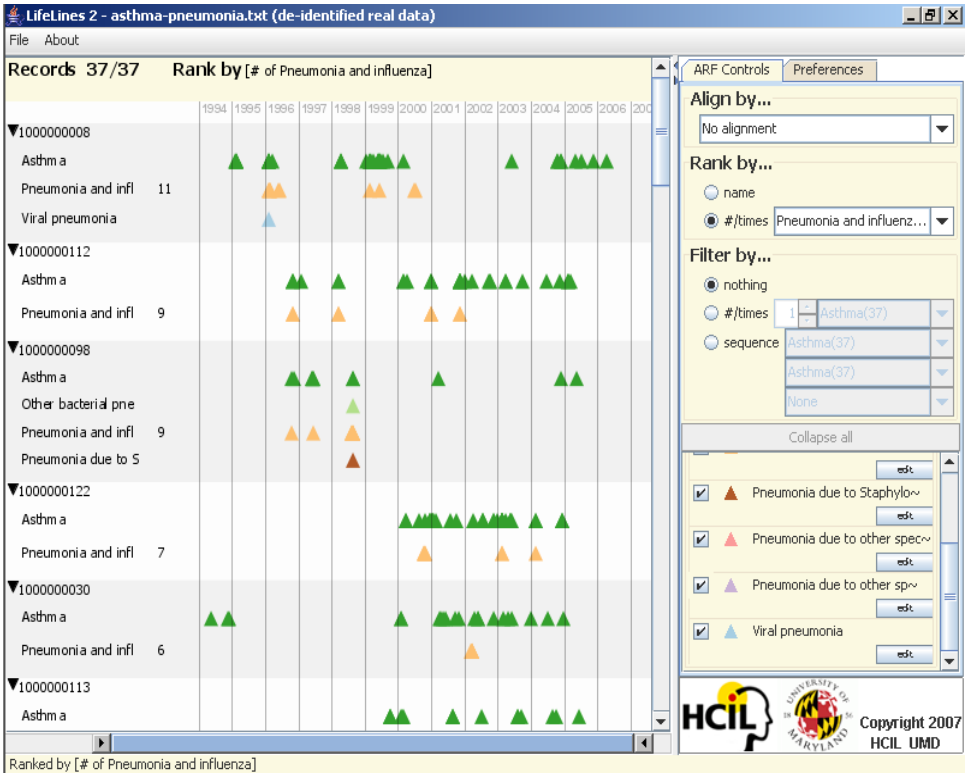

Figure 3.1: Lifelines2 when no alignment is in effect. Each triangle represents an event. Note the data is presented chronologically, and the records are ranked by the number of *Pneumonia and influenza* events.

occurrences. Records that do not contain at least n occurrences of that event are filtered out of the display.

The records are listed in alphabetical order by default but users can rank records by the number of occurrences of an event category. In Figure 3.2 the records are ranked by the number of asthma events, bringing to the top the more severe cases. The number of asthma events for each record is shown next to the event type header for clarity. Instead of ranking the records by number of event occurrences, users can filter it (*e.g.*, removing records that contain less than 2 pneumonia events). Users can also filter out records that do not contain a specified sequence of events

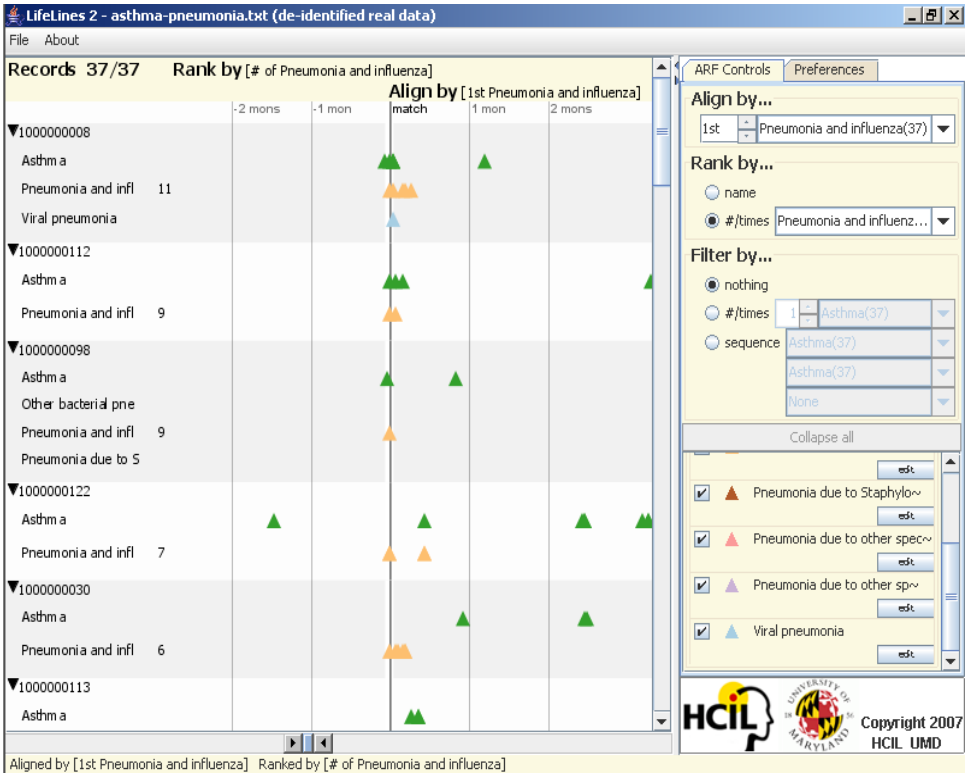

Figure 3.2: This figure shows the same dataset as in Figure 3.1. However, all patient records are aligned by the *1st Pneumonia and influenza*. Note the relative time scale on the top. A single zoom had been applied to the alignment line.

(*e.g.*, asthma followed by pneumonia). Finally the legend area can be used to turn on and off certain types of events from the display to reduce visual clutter and to focus on a subset of event types.

3.3.2 Design Discussion of Alignment

With alignment I believe that Lifelines2 provides a simple yet effective means to quickly focus on a specific characteristic across multiple records, and subsequently allows analysts to explore the data to look for common temporal patterns across

multiple records more quickly. The rationale for this is that without alignment, comparing records via sentinel events would require analysts to zoom in on one record at a time and then inspect the events around the sentinel event. In between zooming in on specific records, an analyst may also need to zoom out to an overview so that he or she can find where the sentinel event is for the next yet-to-be inspected record. On the other hand, when all records are aligned by a sentinel event, one single zoom into the alignment line allows analysts to focus on the details of the sentinel events in all visible records. Overall the need to zoom and pan is greatly reduced, as is the need to keep in memory the scale of time ranges from record to record being compared. Secondly, if there is a secondary event of interest, analysts can better see if there is an occurrence pattern on the secondary event with respect to the sentinel event. Finally, the alignment is dynamic in that analysts can quickly change what the sentinel event is, properly refocus the visualization to match what they are thinking.

Existing visualization systems have exploited visual alignment as a way to rearrange temporal or sequential data to reveal previously unknown patterns. However, these approaches to alignment are different and less-general. Some systems provide an alignment view of the data to allow temporal comparison in relative scale [61, 87, 27, 14], but it is a default view, and users cannot change the alignment dynamically. Systems described in [12, 118] allow data to be displayed in alignment to common cyclical temporal granularities such as days of the week or hours of the day to expose periodicity of the data. These approaches are not data-driven, so analysts cannot use alignment to focus on data characteristics. Other systems such as

Experiscope [29] or SessionViewer [57] require manual specification on data points or time points to be aligned, which is only interactively scalable to a few records. The alignment approach presented here is solely based on data and is automatic in the sense that users do not specify alignment points for each record. Consequently it allows analysts to easily realign as their focus of the data changes. To this end, the alignment presented here is more general than those in the literature. The benefit of alignment in general may seem clear from its wide adoption in these visualization systems. However, there is no previous work quantifying that benefit – which I address in this chapter – nor study of its use in multiple medical applications.

The more traditional ranking and filtering operators expand the set of exploratory tools analysts have for temporal categorical data, however, these operators are not conceptually, visually, or interactively novel ideas and are not the focus of the evaluation. The alignment, ranking, and filtering operators are collectively (and affectionately) called the *ARF framework*.

3.3.3 Design Discussion of Intervals of Validity

To test the idea of displaying intervals of validity (IV), I built a control panel to allow the specification of ranges before and after each event type. Intervals of validity are then displayed visually as a thin line extending from the point event in both temporal directions. The specification of the range contains two parts: (1) a scalar to represent the magnitude, and (2) a unit (*e.g.*, hour, day, month, infinity). Analysts can specify, for example, that *asthma* events have a 4 day range prior to

the actual event to indicate that an asthma episode occurred prior to the recording of the diagnosis event. At the same time, a 2 week range after the events can be assigned to indicate that the patients tend to experience asthma conditions at least two weeks after a diagnosis. Analysts can use *infinity* to designate long-term debilitating conditions such as diabetes once such diagnosis is determined. The goal of intervals of validity is to provide a visual reminder of the possible duration of the state or diagnosis represented by the point event. Analysts no longer need to remember and estimate the duration of each event category, making it easier to spot events that occur concurrently (or in the case of clinical data events that "might" be occurring concurrently). Figure 3.3 shows sample data with and without the interval of validity, and how a trained clinician might erroneously interpret it. In this example the interval values are suggested by our physician colleagues and specified manually by us. Ultimately the length of the intervals would be specified by a trusted, authoritative data provider, or possibly computed based on other factors (*e.g.*, age of the patient). This controlled approach allows us to study how the intervals are interpreted by users and measure if the intervals improve performance at least in the simplest tasks.

3.4 Evaluation

I conducted two separate user studies. The first study aimed to quantify the benefits of alignment and intervals of validity using a controlled experiment. The goal in this experiment was also to observe what strategies analysts chose, and

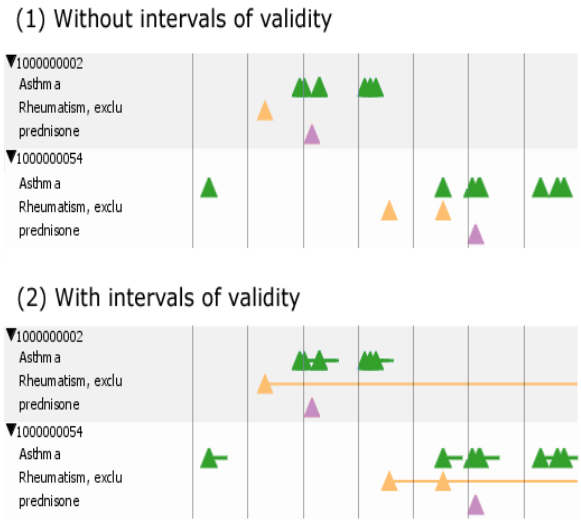

Figure 3.3: The top portion shows two partial records without intervals of validity. Because prednisone (a steroid) prescriptions coincided temporally with asthma diagnoses and with no other events, users may mistakenly conclude that prednisone was given for asthma. The bottom portion shows the same two records with intervals of validity. Rheumatism's lasting interval allows users to visually confirm that there may be more than one reason why predisone was prescribed.

what problems they would encounter. Because medical professionals have very little availability, and are hard to recruit for a user study, this 1^{st} study used data and experts from another domain. We used synthetic data based on graduate school academic events and recruited graduate students, faculty, and staff who are familiar with graduate academic life. I designed the tasks similar to the tasks medical researchers would perform, and verified with medical experts that the tasks were representative . In the second user study, I used real, but de-identified medical data and interviewed medical professionals. The goal of the second study was to obtain domain experts' comments, suggestions, reflections on the uncertainty aspects of

the data.

3.4.1 Controlled Experiment

3.4.1.1 Experimental Procedure

The data consisted of scholastic records for a set of students. Data were point events such as submitting a paper, a software release, dates of a dissertation proposal or defense, signing up for a class, or submitting a job application. There were 20 participants in this study. Each participant was paid $10 for his or her time. The participant who found the most answers correctly (completion time is used as a tie-breaker) was paid an additional $10. The best-performer incentives were used to motivate the participants for correctness over speed and ensure the data collected represent the participant's best efforts.

The experiment was further divided into two independent parts. The first part evaluated the benefits of alignment. The second part evaluated the benefits of showing the intervals of validity. Both parts of the experiment follow a repeated measures design. Each part had two interface variations, and each participant performed a set of tasks once for each variation. I recorded the time to complete each task and errors, if any.

In Part 1, participants were asked to perform tasks regarding events around sentinel events, using 2 interface variations of Lifelines2. One variation had alignment as an operator (ARF), while another did not (RF). I gave a short demonstration of the interfaces, answered questions, and let the participants familiarize

themselves with the interface (for a total 15 minutes of training). I then asked them to perform specific tasks. Because each participant needed to decide on how to best approach a task, I allowed each participant to read each task description and formulate a plan of attack before loading the task data and starting the timer. This ensured that the time recorded was the time of task completion, and did not include plan formulation or task comprehension. I recorded the strategies the participants used. Every participant was asked to perform the set of tasks below, once with the alignment feature available (*i.e.*, with the ARF interface variation), once without (i.e RF). The order in which the two variations of the interface were presented was counter-balanced to mitigate learning effects. The tasks and their design rationale for this part of the experiment are discussed below.

- **Task 1**: How many students submitted a paper within 1 month after proposal? (5 records)
- **Task 2**: How many students submitted a paper within 1 month after proposal? (20 records)
- **Task 3**: How many students published at least 3 papers between proposal and defense?
- **Task 4**: What occurred most often within a month of a student's 1st paper submission?

Task 1 and 2 are similar to tasks where a researcher must study the relationship between a sentinel event and another temporally related event category. The sentinel

event was deliberately made clear in these two task descriptions. While the task descriptions for the two tasks were the same, the data in Task 1 was much simpler than that in Task 2. Task 1 included only 5 records that fit on the screen, so users could find the correct answer without having to interact at all with the display. Task 2 had more records (20) with an expanded time range so users needed to zoom, pan, and scroll significantly to answer the question correctly. First I hypothesized that alignment would reduce the time it took users to perform these tasks even when all data was visible on one screen. I further hypothesized that when more interaction was required, the benefits of alignment would be much more pronounced.

Task 3 is conceptually more complex and there was no clear way to manipulate the data using alignment, ranking, and filtering to find the answers. This task was designed to simulate the process of temporal pattern confirmation clinical researchers might perform (such as confirming a hypothesis about certain patterns of events). While this task required participants to focus on temporal range comparisons relative to a sentinel event (proposal), it did not require detailed inspections that incur extensive interactions. Therefore while I expected to observe alignment's benefits, I also expected them to be less salient.

Finally, Task 4 simulated how a researcher would go about discovering new patterns around sentinel events (in this case, a simple temporally-constrained co-occurrence relationship). I hypothesized that the benefits of alignment would be significant, since both intense interaction and relative comparisons were required in this task.

In Part 2 of the experiment, I used only the interface variation with alignment

(ARF), but how events were represented were varied. In one variation (IV) the lines of the interval of validity were visible; in the other condition (no IV) they were not. Participants performed the set of tasks below, once for each variation. The order in which the interface variations were shown to the users was counter-balanced like in Part I of the experiment. Because every participant performed Part 1 of the experiment before Part 2, they were already familiar with the alignment feature and did not go through another round of training. Participants were asked to fill out a subjective satisfaction questionnaire for each part of the experiment. The two tasks for this part are listed below (Figure 3.4).

- **Task 5**: Assuming a class lasts 3 months, how many students proposed while they were taking a class?
- **Task 6**: Assuming a class lasts 3 months, and it takes 2 months to prepare for proposal, how many students were preparing for proposal while taking a class?

In Task 5, participants were asked to find proposals that occur within 3 months after a class-signup event, and only the interval of the class-signup events was of concern. In Task 6, however, both the intervals of class-signup and the proposal event were important.

I hypothesized that when intervals of validity were shown visually, participants could perform both tasks 5 and 6 more quickly and with lower error rate. In addition, because there were more intervals to keep track of and increased cognitive load, the benefits of the intervals will be more dramatic in Task 6. Since the duration of the

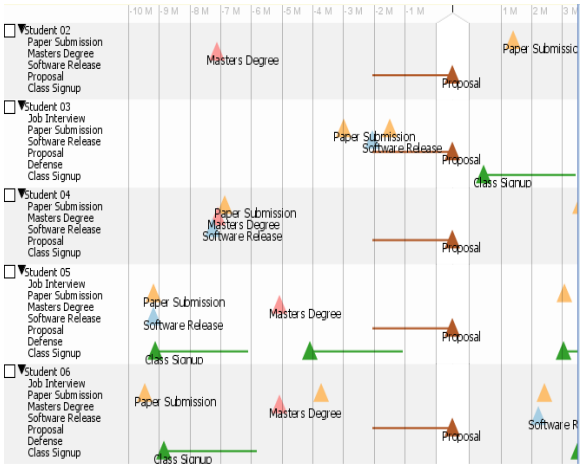

Figure 3.4: A partial screen shot of the data set used in the tasks for Intervals of Validity tasks.

events were described in the task description, this study did not attempt to answer whether the lines might remind users of the likely duration of the event. This was only addressed in the qualitative study described in Section 3.4.2 (page 55).

I used a different dataset for each task in each interface variation to avoid participants remembering previous answers. There were 2 sets of datasets, one for each interface for Part 1. The same applied for Part 2. The data were different enough so that the participants could tell that they were different, but the complexity was comparable between any corresponding tasks in each set. All experiments were conducted on an IBM laptop with the following specifications: 14 inch screen, 1.4 Ghz CPU, 1.2GB RAM, Windows XP Professional. Every participant used an optical mouse instead of the built-in laptop tracking devices.

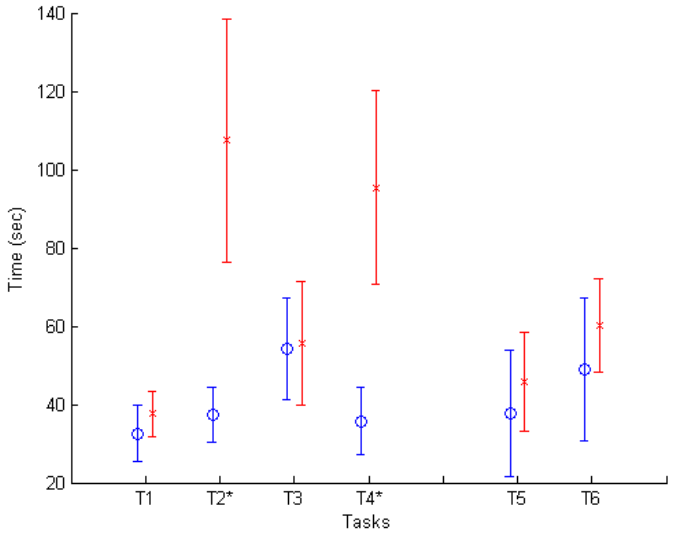

Figure 3.5: Task Completion Time. Each circle or x denotes the mean, and each vertical line denotes the standard deviation. Blue circles denote the results for the align-rank-filter (ARF) variation of the interface in Task 1 to 4 and interval of validity (IV) variation in Task 5 and 6. Red x denote the other interface variation. Although blue circles always have lower means, only in Task 2 and 4 were there statistical significance (denoted by an asterisk in task names). The significance was salient ($p \leq 0.0001$) in both cases.

3.4.1.2 Experimental Results

I analyzed each task separately. I used a repeated measures one-way ANOVA to verify the significance of the time differences in task completion. I used Cochran-Mantel-Haenzsel general association statistic to test whether the error rates between two interfaces were different in both parts of the experiment (p=0.05). Finally, for numerical answers that participants gave, we used a one-way ANOVA to see if the difference in the size of the errors was significant. Figures 3.5, 3.6, and 3.7 show the results, and we discuss them here in detail.

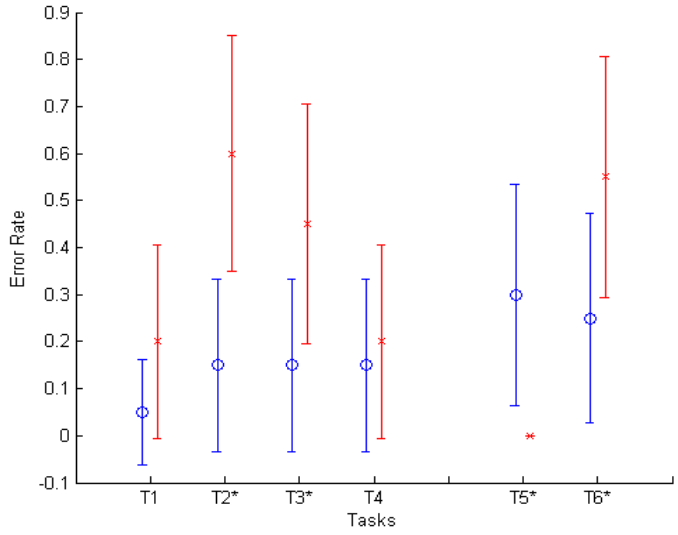

Figure 3.6: Error Rate by Tasks. Blue circles represent align-filter-rank (ARF) variation of the interface in Tasks 1 to 4 and interval of validity (IV) variations for Task 5 and 6. There was a significant difference in error rate in Task 2 ($p \leq 0.01$), Task 3 ($p \leq 0.1$), Task 5 ($p \leq 0.05$), and Task 6 ($p \leq 0.1$). We noted that in Task 5, users performed better when intervals of validity were not displayed.

Throughout the experiment, participants were observed using the following effective strategy: first reduce the data via filtering and alignment (if available), and only manually inspect data that have potential. Ranking was not used very often.

In Task 1 and 2, when using interface with ARF (*i.e.*, with alignment), all except two participants chose to use alignment or alignment in conjunction with a filter. When alignment was not available (*i.e.*, in the RF interface variation), most participants used sequence filter.

In Task 1, while the mean task completion time, error rate, and error size were better in the ARF variation, there was no statistical significance. In Task 2, however,

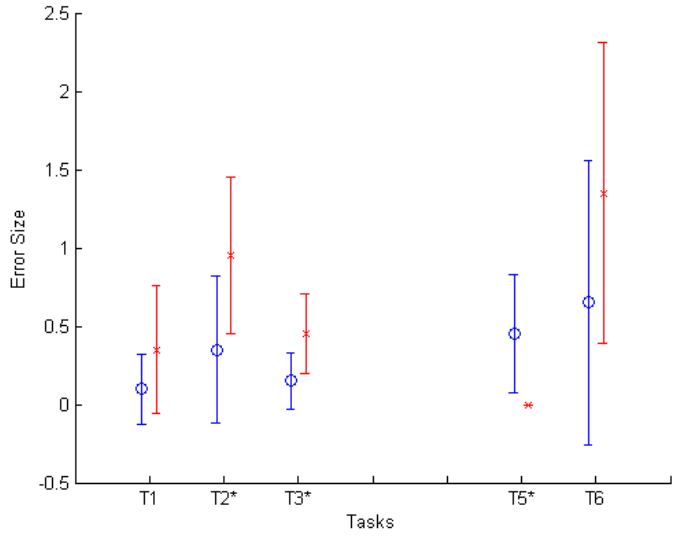

Figure 3.7: Error Size by Task. Blue circles denote the ARF variation of the interface in Tasks 1 to 3 and IV variations in Tasks 5 and 6. Error sizes were significantly smaller for ARF variation in Task 2 ($p \leq 0.05$) and Task 3 ($p \leq 0.1$). Please note that responses for Task 4 were not numeric, so error size was not an applicable measure. Error sizes were significantly smaller in Task 5 ($p \leq 0.05$), in favor of the no IV variation.

the benefits of alignment were statistically significant. Participants were able to complete the task 65% faster with alignment ($p \leq 0.0001$). In addition, participants were less prone to make errors when using alignment ($p \leq 0.01$). When they did make an error, the size of error was significantly smaller than when alignment is not available ($p \leq 0.05$). These results supported the hypothesis that alignment reduces disruptive interaction and allows users to perform these tasks faster and more accurately when the data is larger (*i.e.*, more than a few records). For smaller dataset that fit in one screen, however, the benefit of alignment is very limited.

As we expected, participants had very different strategies in completing Task 3 because the fastest strategy was not as clear as in Task 1 or 2. When the alignment

operator was available in the ARF variation, over two-thirds of the participants aligned by *proposal* and used a type of filter. Most popular filters were "at least 3 *paper submissions*", and the sequence filter "*proposal*, *paper*, *defense*". No participant tried to align or filter solely by *defense*, although it was the most effective technique in this question (students must have a proposal before they can defend, and by aligning by defense, students proposed but have not defended would be eliminated from view, reducing the number of records one needed to inspect). This reflected that it takes time to learn to make the best use of any new feature, despite the familiarity of the domain knowledge. When alignment was not available, sequence filter was the favorite among the participants. The sequences "*proposal*, *paper*, *defense*" and "*proposal*, *defense*" were the most popular.

In Task 3, the difference in completion time between the two interfaces was negligible (2.9%). Participants tended to make an error less often when alignment was available ($p \leq 0.1$), however, there were no significant differences in error size. This result did not support our hypotheses on completion time. Although it supported our hypothesis regarding error rate.

The performance difference in task completion again was startling in Task 4. Using alignment, participants were able to complete the task 62% faster ($p \leq 0.0001$) with no significant difference in error rates. All but 1 participant used alignment when alignment was available. When using alignment was not an option, most users opted to use "paper submission" as a filter but seven participants used neither filter nor ranking, and relied on manual navigation and their vision.

In the second part of the experiment, Task 5 and Task 6 were designed to

measure the possible benefits of intervals of validity in terms of the time it takes to visually find potential overlaps. My hypothesis that intervals of validity help users perform faster and more accurately was not supported. I found that there were no statistically significant differences in completion time. There were significant but conflicting differences in error rates. Surprisingly, users were less likely to make a mistake when intervals of validity were not displayed in Task 5 ($p \leq 0.05$). In Task 6, users were less likely to make a mistake (by a margin of 30%, $p \leq 0.1$) when intervals of validity were displayed.

Task 5's result was puzzling. One participant offered one possible explanation in her comments. She noted that she relied on the visualization more when intervals of validity were displayed, and was less inclined to zoom-in to verify her answers, especially when the task is as simple as Task 5. And so when the difference is only a few pixels on screen, a wrong judgment may be made.

An interesting observation was that every participant used the alignment function in both Task 5 and Task 6, regardless of whether they had been particularly successful with alignment in the first part of the study. We hypothesized that participants recognized that Tasks 5 and 6 were relative comparison tasks, and understood (presumably from previous experiences with it) that alignment would get them to the answer quicker.

In the questionnaire users rated alignment very positively. On a scale of 1 to 9, users agreed that alignment was helpful in Tasks 1 to 4 with a mean of 8.3. They also agreed that they were able to perform faster in Tasks 1 to 4 when alignment was available with a mean rating of 7.95. One participant commented that "scrolling was

a big distraction, and alignment helped eliminating a lot of it". Another participant commented that when using the interface where alignment was not an option, he noticed that he was mentally doing the alignment himself, suggesting that alignment may be a natural operator when people need to reason about time. While intervals of validity were still rated as helpful in Tasks 5 and 6 (7.85), the overall perception was that they were not as critical as alignment. Our participants agreed that they could perform tasks 5 and 6 more quickly with intervals of validity with a rating of 7.55. Clearly, the lack of statistical significance in these tasks suggested that perceived performance gain is not the same as actual performance gain. One participant did comment that the intervals of validity allowed him to get an idea how long the durations are, "especially if I forgot."

3.4.2 Domain Expert Interviews

3.4.2.1 Interview Procedure

In this initial study I interviewed 4 medical professionals – one registered nurse, one physician, and two faculty members in a nursing school, none had a connection with the development of this interface. Three have had experience with medical informatics systems and have been involved in medical research. I showed them our application using de-identified data from our colleagues at Harvard. I asked them to talk out loud while using the application, without any training. I recorded how they interpreted the visual display and their impressions of the interface, the ARF framework, intervals of validity, and collected what they felt were usability issues

and thir suggestions to future refinements.

I first loaded a dataset corresponding to the problem described in Section 3.2 (page 35) regarding the uncertain nature of the data, and how it was to be addressed by intervals of validity. It contained 11 patients who had asthma, and were prescribed some type of steroid. Some of the patients had other conditions (*e.g.*, rheumatoid arthritis, pneumonia) that might also require steroids treatments. The only introduction I gave to the medical professionals was that Lifelines2 was intended to view results from a search in patient databases. I gave no tutorial, nor explanation of what they were seeing on screen initially. I asked our interviewees to comment on what they saw and how they interpreted the display.

Once they had reviewed the data, I asked them to imagine that they were selecting patients for a study, and they only wanted patients who were given steroids for their asthma condition. Those who were given steroids for other reasons should not be included. Of course, there was no way to know the actual reason why a patient was prescribed steroids, but I wanted them to select what they thought were the likely ones, and filter out the ones that seem unsuitable to trigger some reasoning about the uncertainty of the data. I then asked them to go through the same exercise again with the same data, but with the intervals of validity visible (Figure 3.3 shows the differences in display). I used interval values provided by our colleague at Harvard Medical School. Rheumatoid arthritis and rheumatisms were given the interval of infinity after each event occurrence. Asthma had 4 months after, and pneumonia had 2 weeks after. I asked our experts how they would interpret the intervals. I also looked at how their answers differed in the two sessions, and asked

how the intervals made them change their mind.

The second data set they looked at contained patients with asthma and pneumonia. I asked our medical experts to look at the data and see if they could find trends in the dataset. In particular, whether there were more patients who first contracted pneumonia before an asthma event or there were more of those who first had asthma, then pneumonia. I showed them how to use alignment, ranking, and filtering.

Then, using a third dataset containing 45 patients with various heart problems, I asked our experts to use the interface on their own and to discuss what might be the most commonly co-occurring event to an acute myocardial infarction (heart attack) in this set of patients. This time around, I let them go about the task without much guidance or interruption. I aimed to observe whether ARF made sense to them, and how they would utilize its functionality on their own.

3.4.2.2 Interview Results

Three out of four participants had no problem interpreting the visualization from the start. They were able to immediately figure out the timeline, each patient's record, and the temporal event data shown as triangles. One participant did not immediately grasp that individual patient records were displayed. She had interpreted the triangles to indicate sets of patients at the beginning, but soon after realized on her own what was going on. Two participants commented on how helpful the color encoding was.

Looking at the asthma-steroid data, all of them talked about the asthma triangles as flares, not merely diagnoses. When asked to find patients who were given steroids for their asthma condition, most of them employed the strategy I had anticipated: find each steroid prescription event, and see what the closest event prior to that was. If the closest event was asthma, and it was within a few months, then they are likely to think the steroid was prescribed for asthma. Two participants took into account the frequency of how correlated the steroid prescription event is to other events. When no intervals of validity were displayed, none of the participants paid much attention to the possibility of a long lasting rheumatism condition. When the intervals of validity were displayed, they all interpreted them as durations. However, two participants interpreted the intervals as certain or known duration, and not as an uncertain, possible duration.

The important result was that showing intervals affected how our participants viewed and interpreted the data. When they were asked to find patients who were given steroids for asthma with intervals of validity, the lines reminded them the durations of each event, and some of their answers changed from the no interval version, although the direction of change did not always conform to what we expected. We had expected them to have picked fewer patients for the clinical trial because they might have forgotten or ignored an existing rheumatoid arthritis. However, two of the participants picked more patients because some unexplained steroid prescription event is now explained by an asthma that occurred just 3 months before. This was now easy to see with the 4 months duration for each asthma event. Two participants did not like that fact that someone else could assert intervals because

they might not trust that person. They did agree that if they had added the intervals themselves, tailored to the questions they have in mind for the field that they are in, then they would have no objections. One participant commented that "intervals of validity were good reminders of uncertainty, and using the range to perform overlapping tasks is much simpler." He also commented that long intervals draw a lot of attention, and whether such attention is good might depend on the situation.

One other interesting observation was that the Lifelines2 was able to elicit narration from the domain experts. They would look at one particular patient record, consider the temporal patterns in that record, and rely on their domain knowledge to construct a story behind the visualization and explain why it occurred that way. For example, one participant observing the asthma and steroid dataset (which also included pneumonia, rheumatism, etc.) saw one particular patient who had pneumonia on a yearly basis. She was able to make the observation that the patient was probably an elderly and had pneumonia every flu season (the record showed pneumonia events during every autumn). Asthma then would often co-occur with pneumonia to make the patient sicker. In one year, the data shows that the steroid was given to the patient. The domain expert made the observation that the steroid was probably a prophylaxis given ahead of the flu season, and explained the subsequent lack of asthma in that season.

Participants were quick to grasp the ARF framework of alignment, rank and filter. They figured out how to use them on their own without any introduction or demonstration of the functionalities. Using the asthma-pneumonia data, all partici-

pants were able to quickly align or filter to see if there were overall patterns in those patients. They did comment that alignment and ranking allowed them to figure out the data quickly. They also liked that once the data is aligned by a patient's 1st pneumonia, zooming in and scrolling down were all it took to get through all the patients efficiently. One participant said that grouping records by whether a second sentinel event occurred before or after the 1st one might be a good addition. Ranking using temporal relationships to aligned sentinel events was also mentioned as being potentially useful.

When I loaded the set of patients with heart diseases and asked our participants to discover patterns of events with regard to acute myocardial infarction (AMI), 2 out 4 quickly approached the question with alignment on the correct event. They were able to see that coronary atherosclerosis co-occurred with AMI most often in this dataset. One participant, using her medical knowledge, found another association pattern between coronary atherosclerosis and hypertensive disease events in this data.

When I asked our interviewees on what they thought about Lifelines2 and its applications, two participants quickly offered examples where Lifelines2 could be useful in their own research and became very enthusiastic. They both were involved in working with clinical data (though independent from each other), and were impressed with how effortless it was to identify certain patterns using the ARF framework. One example was to use this approach in a study of patterns of events that might suggest medical mis-diagnosis in an emergency room.

In addition to this pilot study the prototype was demonstrated to our sponsors

and partners and received unusually encouraging feedback. During a presentation at the National Institute of Health a respected medical researcher at the University of Chicago commented supportively, "[it will] change the entire paradigm in medicine".

3.5 Discussion

3.5.1 Post-Experiment Design Improvements

From the feedbacks of the controlled experiment participants, interviews with the medical professionals, and continuing work with the physician collaborators, I have made the following changes to Lifelines2 since the experiments described here.

1. The mixed experimental results regarding *intervals of validity* suggested that it is not as critical a feature as I had previously thought. Interface changes have been made to bring forth other more important features, while *intervals of validity* is now embedded deeper in the application. Analysts can still bring up the control panel to set, and adjust and activate/deactivate intervals of validity, but now it takes more effort to get to. More over, the subsequent case studies with physicians and hospital administrators, *intervals of validity* was never discussed as an additional visual aid to the data even though it was a feature that had been demonstrated.

2. A means to maintain the visual encoding of each data set was implemented. A customization (.cstm) file is now automatically created to remember the various settings analysts have set for each particular dataset. The customiza-

tion file is automatically written to disk when Lifelines2 exits normally, and automatically loaded when the same dataset is opened by Lifeliens2. It remembers features such as color/size of event encoding, whether to show labels, size/position of labels, settings for intervals of validity, etc. Additionally, analysts can overwrite how events types are ordered by default. Analysts can specify groups of events and the order in which they should be displayed. For example, the events *High Blood Pressure*, *Normal Blood Pressure*, and *Low Blood Pressure* would be ordered *High*, *Low*, *Normal* if the default alphabetical ordering is used. Instead, analysts can change the ordering by creating a group in the customization file that specifies these three event types must be presented in a specified order. The persistence of visual encoding and options allow analysts to better maintain a mental model of the data in between sessions.

3. The filter control has been extended in two ways. First, to better support the useful alignment operator, the number of occurrence filter (*i.e.*, "find all records that have 3 instances of *heart attack*") has been extended so that when there is an alignment active, analysts can apply the filter to *only before*, or *only after* the alignment, in addition to *the entire time*. Secondly, the sequence filter has been extended to incorporate *absence events*. That is, when filtering by sequence, analysts can specify a search pattern to find students who had "proposal followed no paper submissions followed by defense", or in plain words, find students who "proposed and then defended, but did not submit

any paper during that period". An algorithm designed for Lifelines2 to support this kind of filter efficiently is presented in Chapter 4

4. Instead of using one single (thickened) vertical line to indicate alignment, Lifelines2 used a thick band to indicate alignment. The thick band does not indicate any actual time range on screen, and because of that events that occur very closely to the sentinel events are pushed out of the band, giving the sentinel events more space and clarity. I at first had used a bright yellow band to bring extra visual prominence to the alignment line. Credit goes to Allison Druin and her keen design sense that the current white band takes place.

5. My collaborators told me that although studying the 1^{st} heart attacks across patients is interesting, sometimes they are interested in *all* occurrences of heart attacks. In particular many medical conditions are defined by having certain events within certain time frames, and focusing only on the n^{th} event would not suffice. Therefore the alignment operator has been modified to handle *align by all events* in addition to align by the 1^{st}, 2^{nd}, or the 2^{nd} *to last* event. This way, trends for a specific type of event regardless of their ordering may become more visible.

6. Distribution of event types was requested as one of the several ways physicians would use to understand a group of patients. I have implemented *temporal summaries* to show distribution of event types across multiple temporal granularities. Its detailed description and usefulness are reported in Chapter 5.

3.5.2 Generalizability and Limitations

The idea of alignment is generalizable to broader classes of temporal data. I have shown that it works for point-event temporal categorical data in Lifelines, however, it is not difficult to imagine directly extending to events with durations. No question, the alignment may become more complex, such as options to align by the starting of duration, ending of duration, or mid point of duration.

Temporal visualization systems that include both numerical time series (*e.g.*, a person's body-mass index over time) and categorical events (*e.g.*, medical diagnoses), can also make use of alignment on the categorical events to explore trends in time series with respect to the alignment events. Extending to pure numerical data may be more difficult. One way to deal with this is to first convert numerical data into categories. Another way is to define categories by the numerical trends in fixed temporal duration (rise, fall, sharp rise, sharp fall, plateau, etc.) similar to the formalism described in [3]. Each approach has its strengths and weaknesses, and may be more suitable to one domain more than the other.

3.5.3 Impact

The idea of alignment has raised the importance of relative temporal comparisons. Other systems, including a couple from our lab, have since then utilized and extended it in different ways.

The projects PatternFinder in Amalga [82] and Similan [120] from our lab have incorporated alignment as part of issuing temporal queries. In the PatternFinder in

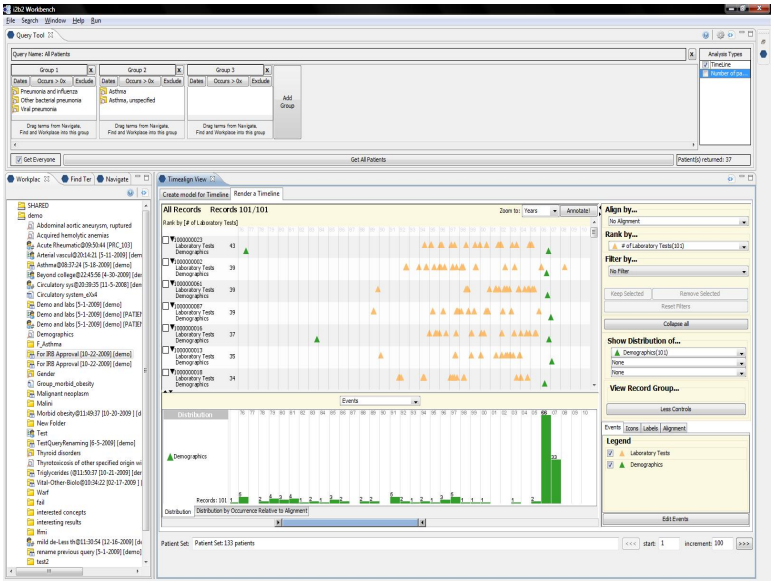

Figure 3.8: The incorporation of Lifelines2 in the patient database query system i2b2 as a way to display query results.

Amalga query interface, analysts would specifically specify a baseline event (prior to the alignment), a sentinel event, and a post-Alignment event and the result visualization centers around the alignment. In Similan, the use of alignment is optional. Analysts uses alignment to both view the temporal data on a relative time scale, and to perform temporal queries relative to the sentinel events. Klimov *et al.*, recognizing the importance of relative time queries, has incorporated mechanisms (similar to that of alignment) to allow analysts to specify relative time queries in patient databases [52]. Finally, my collaborators at Harvard Medical Center have worked to incorporate Lifelines2 into their patient record querying system i2b2 (Figure 3.8).

3.6 Summary

This chapter describes evaluation with two features aimed to support the tasks of visually browsing records of temporal categorical data. *Alignment* and *intervals-of-validity* were evaluated in a controlled experiment. The benefits of alignment were statistically significant for relative comparison tasks where the data is more complex. However, the benefits of intervals-of-validity is not as clear. Subsequent changes to the interface took into account these findings. Lifelines2 as a whole was qualitatively evaluated with medical professionals. The reported learnability and applicability were good. Participants were enthusiastic and interested despite the discovery of several usability issues.

Chapter 4

Supporting Temporal Pattern Search: An Algorithm

4.1 Overview

Temporally ordered events can often reveal interesting information. Understanding whether a particular pattern occurs, how frequently certain patterns occur, and whether one pattern occurs more often than the others are common questions an analyst would pursue. There has been much attention from both academia and industry to develop analysis tools focused on temporal events and their temporal patterns in a variety of domains: health care and electronic health records (EHR) [115, 20, 24], business intelligence [106], and web server log analysis [57].

In particular, electronic health records have gained much attention in recent years. Clinicians recruiting participants for clinical trials often screen candidates by reviewing their records. In the initial stages, they would issue queries to find participants with certain features. These features include patient attributes such as age and gender, but also often include a temporal pattern of events. Some query patterns are as simple as finding participants who "had a stroke after a heart attack". Some may involve negation, *e.g.*, "diagnosed with breast cancer without having a prior mammogram", and some may be more complex, *e.g.*, "with no prior history of heart problem, later diagnosed with hypertension, received no treatments, and finally experienced a heart attack".

Searching for records that contain a specific sequence is an important task to our collaborators in studying patient records. However, most of the visualization applications mentioned above, and current state-of-the-art query systems have no accessible features that allow them to do so. In this chapter, I first motivate the problem and present the Temporal Pattern Search (TPS) algorithm. I analyze the worst-case running time of the algorithm and evaluate its performance against existing common approaches in experiments with random data and in experiments with real data. The content of this chapter corresponds to a technical report submitted for publication [114].

4.2 Background and Motivation

While the typical model to find temporal patterns is to perform the search using a command line language such as SQL, and then review the results in an analysis tool, there are high costs associated with this approach. First, the query languages themselves pose as a significant technical barrier. To be able to specify temporal patterns, domain analysts have to learn the syntaxes, where esoteric parts may be required (Appendix B.1 describes a detailed example). Because domain analysts often do not know the query language, they have to rely on an additional SQL expert to formulate the queries. The collaboration turnaround may take hours, if not days. In a situation where an analyst is iteratively refining a query for the purpose of exploratory analysis, the multiple turnarounds are detrimental to the process.

In some settings, however, existing interfaces exist to circumvent the steep learning curve of query languages. In these settings, analysts can use higher level commands and rely on the interface to translate them into SQL queries. However, more often than not, the interfaces reduce the expressiveness of the underlying language in favor of usability, learnability, and familiarity. For example, searching for temporal patterns may be unsupported by the interface, in favor of the simpler Boolean or conjunctive queries. The inflexibilities often frustrate our physician/clinician collaborators using state-of-the art electronic health record database interfaces such as Amalga [68] or i2b2 [70].

Finally, searching temporal patterns on personal histories that have hundreds or thousands of events with tens of thousands of histories in a database can take a long time. Experiences in building a query interface extension for Amalga revealed some performance problems using SQL [58, 82]. A temporal pattern query in SQL is not feasible for the hospital's database of thousands of patients because of a prohibitively high number of self-join operations. Only after building additional indices and preprocessing (which itself can take hours) could a temporal query be managed [58]. The performance time also increases exponentially with the number of elements. It is unrealistic to perform tailored optimization techniques for each potentially different temporal pattern search. Instead, it is more effective to break down a temporal pattern search into two stages. First issue a conjunctive SQL query to find patients who have events A, B, and C, which can be executed efficiently. The temporal pattern search (*e.g.*, A followed by B followed by C) can then take place in an analysis tool, where data can be further structured to better support these

searches.

However, not all visual analysis tools provide the features that support search for temporal patterns. One of the challenges for these systems is that it may be difficult to have one data structure that is suitable to both a fluid and speedy interaction for the visualization and good search performance. That is, by indexing the data smartly, a system can optimize interactive speed. However, a different indexing may allow the system to optimize search speed. Simultaneously maintaining two indices would be a fine solution, but doubling the memory usage is pricey. When temporal pattern search becomes an indispensable feature, I had to design an algorithm that works with the Lifelines2 data structure while maintaining good search performance. Below I give an overview of the data structure in Lifelines2 and the interface for the temporal pattern search, and I discuss the challenges the data structure presented to the design of the algorithm.

Lifelines2 visualizes each record as a row on the time line (Figure 4.1 (a)). Within each record, the events of the same type are placed on the same horizontal line. Event types are differentiated by color. To clarify, by *record* (as, for example, in *electronic health record (EHR)*) I mean a collection of event histories associated with an entity (such as a patient in the context of EHR) and not tuples as in the database literature. Lifelines2 provides several ways to filter records by their features. One of them is via a sequence filter (for which TPS is designed). Previous research suggested complex interfaces with full control of all temporal constraints among items in a pattern often overwhelms users [24]. Therefore, I decided on a simplified temporal pattern specification interface. Although more restricted, it can

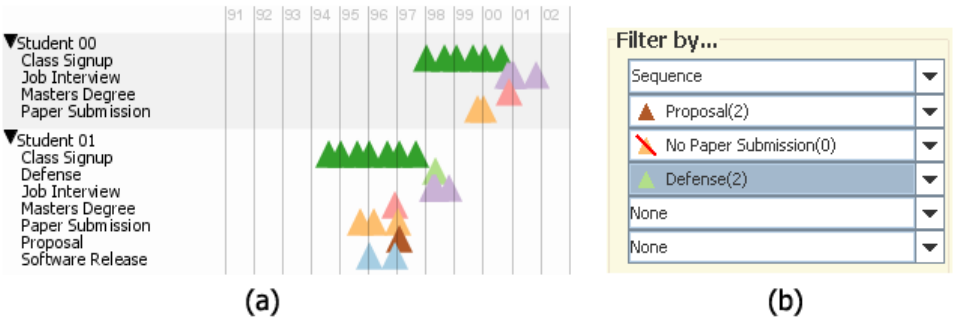

Figure 4.1: (a) Partial screen shot of two students records in Lifelines2. (b) Combo boxes used to specify temporal patterns for search.

still specify all the aforementioned example medical queries in Section 4.1. Lifelines2 allows users to specify a temporal pattern such that each item in the pattern can be a presence item or an absence item (negation). These items are selected from a list of combo boxes (Figure 4.1 (b)). The ordering of the combo boxes determines the temporal ordering. Lifelines2 does not allow other temporal constraints (*e.g.*, event A occurs within 3 days after event B) among the items on the pattern. However, this type of temporal constraints can be specified via *temporal summaries* in Lifelines2 [116](also see Chapter 5). Once a pattern is specified, Lifelines2 finds all records that contain such a pattern and visually filters out the rest.

I made the following design decisions on how I internally represent the event histories in Lifelines2. As these decisions were made to support existing features of the tool (align, rank, filter (without negation in sequence filter initially)) and before I designed the TPS algorithm, the constraints represent interesting challenges. In Lifelines2, each record is represented as a set of sorted arrays of events. There is one array for each event type. All events of the same type are sorted by their time

stamps in their array. This decision comes from the following constraints:

1. **Data Constraint**: It seems most straight-forward to store all events on a single sorted array regardless of type. However, for events that have the same time stamp, this scheme can create conflicts, mislead analysts, and produce wrong results. Using one array for each event type allows us to circumvent this problem. (However, I assume events that have the same type and the same time stamp are the same event). The number of times that two events have the same time stamp varies by dataset. Table 4.1 shows selected clinical datasets the collaborators supplied. The proportion of events that have the same time stamp can range from less than 0.5% to almost 50%.

2. **Drawing Constraint**: Lifelines2 maintains the drawing order of events by event types. Events of the same type are rendered together (on the same horizontal space). Lifelines2 maintains the Z-order by event types to avoid visual inconsistencies that can potentially disrupt analytical tasks. The separated arrays would allow the drawing algorithm an efficient way to access events of the same type.

3. **Interface Constraint**: While searching for temporal patterns is important, it is not all that Lifelines2 does. Other operators designed for exploratory analysis (such as *alignment* and *rank*) benefit from this separate arrays approach. In addition, it is useful to analysts to hide event types that are not of interest. These interface features involve finding event data of a specific type. Separating events into different arrays by type would allow Lifelines2 to afford

Dataset Name	#Records	#Events	#Events with Same Time Stamp
Creatinine	3598	32134	16
Heparin	841	65728	31251
Heart Attack	9361	196581	93610
Transfer	51006	207187	51318
Bipap	6583	135951	14650

Table 4.1: Selected Real Datasets and Their Statistics

these features most efficiently.

These constraints present certain challenges to searching temporal event histories, but also present a unique opportunity to explore and exploit this data structure. This chapter focuses on my exploration in designing a temporal pattern searching algorithm in the presence of these constraints, especially the latter two constraints that involve human performance and visual design. I describe the temporal pattern search problem formally and give a description of my algorithm in the following sections.

4.3 Problem Description

An event e is defined, and uniquely identified by an event type $T = T_e$ and a time stamp $t = t_e$. A personal record (such as a personal electronic health record) consists of a set of k sorted arrays. Each array corresponds to an event type T_{e_i}, where $i \in [1, k]$, and contains all events of that type in the person's history, sorted by their time stamps.

A temporal pattern specifies the ordering of events as a pattern and may include negations. More formally, a temporal pattern $P = p_1 p_2 p_3 \dots p_m$, where each p_i is an event type T_{p_i} or its negation $\bar{T}_{p_i}$. In this paper, we use the word negation

and absence interchangeably.

In the case that a temporal pattern contains only positive items (no negation), the pattern $P = p_1p_2p_3 \ldots p_m = T_{p_1}T_{p_2}T_{p_3} \ldots T_{p_m}$ matches a personal record if and only if $\exists e_1, e_2, e_3, \ldots, e_m$ such that $T_{e_i} = T_{p_i}$ *and* $t_{e_i} < t_{e_{i+1}} \forall i \in [1, m-1]$ in the record. In another word, the event types in the specified pattern exist in the record, and that they occur in the order the pattern specifies.

If a negation exists, *e.g.*, $p_1p_2p_3 = T_{p_1}\bar{T}_{p_2}T_{p_3}$, then the pattern matches a record if and only if $\exists e_1, e_3$ such that $T_{e_1} = T_{p_1}$ *and* $T_{e_3} = T_{p_3}$ *and* $t_{e_1} < t_{e_3}$ *and* $\nexists e_2$ such that $t_{e_2} \in (t_{e_1}, t_{e_3})$. If there are contiguous negation items such as $p_1p_2p_3p_4 = T_{p_1}\bar{T}_{p_2}\bar{T}_{p_3}T_{p_4}$, then the sequence matches a record if and only if $\exists e_1, e_4$ such that $T_{e_1} = T_{p_1}$ *and* $T_{e_4} = T_{p_4}$ *and*$t_{e_1} < t_{e_4}$ *and* $\nexists e_2, e_3$ such that $(t_{e_2} \in (t_{e_1}, t_{e_4})$ or $t_{e_3} \in (t_{e_1}, t_{e_4}))$. The semantics of the temporal pattern we discuss in this paper can be mapped into regular expressions, for example, $P = T_{p_1}\bar{T}_{p_2}\bar{T}_{p_3}T_{p_4}$ can be translated into a semantically equivalent regular expression $.*T_{p_1}[\hat{}T_{p_2}T_{p_3}]*T_{p_4}.*$. My problem is how to search for records that match the specified pattern under the data structure we have discussed.

4.4 TPS Algorithm Description

Code Listing 4.1 shows the pseudocode for the algorithm Temporal Pattern Search (TPS). `IS_A_MATCH(R,P)` takes a record `R` , and a temporal pattern `P`. A record `R` is a table of arrays, indexed by event types. Each array contains all events of the same type, and can be referenced by `R[T]`, where `T` is the event type. Each

event `e` has a time stamp `e.TIME` and a type `e.TYPE`. `P` is an array of pattern items. Each item `item` includes an event type `item.TYPE`, and a flag whether it is a negation item or not `item.isNeg`. Table 4.4 shows a trace of the TPS algorithm with example `P` and `R`. For the ease of narration, we have represented the input record R as an array in 4.4 and refer to its i^{th} elements using the array notation $R[i]$ in the following discussions.

```
0   IS_A_MATCH(R, P)
1     β ← false       //whether to backtrack
2     T[P.length]    //matched times
3     Δ[P.length]    //indices of last pos item
4     Φ[P.length+1] //following neg item times
5     Π[P.length]    //if has neg item after
6     χ ← 0          //current index
7     τ ←-∞          //current time
8     δ ←-1         //last pos item
9
10    while (χ<P.length)
11      if (χ ==-1)
12        return false
13      if (P[χ].isNeg)
14        HANDLE_ABSENCE(R, P)
15      else
16        HANDLE_PRESENCE(R, P)
17    return true
```

Code Listing 4.1: starting function and initialization of global variables.

In addition to the details that are shown, the pseudocode assumes that there exists the following function `NEXT(R[T],t)`, where `R` is a record, `T` is an event type, and `t` is a time stamp. `NEXT(R[T],t)` returns an event `e` of type `T` in `R` such that `e.TIME > t` and that there is not another event `d` of type `T` in `R` such that `d.TIME < e.TIME`. If no such event `e` exists, the function returns `NIL`. In other

index:	0	1	2	3	4	5	6	7	8	9
time:	0	10	20	30	40	50	60	70	80	90
Record **R** =	A	C	E	B	C	A	C	D	C	F

index:	0	1	2	3	4	5
Pattern **P** =	A	$\bar{B}$	C	$\bar{D}$	$\bar{E}$	F

	Current index χ and current time τ	Binary Search Performed in matching pattern with data	Code Line #	β	T	Δ	Φ	Π	χ	τ	δ
0		initialization		FALSE	[ ]	[ ]	[ ]	[ ]	0	-∞	-1
1	χ = 0, τ = -∞	matching P[0] = A … R[0]	(73-78)	FALSE	[0]←0	[0]← -1			1	0	0
2	χ = 1, τ = 0	matching P[1] = $\bar{B}$ … R[3] matching P[2] = C … R[1]	(48-53)		[2]←10	[2]←0	[0]←30	[2]←TRUE	3	10	2
3	χ = 3, τ = 10	matching P[3] = $\bar{D}$ … R[7] matching P[4] = $\bar{E}$ … R[2] matching P[5] = F … R[9]	(42-46)	TRUE				[5]←TRUE	2	19	0
4	χ = 2, τ = 19	matching P[1] = $\bar{B}$ … R[3] matching P[2] = C … R[4]	(68-70)	TRUE					0	29	-1
5	χ = 0, τ = 29	matching P[0] = A … R[5]	(73-78)	FALSE	[0]←50	[0]← -1			1	50	0
6	χ = 1, τ = 50	matching P[1] = $\bar{B}$ … NIL	(30-34)				[1]←∞		2		
7	χ = 2, τ = 50	matching P[2] = C … R[6]	(73-78)	FALSE	[2]←60	[2]←0			3	60	2
8	χ = 3, τ = 60	matching P[3] = $\bar{D}$ … R[7] matching P[4] = $\bar{E}$ … NIL matching P[5] = F … R[9]	(42-46)	TRUE				[5]←TRUE	2	69	0
9	χ = 2, τ = 69	matching P[2] = C … R[8]	(73-78)	FALSE	[2]←80	[2]←0			3	80	2
10	χ = 3, τ = 80	matching P[3] = $\bar{D}$ … NIL matching P[4] = $\bar{E}$ … NIL	(30-34)				[2]←∞		5		
11	χ = 5, τ = 80	matching P[5] = F … R[9]	(73-78)	FALSE	[5]←90	[5]←5			6	90	5

Table 4.2: A trace of the TPS algorithm with example inputs R and P

words, NEXT(R[T],t) returns the event of type T that occurs after and closest to the given time stamp t, if it exists. Since all event arrays in records are sorted by time stamps, it is assumed that NEXT(R[T],t) uses an efficient implementation of binary search.

4.4.1 Algorithm Overview

The main loop IS_A_MATCH(R,P) processes an item from the sequence P one at a time. The variable χ (current index on the sequence) keeps track of which item is current. When all items in the sequence have been found and no constraints from

negation events are violated, TPS returns `TRUE`. However, if χ is set to -1, it means that there is no match and the main loop returns `FALSE`.

When processing each item on the pattern, if it is a presence item, `IS_A_MATCH(R,P)` calls `HANDLE_PRESENCE(R,P)`, which attempts to find an event that satisfies the current item. If it is an absence item, TPS calls `HANDLE_ABSENCE(R,P)`, which finds the next absence event, and checks to see if that absence event occurs between the previous presence item match and the next presence item match. If it does, then a constraint is violated, and the algorithm backtracks. Backtracking means TPS tries to look for an alternative to one or more of its previously made matches. The algorithm increments the variables χ (the current item on pattern) and τ (the current time) when processing the search. When backtracking occurs, TPS rolls back these variables, among others, appropriately to restart a previous search.

The first thing to notice about this description is that there would be no need for backtracking if there were no absence events. Furthermore, when it backtracks, it backtracks only to the previously closest presence item (never to an absence item). The characteristics on how the algorithm backtracks determine what states we need to keep track of.

We use Greek letters to denote global variables. Capital letters represent arrays, and lower case ones represent single variables. Aside from χ and τ, Code Listing 4.1 also shows the initialization of T, Δ, Φ, Π, β, and δ. They describe the states TPS is in when matching and save previously-done work to reduce unnecessary backtracking. β keeps track of whether we are backtracking. β is a boolean, and indicates whether we are backtracking. δ represents the index of the previously

matched presence item (while ? is the current index). *Delta* is an array of length `[P.length]`, and it keeps track of the index of the last presence item for each item in the pattern. So Δ`[i]` indicates the index of the last presence item for the i^{th} item in the pattern (Δ is an array of δ's). T is another array of the same length, and keeps track of the time stamps for all previously-matched presence items.

For each item on the sequence, Φ keeps track of the known time for the following absence event (if the following item is an absence event). This allows TPS to skip some fruitless matches during backtracking. Suppose we are searching for pattern $P = A\bar{B}C$ in a record $AAAAAABAC$. $P[0]$ will initially match the first A in the record. Upon encountering B in the record and backtracking to $P[0] = A$, the next search should start from the time stamp of the known next absence event B (instead of from the time stamp of the known bad A) to avoid the multiple, unnecessary matches to all of the A's before the B in the record. Φ has length $P.length + 1$. The extra space at the end is used when a pattern starts with a negation, and needs a default place to remember that absence item. Finally, for each item in the pattern, Π keeps track of whether it has an absence item immediately before that item. If $\Pi[i] =$ `FALSE`, it indicates that there is no absence item prior to the i^{th} item in the pattern. Because there is no absence item immediately prior, when TPS finds an alternative match for the i^{th} item in backtracking, the algorithm does not need to check whether this alternative match would violate an existing constraint. Π is not necessary, but makes this narration easier.

We introduce the term *absence block* in a search pattern. An absence block is a contiguous block of absence event items in the search pattern, generally surrounded

by presence events on either end (except when the block is the leading or the ending part of the pattern). For example, the pattern $P = T_{p_1}\bar{T}_{p_2}\bar{T}_{p_3}T_{p_4}\bar{T}_{p_5}T_{p_6}$ consists of two absence blocks $\bar{T}_{p_2}\bar{T}_{p_3}$ and $\bar{T}_{p_5}$ and , of size 2 and 1 respectively.

When the current item in the pattern $P[\chi]$ is an absence item, `HANDLE_ABSENCE(R, P)` (Code Listing 4.2) is called. In lines 21-28, TPS looks ahead in the pattern to determine what events are in the absence block. For each event type in the absence block, the algorithm finds the next event of that type in the record, and saves the minimum time stamp value over all such events as `minTime`. If `minTime` does not exist, then that means there are no events of these types in the record, and TPS can safely increment χ and remember that these events do not exist by saving this fact in Φ (lines 30-34). Otherwise, TPS looks ahead for the next presence item, finds a matching presence event in the record and compare its time stamp with `minTime`. If it occurs before `minTime` (no violation), then we can move to the next item in the pattern and update our states (lines 48-53). However, if it occurs after `minTime`, we have to backtrack (lines 42-46).

```
18  HANDLE_ABSENCE(R, P)
19    minTime ← NIL
20    numAbs←0
21    for (i ← χ to P.length)
22      item ← P[i]
23      if (NOT item.isNeg) break
24      numAbs ← numAbs + 1
25      absEvent ← NEXT(R[item.Type], τ)
26      if (absEvent == NIL)
27        continue
28      minTime ← MIN(absEvent.TIME, minTime)
29    if (minTime = NIL)
30      if (δ > −1)
31        Φ[δ]← ∞
32      else
33        Φ[Φ.length-1]← ∞
34      χ ← χ +numAbs
35    else
36      if (χ + numAbs < P.length)
37        nItem ← P[χ+numAbs]
38        nEvent ← NEXT(R[nItem.TYPE], τ)
39        Π[χ+numAbs] ← TRUE
40        if (nEvent = NIL OR
41            nEvent.TIME > minTime)
42          χ ← δ
43          if ( χ  > 0 )
44            δ ← Δ[χ]
45            τ ← minTime-1
46          β ← true
47        else
48          T[χ + numAbs] ← nEvent.TIME
49          Φ[δ]←minTime
50          Δ[χ+numAbs]← χ-1
51          δ ← χ+numAbs
52          χ ← χ+numAbs+1
53          τ ←nEvent.Time
54      else
55        χ ← δ
56        δ ← Δ[χ]
57        τ ←minTime-1
58        β ←TRUE
```

Code Listing 4.2: the HANDLE_ABSENCE function that deals with absence items in the sequence.

When we backtrack, TPS set β to `TRUE` to indicate that it is in backtrack mode. When in backtrack mode, any subsequently matched item will require an additional check to see if a prior constraint is violated. For example, in Table 4.4, the C in pattern $P = A\bar{B}C\bar{D}\bar{E}F$ is originally matched to $R[1] = C$ in stage 2. A backtrack occurs in stage 3, forcing a C to be matched with $R[4]$ in stage 4.

However, making that choice violates $A\bar{B}C$ because $R[3] = B$, causing another backtrack. This is the reason why this additional check is required in backtrack mode. When a violation like this occurs, TPS sets the local variable `backtrackMore` to be `TRUE` in `HANDLE_PRESENCE(R, P)`, line 71. This makes TPS to backtrack additionally to $P[0] = A$ in stage 5 in Table 1, which eventually leads to matching $P[0]$ to $R[5] = A$. Code listing for `HANDLE_PRESENCE(R, P)` shows how TPS deals with presence items and, additionally, backtracks when β is true.

```
56  HANDLE_PRESENCE(R, P)
57    event←NEXT(R[P[χ].TYPE], τ)
58    if (event←NIL)
59      χ ←-1
60    else
61      backtrackMore←FALSE
62      if (β AND Π[χ])
63         if (Δ[χ] < 0)
64            badTime← Φ[Φ.length-1]
65         else
66            badTime← Φ[Δ[χ]
67         if (badTime < event.TIME)
68            χ ← Δ[χ]
69            τ ← Φ[χ]-1
70            δ ← Δ[χ]
71            backtrackingMore←TRUE
72      if (backtrackingMore) return
73      τ ←event.TIME
74      T[χ]←event.TIME
75      Δ[χ]← δ
76      δ ← χ
77      χ ← χ+1
78      β ←FALSE
```

Code Listing 4.3: the HANDLE_PRESENCE function that deals with presence items in the sequence.

When handling a presence item, TPS finds the first occurrence of the given event type after the current time τ (line 57). If there is no such event, TPS immediately sets the χ to -1 (lines 59), which subsequently causes `IS_A_MATCH(R, P)` to return `FALSE`, terminating the search. If there is a match and TPS is in backtrack mode, the algorithm checks to see if the match violates a previous constraint (line 63). This check is performed by comparing the matched event's time with the time of the closest known absence event in the future (`badTime`) that follows the match of the previous presence item (line 67). If the violation exists (matching event time is

greater than `badTime`), TPS sets `backtrackingMore` to `TRUE`, and updates several variables to backtrack more (lines 68-71). If there is no violation (or that we are not in backtrack mode to begin with), TPS updates the variables to advance on the pattern (lines 73-78): current time τ is updated to the newly matched event's time, and this new time is also stored in T for future reference. The current index χ is stored in last index δ, and Δ is accordingly updated. χ is incremented by 1 to advance on the pattern. Finally, β is set to `FALSE` to exit backtracking mode.

4.4.2 An Example

Table 4.4 shows a trace of TPS searching for the pattern $P = A\bar{B}C\bar{D}\bar{E}F$ in the record $R = ACEBCACDCF$. Again, note that I am using an array notation to describe R for ease of narration even though R contains a set of sorted arrays of historical events. Each event in R is associated with a time stamp.

The first column shows the iteration number of the main loop in Code Listing 4.1. The second column shows the value of the two most important variables χ and τ at the beginning of each loop. The third column shows the events in R that are being matched with item(s) in P in that loop. The fourth column indicates which lines of code are reached to assign the values to the state variables in the right portions of the table.

The first row of Table 4.4 shows the 0^{th} loop, where all variables are initialized. I describe the entire trace textually here, but focus on loops 1-6 in detail. Loops 1 and 2 demonstrate how presence and absence items are handled if there are no

backtracks. Loops 3 and 4 are examples that show how negation items can trigger multiple backtracks. Loop 6 shows how TPS handles negation items when there are no such events in the record.

In the 1^{st} loop, TPS attempts to find a match for the first item in the pattern $P[0] = A$. The event A at time 0 is matched. The current index χ on the pattern is incremented to indicate that we will try to match the second item in the pattern. The current time τ, from which the next search will begin, is also updated to 0, the time stamp of the matched event. δ is updated to remember the index (0) of a previously-matched presence item on P. β is kept as `FALSE` as the algorithm does not need to backtrack.

The 2^{nd} loop shows how TPS deals with an absence event in $P[1] = \bar{B}$. TPS uses binary searches to find the first event for each event type in the absence block (in this case there is only one ($\bar{B}$)) and the first presence event (C) after this contiguous absence block. Because the first B TPS finds in $R[3]$ does not violate the absence event criteria, no backtrack is necessary (β is unchanged). TPS remembers that the time stamp of the first B is found in $\Phi[0]$, where 0 is the index of the previous presence event. $\Pi[2]$ is set to `TRUE` to indicate that the item $P[2]$ follows at least one absence item.

In the 3^{rd} loop, TPS again deals with an absence block and its following presence event: $\bar{D}\bar{E}F$. This time, however, because both the first D and E occur between the previously matched $P[2] = C$ and the upcoming $P[5] = F$, β is set to `TRUE` to indicate that TPS will backtrack. χ is rolled back to 2 so TPS will find a new match for $P[2]$. τ is set to be right before the time stamp of the first known

absence event of the absence block. In this case, it is $R[2] = E$, which has time stamp of 20. TPS sets τ to be 20 - 1 = 19, where 1 is the smallest time granularity in consideration (usually milliseconds). This way if a C in the record has the time stamp of 20, the binary search functions would not miss it.

In backtracking mode in the 4^{th} loop, TPS attempts to match $P[2] = C$ to $R[4]$. Since $P[2]$ trails an absence block, TPS checks to see if matching $P[2]$ to $R[4]$ violates a constraint. In this case, it does violate $A\bar{B}C$, for there is a B in $R[3]$. This means TPS needs to backtrack again, to $P[0]$. To do this, TPS decrements χ, but increments τ to right before the time stamp of $R[3]$ (29).

TPS finds a match for $P[0] = A$ in $R[5]$ in loop 5 and updates other variables normally as in loop 1, and sets β to `FALSE`. When processing $P[1] = \bar{B}$ in loop 6, TPS finds no B's in R after the current time $\tau = 50$. This means the constraint of $A\bar{B}C$ will not be violated for any C that occurs after. This fact is stored in $\Phi[1]$. χ is of course incremented by 1. Because the lack of B in the record, TPS only processes the absence block that contains $P[1]$ in this loop. The following presence event is left for the next loop.

In the 7^{th} loop, TPS finds $R[6] = C$ to match $P[2]$. However, in the 8^{th} loop, TPS finds a $R[7] = D$ between the previously found C and the next available F in $R[9]$. This forces TPS to re-search for another match for C. Variables are updated exactly in the same manner as in loop 3.

Fortunately, an alternative is found in $R[8] = C$ in loop 9, and this match does not violate any existing constraints. In loop 10, TPS fails to find another E or D in the remainder of the record. Finally, TPS matches the last item in the pattern

$P[5] = R[9]$, and returns `TRUE`, terminating the execution.

4.5 Analysis

If the search pattern contains no negation: $P = p_1 p_2 p_3 \ldots p_m = T_{p_1} T_{p_2} T_{p_3} \ldots T_{p_m}$, its running time is $O(lg(|T_{p_1}| + lg(|T_{p_2}|) + \ldots + lg(|T_{p_m}|)$, where $|T_{p_1}|$ denotes the number of events of type T_{p_1} in a personal record. This is because the algorithm uses binary search to find an event when a presence item is encountered. If there are no events to be found, the algorithm immediately terminates and returns `FALSE`. In contrast, if such a pattern exists, then the running time is bounded by $O(mlg(n))$, where n is the total number of events (regardless of type) in the record.

When the search pattern contains absence items, TPS may need to perform backtrack when absence events are encountered, and the worst case analysis is based on (1) the possible number of backtracks, and (2) the amount of computation done when a backtrack occurs. In general, the input data trigger a backtrack when an *absence block* in the pattern is not satisfied, *e.g.*, a T_{p_2} occurs between a T_{p_1} and a T_{p_4} for the pattern $P = T_{p_1} \bar{T}_{p_2} \bar{T}_{p_3} T_{p_4} \bar{T}_{p_5} T_{p_6}$. The maximum number of absence blocks in the search pattern is then $\frac{m}{2}$, where m is the length of the pattern. The algorithm works so that when an event that violates an absence block specified in the pattern, the event's time is recorded in the array Φ, and the algorithm will start at that event's time when backtracking. For this reason, we only have to consider non-overlapping presence-absence events in the data as potential backtrack points from the data. Therefore, for a record containing n events, there can be at most

$\frac{n}{2}$ non-overlapping presence-absence event patterns, where 2 is the length of the smallest possible presence-absence pattern. For each such pattern, it can cause up to $\frac{m}{2}$ backtracks. Putting everything together, the total number of backtracking opportunities is $\frac{m}{2} \cdot \frac{n}{2} = \frac{mn}{4}$.

The amount of extra computation resulting from a backtrack is related to the size of the absence blocks. The pseudo code indicates that each backtrack results in a constant number of variable assignments, but the real work lies in the occasions when TPS re-performs its search on the failed part of the pattern. A backtrack is always triggered by an absence block, and a new search is performed to find a matching presence event immediately before the absence block, all absence events in the absence block, and the trailing presence event (if there is one specified in the pattern). This means $O((2 + |ab|)lg(n))$ work is performed, where ab is the size of the largest absence block. This expression is maximized when $|ab| = m$, resulting in $O(mlg(n))$, where m is the length of the search pattern.

The worst case running time would then be the number of backtracking opportunities times the work done in a backtrack situation. However, a search pattern cannot both have the largest absence block and the highest number of absence blocks at the same time. Let m be the size of the search pattern, and x be the largest *absence block* size. We want to find x that maximizes the running time function $f(x) = (\frac{n}{2} \cdot \frac{m-x}{2}(xln(n)))$, where m and n here are considered constants. The second derivative $f''(x) = -1$ shows that $f(x)$ is concave down. By the second derivative test, and the fact that its first derivative equals to zero when $x = \frac{m}{2}$, $x = \frac{m}{2}$ must be a global maximum. This means that the worst case patterns are the

ones that have a largest absence block equaling half the length of the search pattern, and the other half of the pattern consists of only alternating (positive-negative) patterns to increase backtrack opportunities. Substituting $x = \frac{m}{2}$ in $f(x)$, we obtain the worst case bound $O(\frac{m^2 n}{8} lg(n)) = O(m^2 nln(n))$. In addition, TPS uses $O(m)$ space for the state-keeping arrays.

4.6 Algorithm Extensions

The basic TPS algorithm presented here is amenable to handle additional search constraints akin to many of the extensions in the finite automaton approaches. Here I present a straight-forward modification to TPS that handles two common constraints: temporal range constraints and value range constraints.

Temporal range constraints specify how an event must temporally relate to others. Each item on the temporal pattern can have multiple temporal constraints. For example, for the pattern $P = p_1 p_2 p_3 \ldots p_m$ we can additionally specify that p_3 must occur between 5 to 10 days after p_1 and within 3 hours after p_2. It suffices to consider only temporal constraints related to previously matched items, as constraints related to yet-to-be matched items can be converted to this form. It is worth noting that the temporal constraints work for both positive and negative items. When temporal constraints are used on a positive item, they specify the only time spans when the events can match. When temporal constraints are used on a negative item, they specify the time spans when event must *not* match.

The value range constraints, on the other hand, specify that the matching

events must have values within a certain range in order to be considered a match. For example, physicians may look for patients who had a heart attack followed by heart surgery followed by a systolic blood pressure reading of 140 or greater. More complex value range constraints can involve higher dimensional data, or data values relative to previously matched items.

These constraints change how TPS works because matching on event types is now conditional on their time stamp and values. The basic idea to handle these constraints is to selectively filter the events that can be matched in the steps of the algorithm. Since he basic structure of TPS guarantees the correct backtracking, the simplest way to handle these constraints is to extend `NEXT(R[T],t)`. That is, after a binary search is per-formed on an event type array A, and an index is determined, the temporal constraints and the value constraints for that item are checked. If the first event fails the check, subsequent events on the array can be tested until one passes or no more events can be tested.

Code Listing 4 shows the modified function, where M is the array of currently matched events. D and V are, respectively, arrays representing the temporal range and value range constraints. The modified function requires the binary search function as before, and additionally functions that check whether an event passes the constraints. This direct extension makes the worst case bound $O(m^2n^2\alpha)$, where α is the amount of work needed to run `DURATION_CHECK(‘)` and `VALUE_CHECK(‘)`. These functions can have high complexity as there can be at most $i-1$ duration constraints for each i^{th} item on a pattern of length m. Additionally, the values may be multidimensional or of other complex data structures.

When the event type structure is a hierarchy instead of a list and search patterns include supertypes (types that have descendants), the `NEXT(R[T],t)` function needs to be further extended to use binary search on each of the subtype's array, and return the one closest to current time t. Caching of the results of the binary searches can improve performance. Since each `NEXT(R[T],t)` can take up to $O(klg(n))$ to perform, the over all worst-case bound for TPS becomes $O(km^2nlg(n))$, where k is the number of event types.

4.7 Evaluation

There are several regular expression implementations we considered comparing TPS to. I first tested the standard Java regular expression engine in `java.util.regex`, and a faster (though more limited) implementation that uses deterministic finite automata: (`dk.brics.automaton`) [69]. However, initial tests revealed that the Java regular expression engine runs in time exponential to the length of the search pattern in the worst case. I used a fixed input string of 500 characters, and search patterns of varying length ($m = 2, 3, 4, 5$). The string is chosen so that the patterns will not match. The regular expression patterns are obtained by converting positive-only patterns described in our discussions to its equivalent regular expression form. For example, $ABCBD$ is converted to `.*A.*B.*C.*B.*D.*`. Table 4.3 illustrates this exponential behavior of matching positive-only patterns (no negation) of varying length against a *single* string of size 500. This behavior is due to the fact that each `.*` presents a choice point for `java.util.regex`, and combinatorially, there

Search Pattern Length	Time (seconds)
2	0.024
3	0.272
4	31.193
5	3099.675

Table 4.3: Using `java.util.regex` to perform searches exhibits exponential behavior. The time reported is time taken to match a pattern over a single string. The input string size is 500, while the search pattern length varies from 1 to 5. The strings are chosen so that there is no match for the patterns, thus requiring `java.util.regex` to exhaust all possibilities before returning false. This behavior is due to the fact that the patterns contain many `.*`, which can require exponential number of searches.

are $O(2^m)$ choice points in a search pattern consisting of all positive items of length m [19]. For the particular application and type of patterns we consider, `.*` is unavoidable, and occurs very frequently (grows linearly with respect to the number of items in the search pattern in the worst case). Therefore the very poor performance of `java.util.regex` means that this commonly used strategy makes our TPS algorithm seem remarkably efficient. It is worth noting that popular regular expression engines such as the one in perl also suffers from the same problem [19].

The DFA approach, on the other hand, has a similar problem. As long as the search pattern sizes are kept reasonable, the performance is very good. However, when the search pattern is longer than 30 items, the exponential space the DFA requires makes the approach infeasible (it fails to construct the DFA when all memory has been allocated to the Java Virtual Machine).

Instead, we compared our Temporal Pattern Search algorithm to more scalable approaches. We include, first, a simple NFA that handles the wild card character (.), Kleene Star (*) over single symbols, and negation over a set of symbols (`[^xyz]`),

where `xyz` are symbols) (Figure 4.2), and secondly, the bit-parallel Shift-And algorithm described in [71].

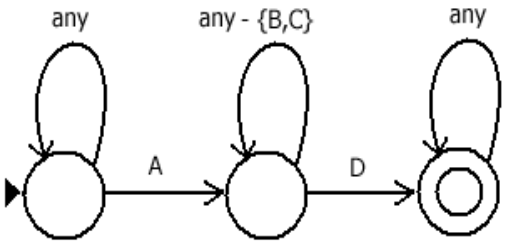

Figure 4.2: An NFA corresponding to `.*A[^BC]*D.*`, which is the regular expression equivalent to the search pattern $A\bar{B}\bar{C}D$. The black triangle indicates the starting state, and the double circle indicates the accepting state. *any* represents the set of all characters

The NFA approach keeps a set of states that it is currently in. Initially only the start state - indicated by the black triangle in the figure - is in the set of current states. As the NFA reads one symbol at a time from the input string, it checks with all the current states it is in and sees if any of state's transition rules fire to bring in a set of new current states. When all inputs are consumed, the NFA checks to see if the accepting state (indicated by the double circle in the figure), is in the set of current states. If it is, then the NFA returns `TRUE` to indicate that a match has been found; `FALSE` otherwise. For a search pattern of length m, there are at most m states in its corresponding NFA (each positive term translates to a state, and each negative block, regardless with length, also translates to one state). For an input string of size n, the running time for the NFA is bounded by $O(mn)$ because the NFA may be in all m states for each of the n symbols.

Due to the limited expressiveness of the search patterns under consideration here, the NFA does not need to wait until all inputs have been consumed to check for the accepting state. In our implementation, the NFA checks for the accepting state after each symbol has been consumed, except when the last item on the pattern is a negations (in which case, the NFA does need to consume all inputs to ensure that there are no violations). This small optimization increases the NFA performance by a significant amount in practice.

The Shift-And algorithm used here differs from [71] in that it models the first state so that the first state is not assumed to be active at all times (needs to be inactive when the pattern starts with a negation, and that negated event is encountered in the search). Since we are using Java Runtime Environment in 32-bit, the length of the computer word $w = 32$. The Shift-And used here is extended to handle patterns that produce more than 32 states. The extension uses an array of integers to represent more than 32 bits and handles the necessary bit operations (shift, and, or, xor, negation) correctly over the array. The extended version, however, also carries an overhead for the array representation. In the evaluation below, when the number of states for the Shift-And algorithm remains under or equal to 32, the original algorithm is used. When the number of states is greater than 32, the extended algorithm is used. When the number of states for Shift-And is less than or equal to 32, the search algorithm performs in O(n). If it greater than 32, it performs in $O(mn/32)$. Like the NFA implementation described above, the Shift-And algorithm also terminates immediately when a match has been found.

4.7.1 Method

The NFA and Shift-And most naturally work with a temporally ordered string of events. TPS works over a set of type-separated, temporally ordered arrays. The aim of this evaluation is to see if we had the most appropriate data support for each approach, whether TPS has any advantage over NFA or Shift-And. That is, I attempt to answer the question "if I had implemented our visualization tool in the most appropriate way to use an NFA (or Shift-And) search, would it perform faster?" The evaluation is structured as follows. (1) It is guaranteed that the input event history data does not have events that have the same time stamp. This means the NFA or Shift-And do not need to be modified to handle this case, and that the input data will be semantically equivalent in the evaluation. (2) The input data are transformed into a single string for the NFA and Shift-And approaches to process. (3) In all the reported results, the preprocessing time is not included. That is, the time used to build the NFA and the bit masks for Shift-And is not included. The time required to build an NFA is $O(m)$, and the time required to for Shift-And preprocessing is $O(mk/w)$, where k is the number of event types, and w is the number of bits in a computer word. The preprocessing time is fairly light, although it would still be reflected in the interface when analysts construct a pattern query.

Let k designate the number of event types in a set of records. I randomly generated a set of 5000 records for each $k = 2, 3, \ldots, 9$ and $k = 10, 20, 30, 40, 50$. Each record in the set has 500 events. Each event in a record occurs with uniform probability. I also randomly generated a set of search patterns for each dataset, where

each event type occurs in the pattern also with uniform probability. In the search pattern input file, there are three types of patterns that are generated: (1) patterns that contain only positive items, (2) patterns that contain alternating positive and negative item (half with leading positive terms, half with leading negative terms) and (3) patterns that represents the theoretical worst-case scenario, given length of the pattern m and number of the event types k. These worst-case scenario patterns are like the alternating patterns, but with a large absence block of size $min(k, \frac{m}{2})$ inserted at a randomly chosen position according to a uniform distribution. These patterns are designed to elicit the worst performance in TPS.

For each input set of records, the script parses the records, constructs the appropriate data structures for TPS, NFA, and Shift-And, and constructs the NFA and Shift-And supporting data structures. Clock is then set to collect only the search time for each approach. Each pattern for each dataset is performed 10 times for each approach. The average is then taken and presented here. The performance numbers are obtained by running our script in Java 1.6- 32-bit environment on Windows Vista 64-bit with a dual core CPU (2.5GHz) and 4GB of RAM.

4.7.2 Results

The results are examined from two perspectives. First, I inspect how the length of a search pattern impacts the performance of the three algorithms. I do this by summing up the time taken to execute all patterns of one type across all alphabet sizes and compute the average time it takes to perform a search on all

records.

Figure 4.3 shows the performance graph when all search patterns include only positive items. The x-axis is the length of the search pattern, and the y-axis is time in seconds and on logarithmic scale. First, TPS performs consistently better than both NFA and Shift-And. TPS is more than one order of magnitude faster than NFA, and nearly one order of magnitude better than Shift-And when $m > 15$. In the Shift-And algorithm, each `.*` is modeled as a separate state, so a pattern length of m in the all-positive patterns has $2m$ states. This is why the performance hit occurs when $m > 16$, as opposed to $m > 32$ as in other pattern types. The sharp decrease in performance represents the overhead required to support arbitrarily long search patterns. Secondly, the running time increases as the length of the pattern increases. I had expected the TPS to outperform NFA on the positive-only pattern portion of the evaluation as TPS's search over positive-only events is bounded by $O(mlg(n))$, that is, this is the "best-case" scenario. I was a bit surprised by its (slight) advantage over Shift-And when the number of states was fewer than 32, given that Shift-And runs in $O(n)$ in these cases.

Next, I look at the results for the *alternating patterns*. Unlike the *positive-only* patterns, these patterns provide many opportunities for backtracking, and I expected to see the TPS performance to be worse than the previous graph. Figure 4.4 shows the performance graph over a variety of search pattern lengths. The patterns with odd lengths are "easier" for all three approaches. They all can find those pattern faster. The fluctuation in performance reduces as the length of pattern increases. Shift-And has the smallest fluctuation, while TPS has the largest, enable

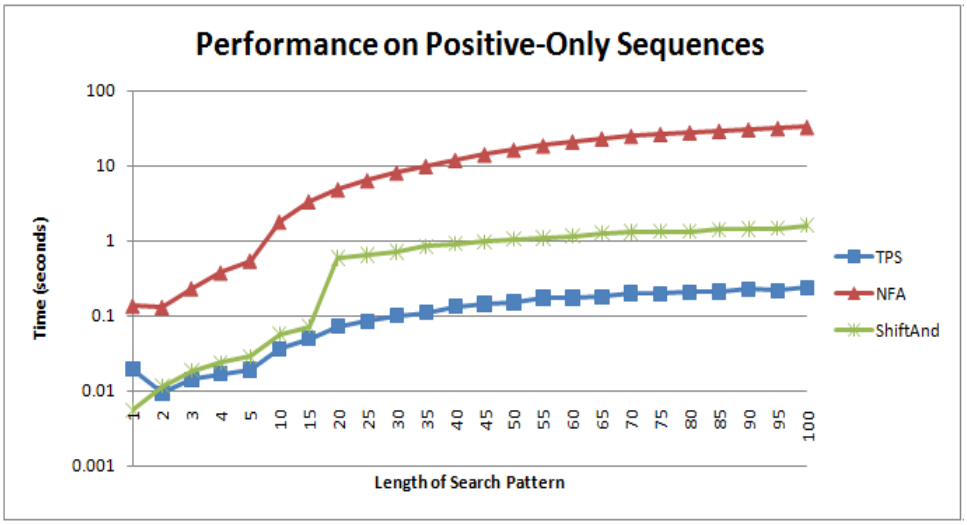

Figure 4.3: Comparison of the performance of TPS, NFA, and Shift-And in search pattern length $m = 1, 2, 3, 4, 5, 10, 20, \ldots, 100$ for positive-only patterns. The vertical axis is logarithmic in time. TPS outperforms the other two approaches in general (except when $m = 1$, and betters NFA consistently by one order of magnitude. The sharp increase of time required at $m = 20$ for Shift-And reflects the use of using multiple integers to represent more than 32 states.

it to outperform Shift-And in these odd-lengths patterns. Despite the fluctuation, it is clear that TPS still consistently outperforms NFA, but the gap is often less than an order of magnitude. The Shift-And approach generally outperforms TPS for shorter patterns ($m \leq 32$). However, TPS outperforms Shift-And for the small patterns with odd lengths. All three performance lines grow much slower than the positive-only sequences.

In the *worst-case patterns* scenario, we expect TPS to lose most of its advantage while NFA performs similarly to the alternating patterns scenario. NFA does indeed perform similarly to the *alternating patterns* scenario, except with an initial growth when $m = 10, 20, 30$ parts, which TPS also experienced (Figure 4.5). In

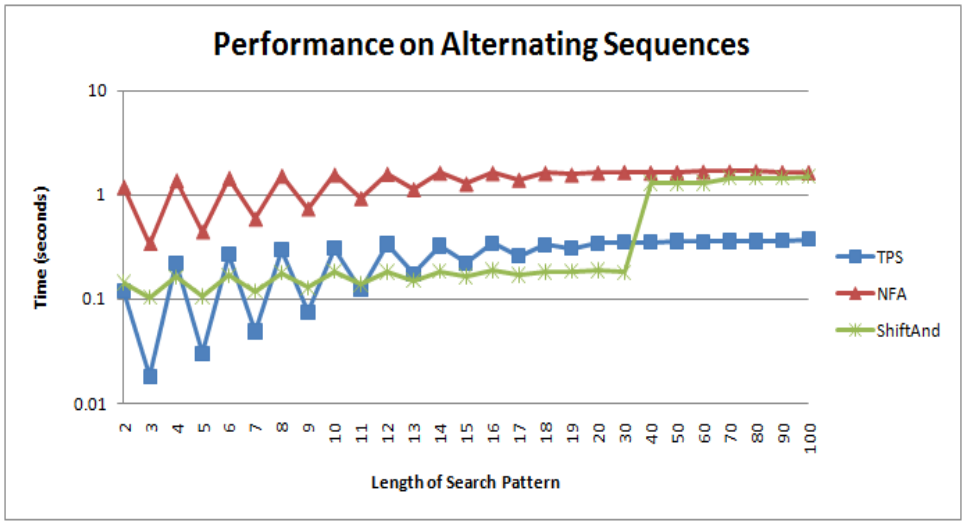

Figure 4.4: Comparison of the performance for TPS, NFA, and Shift-And in search pattern length $m = 2, \ldots 20, 30, 40, \ldots, 100$ for alternating patterns. The vertical axis is logarithmic in time. Similar to the case of positive-only patterns, TPS consistently outperforms NFA for these search patterns by roughly an order of magnitude. However, the Shift-And approach is, in general, faster when the m is less than or equal to 32. For very small size of m, the results are more sensitive to the randomly generated patterns. All three approaches perform much better with $m = 3, 5$. TPS is much more sensitive to this than the other two approaches, and as a result, had the most performance gain.

this scenario, TPS's performance gets closer the NFA's, taking nearly one second to perform a search over all the records. However, it still beats NFA in all of our test cases, and we do not observe a growth adhering to the m^2 term in the worst-case bound. Similar to the alternating cases, the Shift-And approach is very fast for m up to 30. However, it loses its performance advantage to TPS with $m > 40$.

To study why TPS is performing relatively well when its worst case running time is comparatively bad, I plotted the average time it takes to perform a search over all records when the number of event types is varied. Since TPS examines an

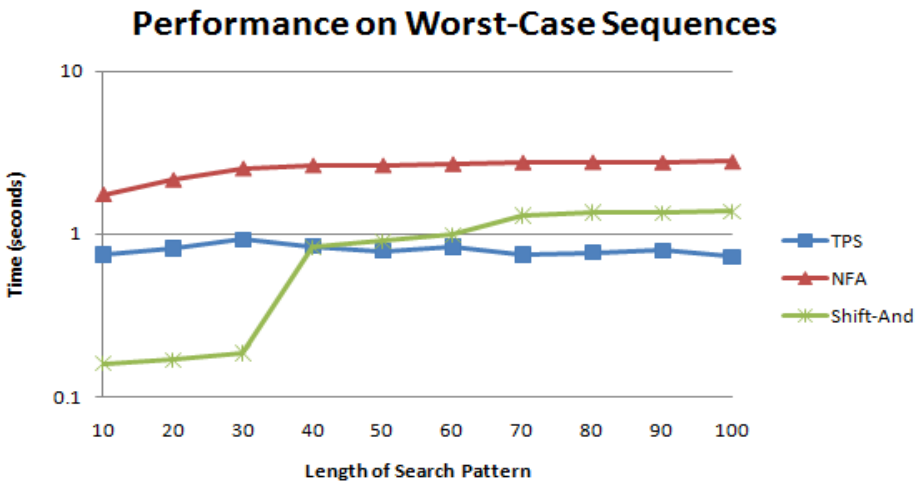

Figure 4.5: Comparison of the performance of TPS, NFA, and Shift-And in search pattern length $= 10, 20, ‘, 100$ for worst-case patterns. The vertical axis is logarithmic in time. Like two previous cases, TPS consistently outperforms NFA, but the differences are considerably smaller than in the comparison for positive-only patterns. Shift-And handily out-performs the other two for $m < 40$, but loses to TPS (significantly for $m > 60$) otherwise.

event on a need-to basis, the event type size does not affect the asymptotic bound. However, if the number of events is large compared to the number of event types in the search pattern, TPS can ignore most of the events. On the other hand, both NFA and Shift-And are required to, in general, examine all the events. I focus on the two more difficult cases: *alternating* and *worst-case*. In Figure 4.6, initially TPS and NFA have similar performance ($m = 2$). As k increases, both NFA's and Shift-And's performances remain roughly in the alternating patterns scenario. On the other hand, TPS's performance steadily improves as k increases. Figure 4.7 shows a similar trend with worst-case patterns scenario. The difference here is that the Shift-And approach also enjoys some performance gain when k grows past 9,

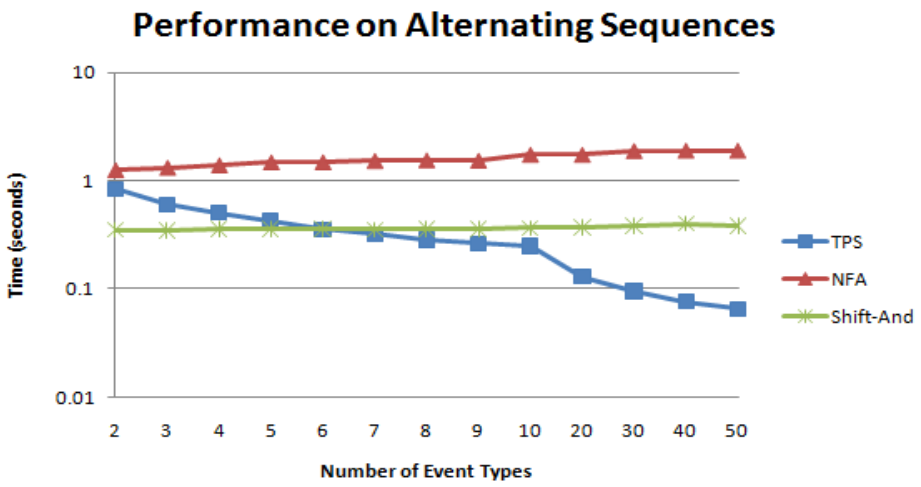

Figure 4.6: Comparison of the TPS and NFA performance in number of event types k = 2, 3, ‘, 9, and 10, 20, ‘50 over all lengths of search patterns for alternating patterns. The vertical axis is logarithmic, and it is time in seconds. TPS performs similarly to NFA initially, but gets dramatically better than both NFA and Shift-And as the number of event types grows.

though not as dramatic as TPS's.

On the flip side, if TPS is required to look at all of the events, TPS is not expected to perform better than NFA. To show this, I fix the search pattern to be $A\bar{B}C$ ($m = 3$) and the events in records to be $ABABAB \ldots ABC$. This combination results in a search that requires TPS to look at every event in the record, creating a very large number of backtracks, and eventually returning `FALSE`. I vary $n = 100, 200, \ldots, 1000$ to show the effects of n. This experiment is performed for $m = 3, 5, 7, 9, 11, 31, 41, 51, 61$. Figure 4.8 shows the average performance for $m = 3, 5, 7$ for the three approaches on 5000 records for each n. n affects the running times linearly for $n = 100, \ldots, 1000$. TPS and NFA have identical performance, while

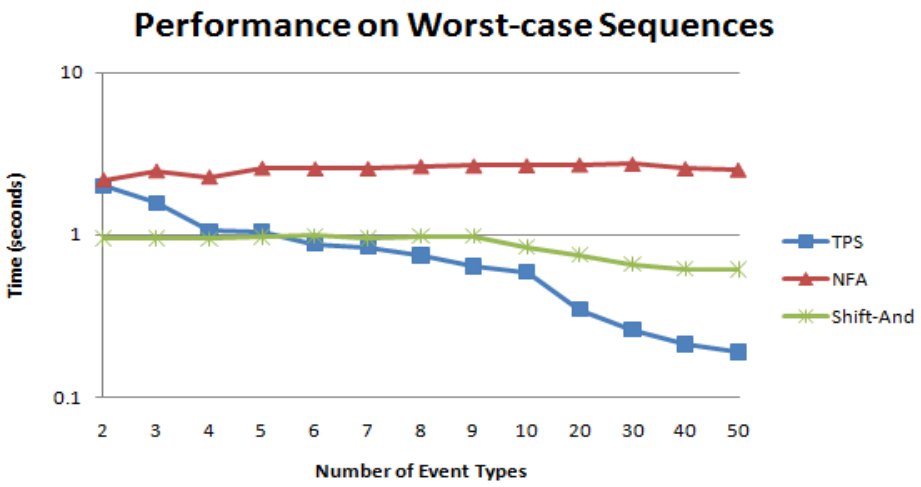

Figure 4.7: Comparison of the TPS and NFA performance by varying number of event types $k = 2, 3, `, 9$ and $10, 20, `, 50$ over all lengths of search patterns for worst-case patterns. The performance graph is similar to Figure 4.6, with the only exception that Shift-And also improved in performance as the number of event types increases past 9.

Shift-And outperforms them by an order of magnitude. When the same experiment is performed for larger m ($m = 41, 51, 61$), Shift-And's performance edge becomes very small (Figure 4.8).

4.7.3 Evaluation with Real Scenarios

The experiments with random data above show the strengths and weaknesses of TPS. They give a reasonably clear picture on the situations where TPS is expected to perform well or badly. However, the evaluation would not be complete if these algorithms are not applied to real scenarios. I used the real datasets listed in Table 4.1 and the real search patterns our physician collaborators are interested in to evaluate the three algorithms. The datasets in Table 4.1 contain events that share

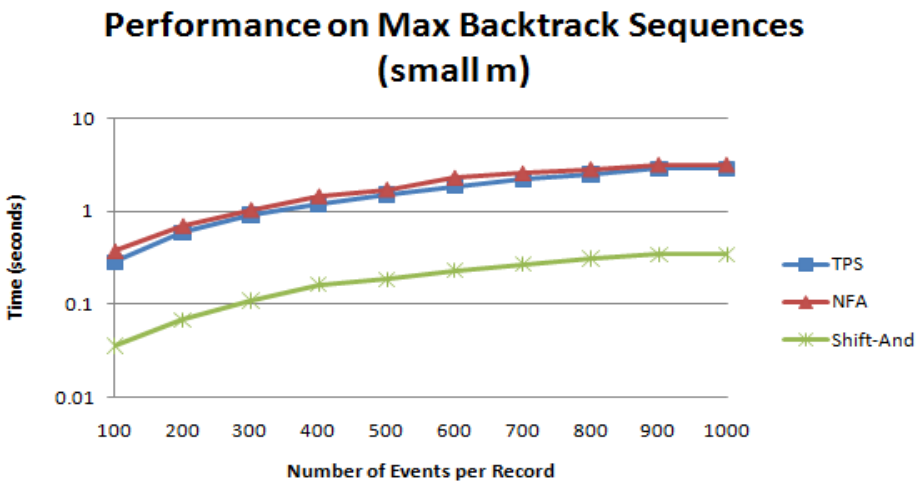

Figure 4.8: Comparison of the performance of TPS, NFA and Shift-And by varying the number of events in a record $n = 100, 200, \ldots, 1000$. The search pattern is fixed to be . The values presented is the average of the pattern with $m = 3, 5, 7$. The records are fixed to $ABABAB`ABC$. This guarantees $n/2$ backtracks for record. There are 5000 records for each n. The vertical axis is logarithmic in time. TPS and NFA have similar performance, while the Shift-And algorithm outperforms the other two by an order of magnitude.

the same time stamp. I had to modify these datasets to ensure that these events do not exist since neither the NFA nor the Shift-And approaches can handle them. I followed the same procedure as described in Section 4.7.1 for data conversion and timing.

Table 4.4 shows the number of temporal patterns our collaborators are interested in for each dataset, the average length of the patterns, and how much time each algorithm took to perform. We did not perform the evaluation for one of the datasets (Heart Attack) in Table 4 because our users had no temporal patterns they wanted to search for. TPS outperformed the other two approaches in all but

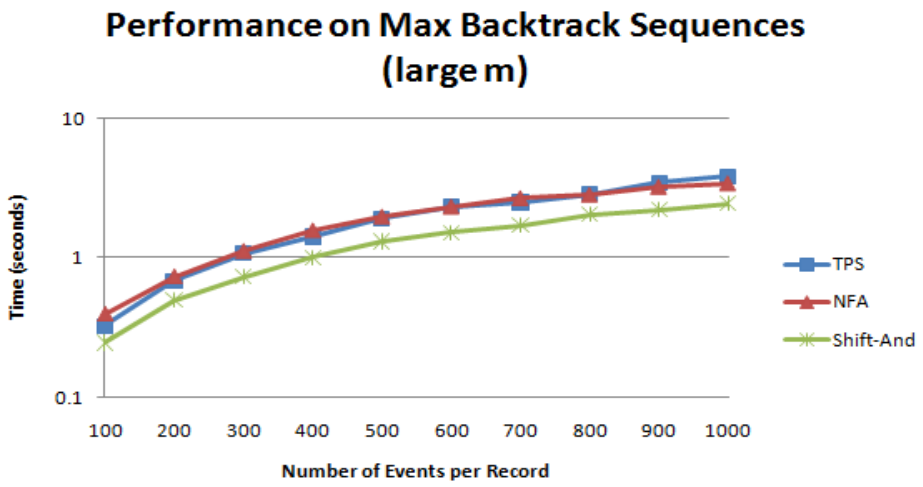

Figure 4.9: Same comparison as Figure 4.8, except that the values represent the average run of large pattern length: $m = 41, 51, 61$. The advantage of Shift-And is far less prominent when m is large.

Dataset Name	**Patterns**	**Average Pattern Lengths**	**Time (milliseconds)**		
			TPS	NFA	Shift-And
Creatinine	5	3.2	51	325	20
Heparin	5	4.6	12	26	20
Heart Attack	-	-	-	-	-
Transfer	5	3	93	981	99
Bipap	9	3.89	66	711	82

Table 4.4: Selected Real Datasets, Their Statistics, and Algorithm Performance

the Creatinine dataset, even when the length of the patterns is small, where the Shift-And algorithm has its biggest advantage in the randomized evaluation.

4.8 Discussion

While I have shown that the worst-case asymptotic bound for the TPS algorithm is worse than that of a NFA, the empirical evaluations show that TPS consistently outperforms the NFA approach under our experimental conditions with

randomly generated inputs. The experimental conditions are chosen to reflect the situations under which we expect users to be using Lifelines2. In working with our collaborators in the medical field, the average number of events in a single patient record usually are in the low hundreds, with the highest nearing a thousand, and lowest only a handful. The number of types of events is typically between 10-35. In practice, it is rare to see the cases where k is so small and TPS performs only marginally better than NFA. It is also rare when k is as large as 50 that TPS advantage exceeds more than one order of magnitude.

One the other hand, the very efficient Shift-And algorithm has great speed advantage when w is at most 32 and consistently out-performs TPS in the experiments with random data. When it is not, its performance is slightly worse than that of TPS on average. TPS also has a slight performance edge in the evaluation with real data and search patterns. While the NFA approach and the TPS approach are both easily extensible to handle additional constraints, the bit-parallel approaches are not. These approaches rely on bit masks that are built in the preprocessing stages so that when a symbol is read, the mask can be retrieved in constant time and utilized. However, when these additional constraints are present, the masks can no longer be pre-built. The mask that needs to be applied would be conditional upon the values in addition to each of the symbols read.

For $n \leq 1000$, the performance growth is roughly linear even in cases that the maximum number of searches and backtracks are required (Figure 9). We also do not observe a quadratic growth for $m \leq 100$, although the asymptotic bound for TPS contains a m^2 term. While I do not expect users to specify a pattern query of

these extreme lengths, it is comforting to know that if users wished to do so, TPS would be able to handle it.

The implications of these results are that while TPS may not be suitable for all situations, I argue it is the appropriate choice for our application Lifelines2. In addition, other analysis tools focused on temporal categorical data such as DataMontage [20], Event Tunnel [106], and SessionViewer [57] can benefit from TPS to provide sequential search capabilities. In addition, TPS can be used as an external function in database applications where searching for temporal patterns is required.

4.9 Summary

I present an extensible algorithm that is suitable to visualization systems such as Lifelines2 for temporal pattern search. The TPS algorithm works on the preexisting data structure of Lifelines2 – a record is a set of time-stamped-sorted data structures, where each structure is for each event type in the data. Even though TPS's worst-case running time is not as good as those of NFA or the bit-parallel approaches, its performance with a wide range of random data show that it performs better than NFA, and it is better than Shift-And when the number of patterns become larger than the size of the computer word. The extensibility of TPS allows the algorithm to expand to handle additional constraints like many other NFA approaches, which the bit-parallel approaches cannot handle. When experimenting with the real data, TPS performed better than Shift-And in three of the four data sets. With these favorable performance numbers, I argue that the TPS was the right

design decision for Lifelines2 and perhaps for other analysis tools.

Chapter 5

Supporting Hypothesis Generation: Temporal Summaries, Grouping, and Comparison

5.1 Overview

Generating new hypotheses is an exploratory and iterative process. It requires analysts to make generalizations about a dataset. However, making generalizations is particularly problematic when dealing with large numbers of personal records of event data, where each record holds many events, and some of which are of the same type. Often analysts must generalize both how events occur in relationship to others events (temporal ordering) and also the frequency of the event occurrences (prevalence). In doing so, analysts often compare datasets that differ in a single aspect (such as control vs. experimental) in an attempt to gather support for a hypothesis. Unfortunately, current analysis tools for temporal event data fall short in accentuating event temporal ordering and prevalence. They also fail at providing mechanisms for flexible and rapid comparisons.

While command-line query languages provide the means to access large amounts of temporal event data, temporal SQL queries are very difficult to specify, and the traditional tabular display of results neither highlights the temporal ordering nor exposes their prevalence. My previous work sought to address the temporal ordering

problem by using the Align-Rank-Filter (ARF) framework to manipulate the visualization of temporal event data. In particular, I showed in a controlled experiment that using alignment improves user performance greatly in recognizing temporal characteristics among events.

While alignment has been well-received by users, my ongoing collaboration with domain experts revealed the following needs in their daily work. They need to be able to specify temporal range constraints in their searches (*e.g.*, find all patients who have had an open-heart surgery within 3 months of their first heart attack). They need to view multiple records as an aggregate to study prevalence of event data over time. They need to divide and subdivide a set of records into logical groups and subsequently compare these groups.

In this chapter, I describe my solution - temporal summaries - and how they support these needs. Temporal summaries are stacked bar charts over a time frame. By default, events are the objects of aggregation, and each event type is depicted as a color-coded stack. A single temporal summary allows analysts to compare the distributional trends of multiple event types over time. Applying multiple temporal summaries on multiple groups of records allows analysts to compare these groups of records in a coordinated manner. As analysts zoom in and out, temporal summaries automatically rescale to display the aggregation in the corresponding granularity. Finally, temporal summaries provide affordances for analysts to specify temporal range constraints. This enhances the existing ARF framework, and enables more ways of creating logical sets of groups. The content of this chapter corresponds to one of my earlier publications [116].

5.2 Temporal Summaries

In Chapter 3, I demonstrated how Lifelines2 brings temporal ordering of event data to the analysts' attention via alignment of sentinel events. However, even with alignment, analysts are still burdened to visually scan the entire dataset to make sense of and to make generalizations about the data. To overcome this shortcoming, I added temporal summaries and appropriate controls ((b) and (d) in Figure 5.1). A temporal summary is a stacked bar chart that aggregates a variety of metrics (event counts, record counts, etc.) about a group of records over a time frame of varying granularities.

Consider the medical scenario *contrast-induced nephropathy*: some patients who undergo a radiology contrast experience adverse reaction and as a result, their renal functions decline. If not noticed and taken care of, the condition may be fatal. To monitor a patient's renal function, blood tests are performed regularly both before and after the radiology contrast to monitor the creatinine level. High creatinine indicates lowered renal function. The adverse reaction typically occurs within 14 days after a radiology contrast. This scenario was the primary scenario physician collaborators and I shared in developing the visualization and interaction features in Lifelines2. Here I use this same scenario as a concrete example to introduce the features of temporal summaries. I show how analysts utilize temporal summaries to see patterns, generate new hypotheses, and possibly change the direction of visual exploration. More specifically, I show how temporal summaries work with the existing ARF framework to find the patients who may have this adverse reaction. I also show

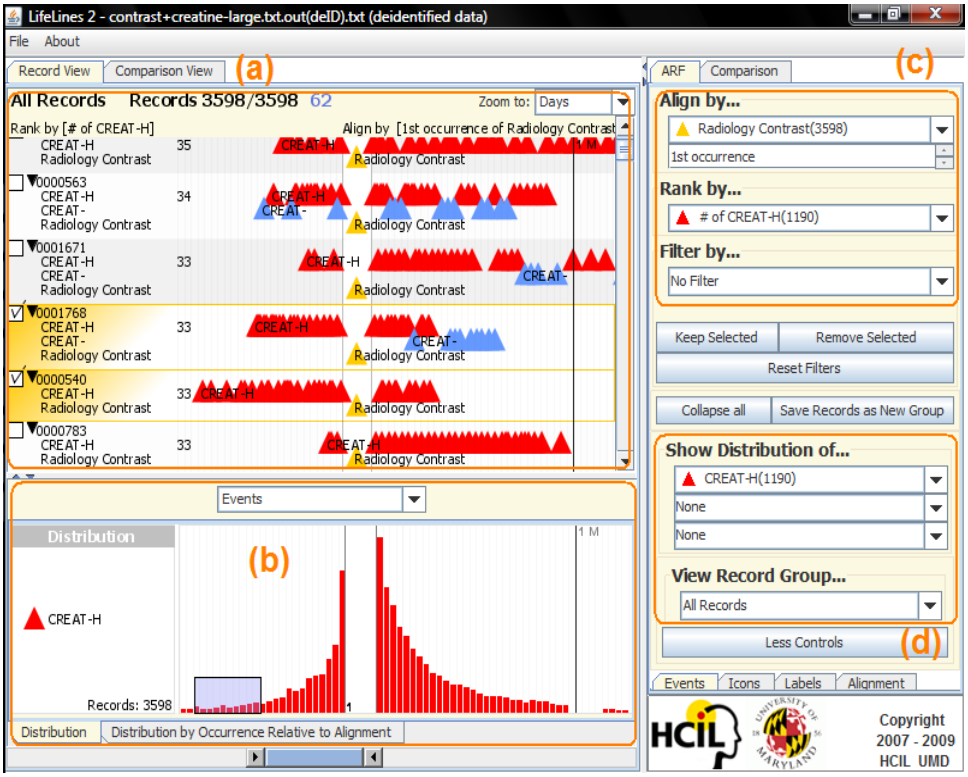

Figure 5.1: Annotated Lifelines2 screen shot. (a) The main visualization panel. A temporal summary is shown in (b). To the right is the control panel. (c) shows the controls and the current state of Align, Rank, and Filter. (d) shows additional controls for temporal summary, and also for navigating groups. Group creation and other controls are wedged between (c) and (d).

how analysts dynamically create subgroups of records, and use temporal summaries to perform comparisons among them. A video that covers most of the analysis presented here is accessible from `http://www.cs.umd.edu/hcil/lifelines2/` or directly [1].

[1] `http://www.cs.umd.edu/hcil/lifelines2/videos/Lifelines2-(Contrast-Creatinine)-11-10-08_flash/full(1024x768)/index.htm`

5.2.1 Integration with the Improved ARF Framework

The existing Align, Rank, and Filter framework allows analysts to visually align all records by a chosen n^{th} event (whether it is from start of the record or from the end), or by all occurrences of that event. When "all occurrences" is chosen, Lifelines2 duplicates records that have more than 1 such event. Analysts can rank records by number of occurrences of an event type. For example, Figure 5.1 shows that all records are aligned by the 1^{st} occurrence of *Radiology Contrast* and ranked by number of occurrences of *CREAT-H* (creatinine high) so that the records that have the most are displayed on top. The number next to each record's header on the left indicates how many *CREAT-H* events that record has. Lifelines2 provides several filter mechanisms. First, analysts can select an event type and a number to filter by number of occurrences of an event. Analysts can also filter by a sequence. By selecting from a list of drop down boxes, analysts can specify a temporal sequence, and remove records that do not contain such a sequence. Analysts can now specify both event presences and event absences in the sequence filter. That is, analysts can specify A before C, with no B in between. The sequence filter does not afford ways to specify temporal constraints such as B after A within 2 days (Chapter 4).

In (b) of Figure 5.1, the temporal summary is aggregating the event *CREAT-H* over 3598 records and over the visible time frame. Analysts can select up to three types of events from the controls in (d). When multiple events are selected, they stack up, and analysts can visually compare the relative proportions over time (Figure 5.2). The combo box on the top right of (a) indicates that the aggregation

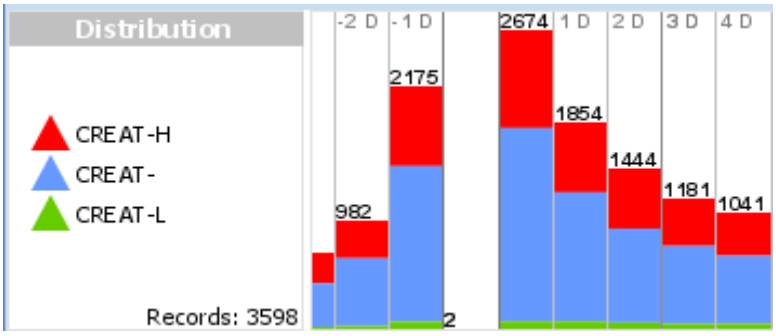

Figure 5.2: A temporal summary showing the daily distribution of creatinine test results of 3598 patients when aligned by their 1st occurrence of *Radiology Contrast*.

is in the granularity of days. Temporal summaries will automatically aggregate over an hour or over a month as analysts zoom in or out. The combo box on top of (b) indicates that the summary is aggregating events, but analysts can make it aggregate records instead (how many records have at least one event of the specified types in each day) or events per record by controlling the combo box. Analysts can directly draw one or more selection boxes onto temporal summaries to specify a temporal range selection (Figure 5.1 (b)). Records that are selected are highlighted in orange gradient with check marks next to them (Figure 5.1 (a)). By affording a way to specify multiple temporal range filters, temporal summaries enhance ARF.

5.2.2 Comparing Within A Single Group

5.2.2.1 Showing Distribution of Multiple Events

Medical analysts would bring up Lifelines2 and rank by *CREAT-H* events to bring the most "severe" patients up top and align by the *1st Radiology Contrast*.

Then they would bring up temporal summary and show the distribution of *CREAT-H* as in Figure 5.1. The medical analyst may hypothesize that the patients either had worse renal function immediately before and after their first radiology contrast, or that more tests had been performed closer to a *Radiology Contrast*. Adding all other creatinine test results CREAT- (normal), CREAT-L (low) to the temporal summary revealed that it is more likely the latter case, as more tests were performed near the alignment (Figure 5.2).

5.2.2.2 Splitting Records By Event Occurrences

The second tab at the bottom of the temporal summary in Figure 5.1 lets analysts split the records in view into four mutually exclusive groups, and display the 4 summaries simultaneously. These records are split by the occurrences of their *CREAT-H* events with respect to the alignment. The records that have the *CREAT-H* events only before the alignment are in the first group, the records that had *CREAT-H* only after are in the second group. Those that had *CREAT-H* both before and after were in the third group, and those that had *CREAT-H* in neither were in the fourth group. Events that co-occur at the same moment as the alignment event are not considered when deciding which group they fall into, unless it is the only such event in the record (in which case, the record will be classified into the last group). Figure 5.3 shows the respective temporal summaries. The third summary shows that over 500 patients have *CREAT-H* before the alignment. The medical collaborators believed that many patients in that subgroup probably

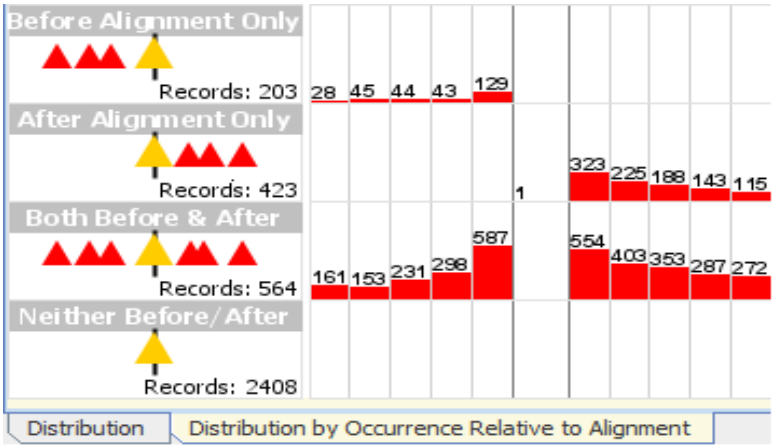

Figure 5.3: The second tab in (b) of Figure 5.1 is activated. The data in Figure 5.1 is split into 4 mutually exclusive groups: those who have *CREAT-H* only before alignment, after, both, or neither. 2408 patients never had any *CREAT-H*. 564 patients have *CREAT-H* both before and after the alignment.

experienced chronic renal insufficiency, and should be removed from the data as their lowered renal function was not related to the radiology contrasts. Upon our physicians' suggestion, I used the occurrence filter to remove those who have had at least 4 *CREAT-H* prior to the alignment (using the "Remove Selected" and "Save Records As New Group" buttons on the control panel on the right). This removed 219 records.

5.2.2.3 Direct Manipulation of Temporal Range Filters

Next I aligned the rest of the records by all occurrences of Radiology Contrast. Lifelines2 duplicated the records that have multiple *Radiology Contrasts*, and placed the duplicates and the original in accordance with the alignment in the same display. After applying the alignment, the temporal summary aggregated over all of the

duplicates and originals in the temporal summary. This means when I specified a selection box over the summary now for the first 2 weeks after the alignment, I was selecting all patients who had at least one *CREAT-H* events within 2 weeks after any *Radiology Contrasts*. Multiple-event alignment like this one allows me to select all occurrences with respect to a single event, and the temporal summary allows me to specify the temporal constraints. Once a selection is made (either through selection from temporal summaries, or through the filtering mechanisms), analysts can create a new group of records and give it a name that they can refer back to, using the controls nestled between (c) and (d) in Figure 5.1.

This phase of filtering reduced the patient count to 792, but when my collaborators looked at them, almost 90% of the patients in this group did not display the temporal characteristics of contrast-related renal insufficiency. They then further recommended to restrict to only patients who have a baseline reading of normal creatinine reading prior to contrast. We decided to apply a sequence filter that specified patients who had a *CREAT-* (normal reading of creatinine), followed by no readings of *CREAT-H*, followed by a *Radiology Contrast*, and finally followed by a reading of *CREAT-H*. Finally, this resulted in just 157 patients (Figure 5.4).

Since I had ranked all patients by the number of *CREAT-H* events in descending order, the first six patients in Figure 5.4 are likely to be the most severe. Inspecting the event patterns, these patients also showed strong evidence for contrast-related renal insufficiency in the two-week time frame after the alignment. The temporal summary showed a peak of *CREAT-H* events on the third day after the alignment, an unusual pattern. Summaries of all previously created groups (not

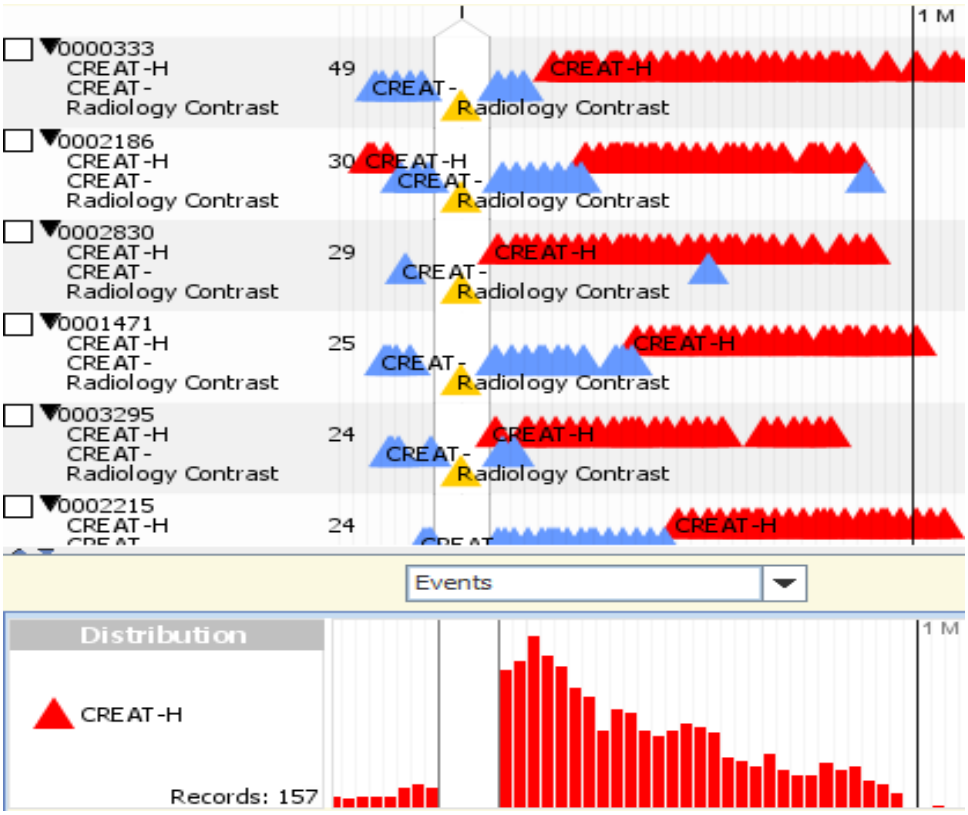

Figure 5.4: The first 6 patients in the final group of 157 that fits our filtering criteria. The unusual peak of high creatinine tests on the third day after the alignment is indicative that our exploratory search is heading to the right direction.

shown) showed a steady decline of the *CREAT-H* counts after Radiology Contrasts. This anomaly suggests that the exploration with the physicians was in the right direction. After removing patients who had 2 *CREAT-H* or fewer events as unlikely candidates, I presented our collaborators a total of 84 potential patients who experienced contrast-related renal insufficiency. Our collaborators then were able to refer to the existing literature and determine that the number is well-within published studies for a large scale hospital.

5.2.3 Comparing Groups

The previous sections show that temporal summaries are useful in showing event distributions of a single group over time. They are data overviews that help analysts make decisions on how to proceed with an exploration. They are also tightly integrated with the existing ARF framework to enhance searching. But temporal summaries are most useful when used to compare multiple groups of records at once. Analysts can select "Comparison View" (a tab on top of (a) in Figure 5.1) to compare automatically created groups (Section 5.2.2.2), or compare previously created groups. In Comparison View, analysts can specify any number of events (instead of only three) to be included in the comparison, and align the data just as before. If analysts choose to perform a within-group comparison, they must also specify a split criterion. We have already seen split by event occurrences relative to the alignment in Section 5.2.2.2. Analysts can also split by whether a record has N numbers of a certain event, automatically creating 2 mutually exclusive groups. When between-group comparison is selected, analysts can select multiple previously created groups and explore these groups using temporal summaries in a coordinated manner. It is worth noting that the automatically split groups are always mutually exclusive, while analyst-created groups are not constrained by mutual-exclusivity.

Figure 5.5 shows a partial screen shot of the Comparison View. The controls to the right indicate that this is a between-group comparison. The final group *Final 157* and its complement are selected and shown as two summaries. The creatinine high (*CREAT-H*) and normal (*CREAT-*) events are aggregated by month. All

patients are aligned by their first radiology contrast.

When comparing groups that have a large difference in the number of records (*i.e.*, 3441 vs. 157), a raw count of events is often not informative. The counts need to be normalized by the number of records in each group in order for meaningful comparison. Using the "Events (Normalized by Records)" display option, as in Figure 5.5, the event counts are normalized. The numbers on top of the bars indicate how many creatinine high and normal readings per patient exists in that month. This comparison shows that the *Final 157* patients had a much higher average than its complement. This is because the patients in the *Final 157* are higher-risk patients, and tests were performed more frequently for each patient to better monitor their health.

The contrast-creatinine scenario was developed with our physicians over a period of six months. With the physician collaborators' help, I iteratively refined my design and added features to support their needs. The successful process of narrowing the set of patients and showing comparisons pleased the physician analysts. I was able to build confidence in the value of Lifelines2, and subsequently proceeded with the following case studies.

5.3 Case Studies

I present two case studies here. The first is another application in medical records where the physicians are studying the length of stay of patients in the hospital. I also report a second case study on graduate student academic records.

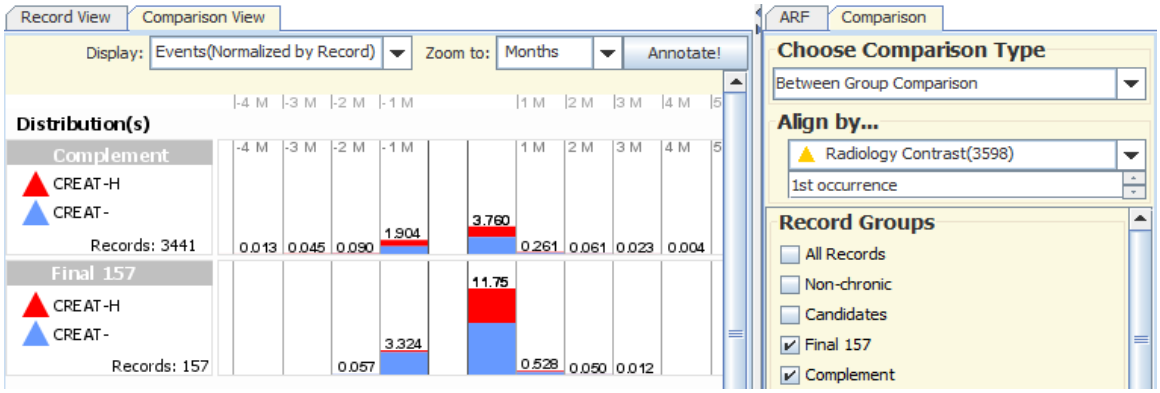

Figure 5.5: Summaries are shown in this between-group comparison. One is the result of the final filter (*Final 157*), and the other is the complement set (*Complement*). Creatinine high (*CREAT-H*) and creatinine normal (*CREAT-*) are aggregated by month. The events counts here are normalized by the number of records in that group for meaningful comparison. The numbers on top of the bars indicate the average number of *CREAT-H* and *CREAT-* events per record in that month. The patients in Final 157 clearly have much higher normalized averages

Both of these studies focus on searching for specific groups of records, and comparing among different groups to evaluate the analysts' hypotheses, although the medical scenario is more mature.

5.3.1 Heparin-Induced Thrombocytopenia

Thrombocytopenia is a medical condition in which the platelet count in blood stream is low. Heparin, a drug used as an anticoagulant, is known to cause this adverse side effect in 0.5-10% of patients. Heparin-induced thrombocytopenia (HIT) is characterized by a sharp (usually greater than 50%) drop of platelet counts within 5 to 9 days after the first administration of heparin. However, not all drops indicate HIT. Lowered platelet count can simply be the normal side effects of heparin.

To ascertain whether a patient has HIT, an additional test called HIT antibody test is ordered. Unfortunately, the HIT antibody test has high *sensitivity* but low *specificity*. When the test returns negative, then the patient is likely not to have HIT (98% accuracy). But when the result is positive, the test is only about 25% accurate. In clinical care, a hospital does not have the luxury to spend an additional 5-7 days to perform a more accurate test. Instead, patients whose HIT antibody test returns true are treated as if they have HIT. A recent medical study on 22 patients showed that a hospital treating 50 HIT patients a year can incur $70,000 to $1,000,000 and each patient can increase length of hospital stay by at least 14.5 days [101]. These patients can increase the financial cost and stretch the resources of a healthcare facility. A video of most of this analysis is accessible from `http://www.cs.umd.edu/hcil/lifelines2/`, or directly [2]

Our physician partners at Washington Hospital Center are interested in verifying these results and see if their clinical care data differ from the study. In particular, they want to focus on the patients who have been admitted to the intensive care units (ICU) with HIT. Our collaborators could query for the relevant data in their medical database and perform the analysis that way with the help of the database administrator. However, they would rather see the temporal ordering of these events and interactively narrow the data down because in order to determine whether a patient actually has HIT, the temporal ordering of events and the temporal constraints are important.

[2] `http://www.cs.umd.edu/hcil/lifelines2/videos/Lifelines2-(HIT)-2009-05-10_flash/full(1024x768)/HIT(2009-05-10)(1024x768).htm`

Over a period of one and half months, an additional case study observer (Plaisant) and I visited Washington Hospital Center three times to meet with the physician collaborators (Mukherjee, Smith) and the database administrator (Roseman). Each meeting lasted approximately two hours. Much work had been devoted to understand the medical database and to clean it up to make it suitable for this case study. Over this period of time, Roseman and I worked via email to obtain de-identified medical data, and converted them into a format Lifelines2 accepts. Microsoft Amalga [68] was the clinical information system that served as the data source for the de-identified data on which the heparin-induced thrombocytopenia case study was conducted. Its data-centric architecture enabled the database administrator to relatively easily extract the requisite data. Over all, there were over 30 emails exchanged between the developer and the collaborators to discuss the topic. When all of us met face-to-face, we spent most of the time exploring the data using Lifelines2 together. In the following exposition, "we" is used to include everyone involved in the case study.

We obtained de-identified data on all 841 patients who visited the hospital and had a HIT test for the calendar year of 2008. For each patient, we have the medical designation of platelet counts in categories (*High*, *Normal*, *Low*, *Critical*), HIT test results (*Positive*, *Negative*, *Borderline*), administration of any of the 9 heparin variants, admission and release from ICUs, and discharge code from the hospital (*Dead* or *Alive*). The categories were further pre-processed to include higher level categories. For example, platelet *Normal* and *High* events are considered the same in this investigation, so we created the new category *High/Normal* (while

also keeping the existing *High* and *Normal* categories just in case) to facilitate our exploration.

From the original dataset, we filtered to find patients who were admitted to the ICU and also had exposure to heparin. This *ICU-HEP* group has 450 patients. Then we applied another filter, to divide this new group into two subgroups: the 93 patients who had a HIT positive test (*ICU-HEP-HIT+*), and the 357 who had not (*ICU-HEP-NoHIT+*). The hypothesis is that there is a difference in the length of hospital stay between those who may have HIT, and those who almost certainly do not have HIT. We aligned the patients by their first admission to ICU, and compared these two groups against each other to see if there are noticeable differences in the distribution of discharge events (Figure 5.6 (a)).

The large difference in patient numbers (93 vs. 357) makes comparison of raw counts meaningless and also impossible for our physician partners to detect trends. We normalized the counts by selecting "Events (Normalized by Records)". In the new summaries, the counts are normalized by the number of patients in each group, and the bar heights are normalized across the two summaries for direct visual comparison (Figure 5.6 (b)). It was easy for our collaborators to recognize that the discharge distribution in *ICU-Hep-HIT+* looked more stretched out (right-skewed), indicating that the patients tended to stay in the hospital longer when they had a positive HIT antibody test result.

We further created more subgroups from *ICU-Hep-HIT+*, ones that approximated the ideal temporal orderings of HIT patients closer. Our physician partners hypothesized that by narrowing down to patients who are more likely to ac-

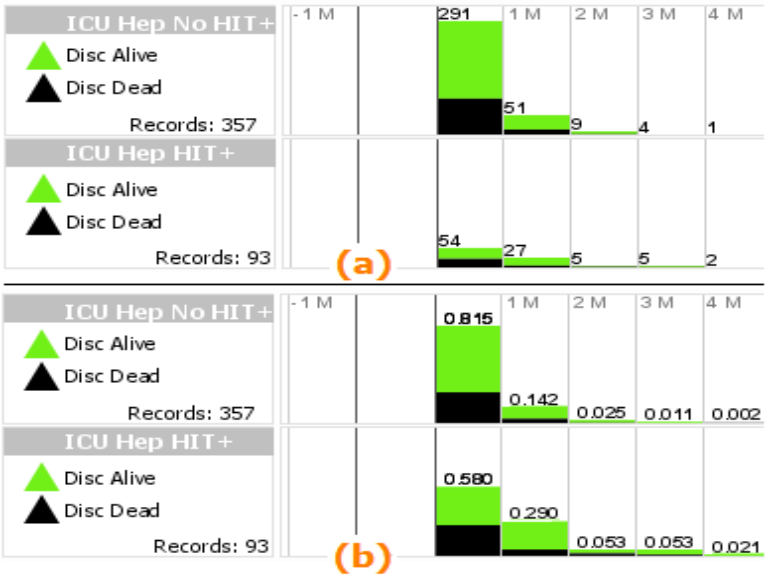

Figure 5.6: The temporal summaries show discharge patterns (Discharged Alive in green, Discharged Dead in black) aligned by the first admission to ICU. In (a), the raw count of event are shown, but the large disparity in number of patients between the two groups makes it hard to compare. In (b) the counts are normalized by the number of patients. It is clear to see that patients in ICU Hep HIT+ tended to stay longer in the hospital than those in ICU Hep No HIT+, where over 80% of patients in were discharged within 1 month.

tually have HIT, the discharge patterns in temporal summaries may stretch even more. First, we used the sequence filter to find those who had never had a *Platelet Low/Critical*, followed by a *Platelet Normal/High*, followed by any type of *Heparin*, followed by *Platelet Low/Critical*, and finally followed by a *HIT Positive* test. The filter identifies only patients who had only normal levels of platelets up until they were exposed to heparin, after which they experienced a drop in platelet, and a HIT positive result was returned. This group is named *Sequence*, and contains 63 patients. From *Sequence*, we then selected, via temporal range filter in the temporal summaries, only those who had *HIT Positive* results within 5-9 days after their first

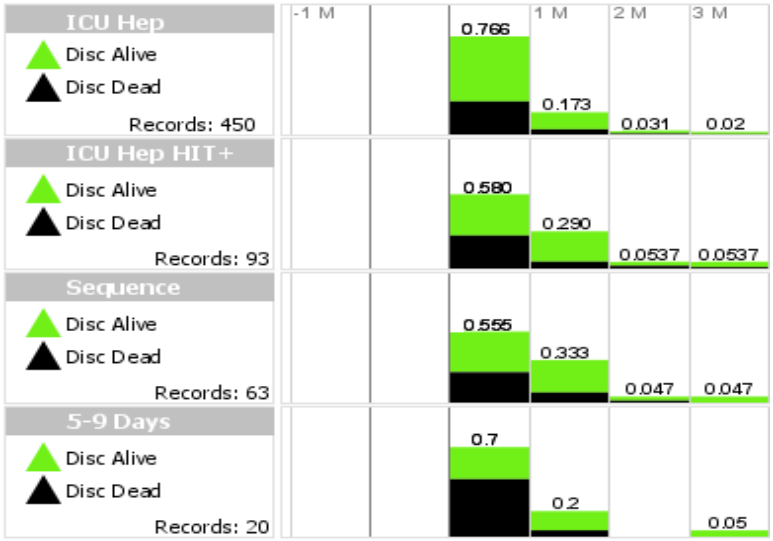

Figure 5.7: Normalized hospital discharge data aligned by first admission to ICU from 4 groups are compared here *Discharged Alive* in green, *Discharged Dead* in black). Each group is a subset of the one above it, and a "closer" approximation to true HIT patients. We hypothesized that the closer approximation, the more stretched the discharge pattern should be. The first three seem to support our hypothesis, but not the last one – we see that there are far more *Discharged Dead* than the others in the first month, and this may be skewing the data.

exposure to *Heparin*. This *5-9 Day* group has 20 patients.

The hypothesis is that as we used more stringent filters to create patient groups that better approximate the true group of HIT patients, we expected to see the discharge pattern to be more and more spread out (*i.e.*, more patients with longer stays). The comparison of these 4 groups in Figure 5.7 showed that while that seems to be the general trend in the first three groups, the last *5-9 Days* group does not follow this trend. We believe it is due to the small number of patients in that group and the higher-than-average number of *Discharged Dead* patients in the first month.

Through these exploratory analysis exercises, we have found that for patients in ICU, those who had *HIT Positive* tended to stay in the hospital longer than those who had not. Extended stay in the ICU generally translates to increased cost, but we thought it might be worth exploring with just the data we have.

We wanted to see if the patients who approximated closer to true HIT patients received more resources in terms of number of platelet tests performed. In this comparison, we included the last three groups in Figure 5.7 and also the group of patients who were admitted to ICU, had exposure to heparin, and had a negative result in the HIT test. We aligned by each patient's first admission to ICU and compared the normalized platelet data to see how many platelet tests were performed per patient in each month (Figure 5.8). We had expected to see the lower groups to have higher number of platelet tests, but there was little visual evidence to support that hypothesis. There was indeed a large difference in the number of platelet tests per patient in each month between the first and the second group, but there was little difference among the other three. The reason is that the hospital treats all HIT test positive patients (the last three groups) with the same diligence and caution even though the HIT test is only 25% accurate when it is positive.

Lifelines2 succeeded in allowing our physician partners to investigate and gather visual evidence with respect to their hypotheses. The comparisons on the discharge pattern showed that the data seem to support the hypothesis that HIT patients tended to stay in the hospital longer. However, because hospitals do not know whether a patient has HIT a priori and can only rely on the result of the low-sensitivity HIT test, we see evidence that the hospital treats all of the HIT positive

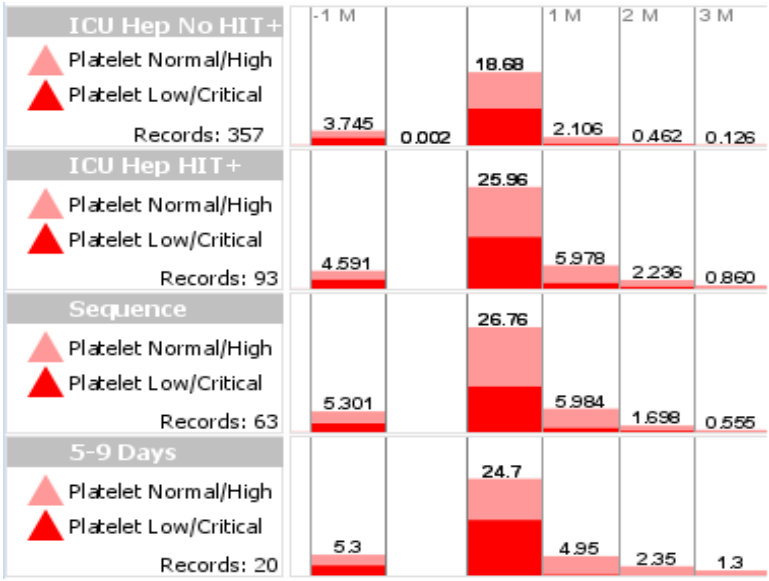

Figure 5.8: Normalized platelet data aligned by the 1st admission to ICU for 4 groups (Platelet Normal/High in pink, and Platelet Low/Critical in red). These numbers of platelet tests only dramatically increase from the first group to the second.

patients with heightened diligence and care with regards to monitoring platelets. If it were true that HIT patients do incur more cost, the cost would have to come from elsewhere. It is interesting to note that although both are comparisons of categorical data, the first comparison highlights the behavior of patient health (how soon they get well with the hospital's help), while the second highlights the behavior of the hospital care (how well the hospital treats the patients).

Dr. Mukherjee, who lead the investigation in this case study was enthusiastic about the results, but was even more enthusiastic about the process. He had been using Excel spreadsheets to organize the patient records. He had no better way to visualize the patients, and he had to perform manual review of each patient record. While manual review is still necessary, he was looking for a way to reduce the number

of necessary manual reviews. Lifelines2 allowed him to specify complex temporally ordered and temporally constrained filter criteria to remove many patients. He was able to create successively smaller groups of patients, and decide which group presents a good starting point for manual review. He commented how being able to see the individual records is important to him. He can build a narrative by the events that are visible to him, and make decisions about whether a manual review is necessary for a particular patient. He also liked how he could compare multiple groups of patients using temporal summaries and make a judgment on whether a previous filter was too aggressive. He believed that using Lifeines2 first to identify the group of patients that require manual review can save him tens of hours of work.

5.3.2 Monitoring Graduate Student Academic Progress

A second application of the temporal summaries is to monitor and evaluate graduate student progress. Progress through a PhD program can be measured loosely by grades in course work, advancing through program milestones, publishing research papers, etc. Each year, the faculty and graduate office of our department review the progress of each student, considering each of these factors, with the intent to offer advice to students and their advisors. The tool described in this paper is the first visualization tool to be applied to the process.

With various queries in SQL alone, the review can identify students who have fallen behind or are approaching a program deadline. However, temporal summaries can help to solve two classes of questions that SQL supports poorly. First, are there

factors that may predict falling behind schedule? In the case study below, I describe a faculty investigating how well being a teaching assistant (TA) for four semesters or more predicts a longer time to advance to candidacy. Second, is there evidence that the graduate review helps students be better aware of milestones and make better progress? The review has incidental benefit through, for example, making all advisors aware of graduate program deadlines, but quantitative evaluation is difficult.

There are three fundamental differences between student time lines and patient histories in the other studies. First, although the "outcome" in the medical setting is clear (months spent in the hospital, how patients were discharged), the outcome for a graduate student is much less well-defined. Further, the department considers and maintains only the information that describes currently-enrolled students; those who have left the program without a degree, who are on-leave, or even have completed the program, are not evaluated and are not (currently) in the graduate review data. This limitation in the data reduces the precision of any conclusions: for example, of the population of students who entered the program six years ago, only those who have not yet completed their dissertations are included, potentially increasing time-to-milestone statistics. Second, the confidentiality of current student records and the lack of a good data de-identifier constrain how we present results: only screen shots of aggregation of data without numbers are presented. Third, student time lines do not precisely match chronology: students may start in the spring or take a leave of absence, adjusting how time spent in the program corresponds to real time.

How might we use detailed information about a student's time line to better predict timely completion of milestones? After an introduction to the tool, the analyst (a faculty) set about to determine how spending many semesters as a TA affected the time to propose a thesis. Events represented the start of a graduate career, each semester as a TA, and advancing to candidacy. To gather evidence for the hypothesis that more TA'ing implied a longer time to graduation, the analyst constructed three groups: those who had advanced, those who had advanced after four or more semesters of TA'ing, and those who had advanced after three or fewer. To create these groups, the analyst aligned all students by the *Advanced to Candidacy* event, implicitly selecting only those students who had advanced. From this group, he used filters to choose two disjoint, approximately equally sized sub-populations: those who had TA'ed four or more semesters, and those who had TA'ed three or fewer semesters. The three groups were then visualized in Group Comparison tab of Lifelines2. The analyst aligned the groups by the event *Admission*, and showed the distribution of *Advanced to Candidacy* (Figure 5.9). When comparing these two groups and the union, it appears that students who TA'ed four or more semesters on average one additional year to advance than those who TA'ed three or fewer semesters. While the difference is startling, it is worth noting whether time spent TA'ing is a cause of delay or merely a symptom is not easily determined, but that does not diminish its predictive value.

To quickly evaluate the potential benefit of the graduate review, the analyst next constructed groups of students who were classified (by the SQL-query-based tests) as falling behind schedule to explore their success in later years. At each

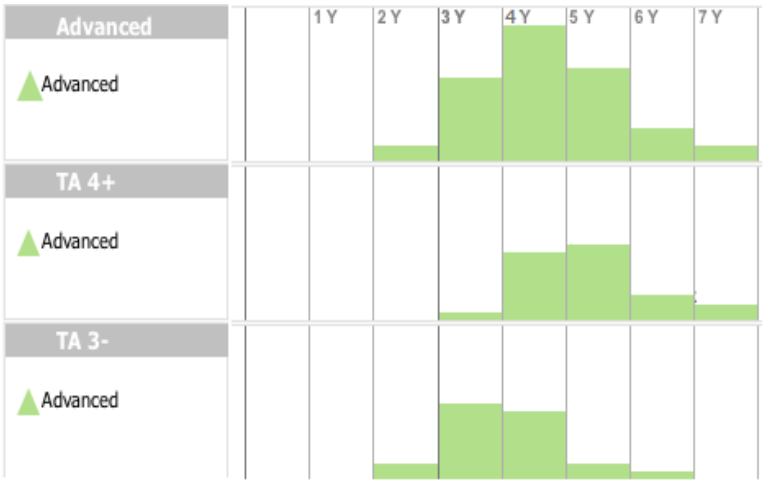

Figure 5.9: There are three groups in comparison here (from the top): Students who advanced to candidacy, students who advanced to candidacy while TA'd for 4 or more semesters, and students who advanced to candidacy while TA'd for 3 or fewer semesters. All data are aligned by the admission date. The students who had TA'ed for 4 or more semesters averaged one year longer than those who had not.

annual review, each student is given a high-level categorization that attempts to capture whether the student is *On Target*, *Concerned*, or *Very Concerned*. One way to investigate whether the annual review is beneficial to students is to examine the students who were first marked *Concerned* and see if in the following years whether these *Concerned* improve. After aligning students by the first occurrence of *Concerned*, the events that followed showed that over 70% of the students who received such a mark were no longer marked as under *Concerned* or *Very Concerned* the following year.

These analyses are preliminary; their results are not demonstrably true because of the biases in the partial dataset. However, to assist the analysts in exploring the data to quickly gather evidence for simple hypotheses, the temporal summary

provided significant help in answering key questions about the review of student progress. This initial portion of a continuing user study was conducted over a month. The faculty analyst and I met and worked together for about four hours. The faculty drove the application and spent significant amount of time using the tool on his own. The faculty and I communicated steadily via e-mails over this period.

5.4 Discussion

The two case studies were conducted differently. In the HIT study, all the exploration was done collaboratively, but our medical collaborators dictated what operations to apply, and the developer "drove" the software. In the student record case study, all actions were performed by our collaborator, and he continued using the tool independently. Our collaborator is a professor in the Department of Computer Science. He is not affiliated with my project, nor had he heard of Lifelines2 prior to our first meeting. He maintains the student record database and knows the schema. He extracted and converted all the data to an input Lifelines2 can accept all on his own. Although the HIT study was not driven by the analysts, more aspects of the application were used in more depth.

Our medical collaborators were happy to see patient records on a time line and not in a spread sheet. One said, "I am a very visual person, and the events laid out this way corresponds exactly to how I think." They did not seem to have problems understanding ARF or interpreting the data the first time they saw it. After a 10-

minute introductory demonstration of the features, our collaborators were in control, dictating what steps to take in the exploration. The dictations included both the high level logical and also what specific operations to take, reflecting their full grasp of the features.

On the other hand, having to drive the application through someone else's dictation revealed a lot of room for improvements. The exploratory steps in this case study involved a lot of group creation, between-group and between-view navigation. When creating a large number of groups, better management is needed. Additional group creation mechanisms such as taking the intersection of two chosen groups would provide for quicker results. Group names alone are not enough to help analysts keep track of the groups. Provenance information such as "what filtering mechanisms did I use to create this group" needs to be automatically saved to help analysts remember. We had designed the ARF framework so each group retains its own current state of align, rank, and filter. However, this case study revealed that analysts would want to apply the same ARF to all groups, and track how these groups differ. Allowing linked exploration among all groups would be helpful.

In the student record scenario, our analyst was driving the application, so the developer could observe how analysts unfamiliar with Lifelines2 might use it. We observed that both ARF and the temporal summaries were easily grasped. The analyst commented, "Alignment is so useful". However, the nuances on how to better strategically use ARF escaped our analyst in early use, which confirmed findings from our previous controlled experiment [21]. In a few instances, the analyst asked if it was possible to create a certain group. The process required several steps and

a combination of more than one selection mechanisms. A new user who had spent fewer than 10-15 minutes with the application was not expected to make that kind of connection. On the other hand, selecting from temporal summaries and grouping selected records were understood well in that time frame. Our collaborator used temporal summaries to perform temporal selections and create additional groups with relative ease.

5.5 Summary

I present the feature *Temporal Summaries*, and a detailed scenario describing how temporal summaries are used to give dynamic overview of the event trends and allow analysts perform temporal constraints. I show in the Heparin-induced Thrombocytopenia case study and the graduate student academic progress monitor case study how temporal summaries, coupled with grouping support, help the analyst search for the data and perform the analysis they intended. They were able to find visual evidence that supported some hypotheses, or find the lack of visual evidence that lent no support for their hypotheses. In the Heparin-induced Thrombocytopenia study, the physicians were also able to come up with a new (though related) hypothesis for further analysis. The successes in these case studies give evidence as to the usefulness of temporal summaries, and the process suggested some potential improvements.

Chapter 6

Solving Real Problems: Case Studies and a Process Model

6.1 Overview

As many interactive visualization systems have been studied in the confines of a laboratory, the experimental results are replicable and consistent. However, these settings are merely a simulation of what is out in the real world, and sometimes practitioners can find it difficult to generalize their results from an experimental setting to a real-world setting, especially when the visualization system has many features or configurations. As a result, there is a movement by visualization designers to develop less confined methods to evaluate the value of visualization systems [99, 74, 43], and several publications have utilized these methods to evaluate their systems [94, 79, 67, 93]. In particular, Shneiderman and Plaisant have advocated a Multi-dimensional, In-depth, Long-term Case Studies (MILCS) method to study how visualization systems are used in real systems [99]. The methodology consists of three main ideas. First, in the study, the system is used by analysts who have a need to use it to solve real problems with real data, outside of the laboratory setting. Secondly, the visualization designers compile data from multiple sources – such as analyst interviews, case study observations, and software logs – to build a narrative of usage in the context of real scenarios. And finally, the success of the visualization tool is measured by the success of the analysts.

While the small, yet fundamental, feature such as *alignment* can be evaluated in a laboratory setting, the evaluation of the larger set of features such as *temporal summaries*, *group comparison* would not be complete without long-term case studies. A number of case studies on Lifelines2 with real data and real participants have been conducted since the start of the Lifelines2 project, with some already presented in Chapter 5. Following the MILCS guidelines, there are two phases for the studies. In the *early adoption* phase, I assess how Lifelines2 fits the participants' needs, and new major features are developed as a result. In the *mature adoption* phase, no new major features are developed. The efforts are concentrated on bug fixings, adding minor features, and collecting participants experience with the tool.

I present these case studies and discuss them in detail. I then generalize these studies and present a process model for performing exploratory analysis of records of temporal categorical data. By analyzing the usage logs from the Lifelines2 system with collaborating testimonials from the case study participants and case study observations, I present some recommendations to the future designers of visualization systems for temporal categorical data.

6.2 Case Studies: Early Adoption

The contrast-induced nephropathy case study presented in Chapter 5 is one of the three main early adoption studies where the primary goals were to investigate user needs in searching and browsing temporal categorical records. The other early adoption case studies include an earlier study on heparin-induced thrombo-

cytopenia, the precursor study of the one presented in Chapter 5 and a study on trauma admission with hematocrit. Both of these studies are mainly directed by Dr. Mark Smith (director of the emergency department) and Dr. Greg Marchand (Associate Medical Directory at MedSTAR Transport Services and Assistant Professor of Clinical Emergency Medicine at Georgetown university School of Medicine) from Washington Hospital Center/MedSTAR.

6.2.1 Early Heparin-Induced Thrombocytopenia Study

Contrast-induced nephropathy and heparin-induced thrombocytopenia case studies share some similarities in their medical definition. Contrast-induced nephropathy is defined by a rise in serum creatinine value in the blood, and usually occurs within 14 days after a radiology contrast. This is an adverse side effect that may be fatal if not treated immediately and appropriately. Heparin-induced thrombocytopenia is defined as a sharp drop in platelet value in blood (usually more than 50%), occurring within 5-9 days after exposure to heparin, a blood-thinner. They both focused on adverse reactions due to medical procedures. The adverse reactions are not otherwise noted in the hospital database (no events marked as *contrast-induced nephropathy* or *heparin-induece thrombocytopenia*). The only way to tell if a patient has the adverse effect is to examine the event patterns in patient record. The number of patient records that require review is large. Without effective filtering and visualization strategies, the task would be too time-consuming. The sentinel events were clear on the scenarios. Furthermore, the medical definitions also require

a temporal constraint on a second event relative to the sentinel event.

A number of features in Lifelines2 were developed to support the common needs in these early case studies:

- *Absence events* in the sequence filter was implemented to allow the specification of baseline events prior to the sentinel event.
- *Alignment-by-all-events* and *temporal summaries* were created to handle temporal constraints.
- *Grouping* support was added to allow incremental filters.

However, the heparin-induced thrombocytopenia study had additional complexity in the data that required additional preprocessing support. This case involves the usage of a class of drugs collectively called "heparin." There were 12 different types of "heparin" in the hospital database. When the collaborators and I inspected the data in detail, we realized that the data needs to be cleaned. For example, one type of heparin "Heparin/NaCl 0.45%" turned out to be *heparin flushes* – heparin given to keep IV catheters open – and should not be considered as the administration of normal heparin doses. For this particular clinical question, *Plate High* and *Plate Normal* mean the same thing, while *Plate Low* and *Plate Critical* mean the same thing. These two issues mean that data-preprocessing in the form of category re-organization is required. I created a tool called "Category Mapper" that allows the mapping of original events to new events. Analysts can pre-process the data by merging categories, removing categories, and duplicating categories. For

example, we created the *Heparin Variant* category that is the superset of all other heparin events (aside from the known heparin flushes). The physicians insisted on keeping the original heparin categories so they could examine them individually if needed.

Another obstacle this data set presented was its scale. The data set originally had 8487 records with 153066 events. After preprocessing and the duplication of some events, the total number of events increased to 226585. There are almost 70000 *Heparin Variant* events total, and when align-by-all was applied to *Heparin Variant*, 70000 visual representation of the records are created, overwhelming the system. In order to deal with the problem, I reworked the underlying system to reduce the memory usage. Event names were transformed into integers. Visual representations of records are only created for the ones only visible on screen. They are created on-demand, and cached for later access. These optimizations allowed Lifelines2 to handle this large dataset and its subsequent operations.

Even with all the additions to Lifelines2 and refined data, the case study did not progress as well as anticipated. This is due to several reasons. While Dr. Smith and Dr. Marchand were interested in identifying patients that may have the heparin-induced thrombocytopenia, they were busy fulfilling their daily duties, and could not meet very frequently (once every two months was fairly average). Even though the database administrator David Roseman was able to meet with me to deal with data issues and keep an eye on the progress, the case study ultimated needed the physicians' guidance. The progress of this case study was very slow. From initially obtaining the data to refining the various aspects of Lifelines2 to handle the data

to finally allowing data exploration with the physicians, it took about four months. After sitting down with the physicians to look at the data, they realized that there may be too many other reasons why a patient's platelet may drop after having received heparin. The physicians were unsure as to what other pertinent data can be added to increase the quality of analysis. It was decided that this study should be on-hold until a more focused question can be asked and more specific data to be included.

This case study eventually led to the study of HIT patients in ICUs and the cost they incur to hospitals reported in Chapter 5. That study was conducted with Dr. Vikramjit Mukherjee, who was more available and could meet more often. In contrast, that study took one month and half total.

6.2.2 Trauma Patients and Hematocrit Study

The trauma admission and hematocrit study was mainly directed by Dr. Greg Marchand. Hematocrit is the proportion of blood volume that is occupied by red blood cells. Low or critical levels of hematocrit indicate significant hemorrhaging. Dr. Marchand was interested in what proportion of trauma patients whose hematocrit level dropped significantly (usually more than 25%) in the first 24 hours upon trauma admission. Subsequently he was interested in the relationship of hematocrit test values as a predictive measure for patient discharge status (whether discharged dead or alive). This study was conducted at the same time as the early heparin-induced thrombocytopenia study.

The Washington Hospital Center initially provided 7801 patients with 68941 events in this dataset. It consists of patients who were admitted to the trauma service with hematocrit test (HCT) results. We were able to find the proportion of patients who had low or critical levels of hematocrit within 24 hours of trauma admission using temporal summary and range selectors. The dataset later expanded to include discharge status.

This study required comparison of two groups. One with recorded low hematocrit readings within 24 hours of entering trauma care and one without. I implemented a comparison view to Lifelines2 to allow comparison of event trends between existing groups. Because sometimes analysts compare groups that have very different sizes, I added options to allow different aggergations to be visualized. Analysts can choose to visualize (1) Number of Events (default),(2) Number of Records, (3) Number of Instances (when align-by-all is in effect), and (4) Number of Events per Record (for normalization). In particular the screen shots presented below use the fourth option.

Figure 6.1 shows the number of hematocrit tests performed per patient in each day, aligned by trauma admission. The orange portion of the graph shows the normal or high readings of hematocrit (*HCT Normal/High*), while the pink represents the low or critical readings (*HCT Low/Critical*). The top temporal summary is the population of patients discharged alive (over 7000 patients), and the bottom one represents the population of patients discharged dead (453 patients). The numbers indicate that the patients in the dead population received more hematocrit tests in the first 24 hours, presumably because their more severe conditions required

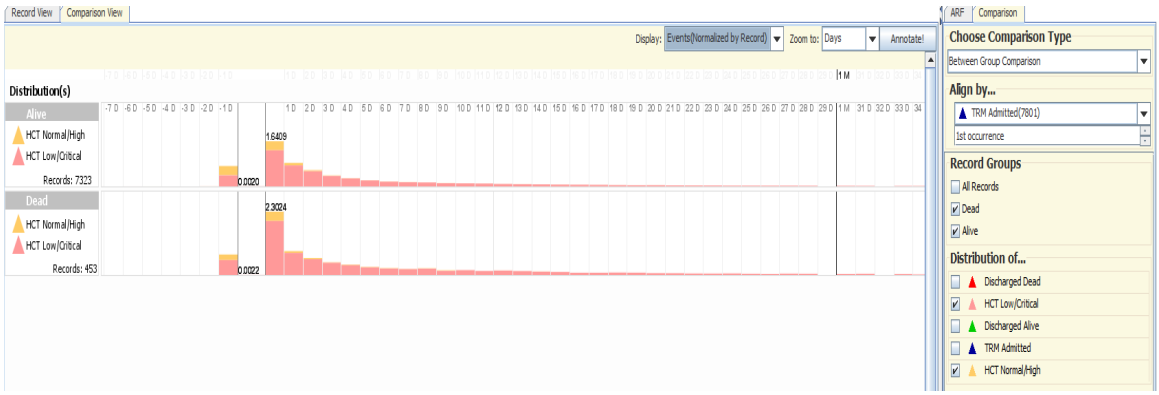

Figure 6.1: the comparison of the distribution of hematocrit lab results of two patient groups: the ones that were discharged alive (top), and those discharged dead (bottom). The data is aligned by Trauma admission. Orange indicates normal/high hematocrit events, and pink indicates low/critical events.

more monitoring. It also shows that the dead population tended to have much higher number of hematocrit low/critical values. Without this comparison, these differences would otherwise be difficult to tell from the raw event representation in Lifelines2.

Figure 6.2 shows the comparisons of discharge patterns of the two same groups (normalized by record numbers). In contrast to Figure 6.1, the were was not aligned by a particular event. The discharged alive distribution (green, in the upper temporal summary) has fewer variations than the discharged dead distribution (red) because the sample size for discharged dead is much smaller and that graph is more sensitive to small changes. The interesting thing to note is that the discharged alive distribution shows the annual peak in late summer-fall (August-Octorber), and the pattern is almost cyclic. On the other hand, there does not seem to be such regularity in the discharged dead population.

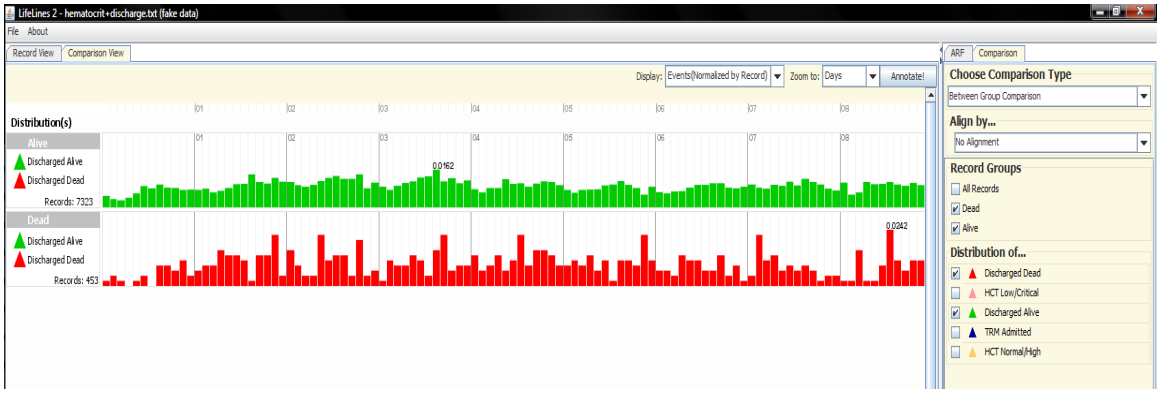

Figure 6.2: The same two groups as in Figure 6.1, but this figure shows the distribution of the discharge numbers (normalized by record counts) across calendar year. Green indicates discharged alive, and red indicates discharged dead. There is a fairly regular pattern (almost cyclic) for discharged alive year-in year-out with the annual peak usually centers around late summer/early fall. The discharged dead aggregation does not have seem to have as regular pattern. This is partly because there are so much fewer discharged dead cases (~ 450) when compared to alive cases (> 7000).

Despite the warm reception and small successes, this case study was never fully completed. One problem that was discovered in these early studies is that sometimes the numbers in the medical description play a huge importance – as the 50% drop in the heparin-induced thrombocytopenia case and the 25% drop in the hematocrit case. Since Lifelines2 focused only on categorical data, and had no search support for numerical data, it was uncomfortable for the physicians to make a judgement on the quality of search results based on the data available. They saw the values in being able to perform exploratory analysis, trying different paths of exploration, and quickly switch from one focal point to another, but the lack of support for numerical values was a constant criticism of our approach. While numerical support remains a feature the physician collaborators wanted, I have chosen to stay away from it for

the sake of the dissertation: There had been much visualization work on that area and had less research value.

6.2.3 Observations and Discussions

The physicians were observed to be enthusiastic about Lifelines2. They saw Lifelines2 as a wonderful visual tool, and found Lifelines2 enables them to understand their data more effectively. Dr. Marchand, in particular, said that,"I am a very visual person. To be able to see the patient records this way allows me to understand it so much quicker and more reliably." When all of us met to study the data, he and his team dictated the steps that needed to be done, and I drove the software. The nature of the dictation was, "Now align by *Trauma Admission*, and show me the distribution of *Platelet Low/Critical*... and rank by the same category."' That is, the dictations were specific to Lifelines2 operations. This is indicative of their understanding of the features in Lifelines2 and how their medical questions can be translated to the combination of these features. They were able to grasp this level of understanding after just a 10 minute demo of Lifelines2. They seemed to pick up the general ideas of ARF and temporal summaries in that amount of time.

The physician and his team of physicians were observed, when in meetings with me, to constantly refer to the main visualization of Lifelines2 and discuss the potential circumstances of each particular patient. That is, Lifelines2 visualization of each patient record encouraged narrative about the patients. The physicians were able to "fill in the gap" for many of the patients. In the qualitative study presented

in Section 3.4.2 (55), I also observed the medical expert to narrate and enrich the patient stories behind the visualization based on their domain knowledge. The narration then is used to guide their reasoning and exploration through the data. For example, in the early heparin-induced thrombocytopenia case, the physicians knew there were a large number of drugs containing the name "heparin", and they also knew that "heparin flushes" behave differently than the normal heparin doses. However, it was not until they looked at the data in Lifelines2 and examined the patterns of platelet counts with respect to each heparin drug did they realize that responses were very different depending on which heparin drug was administered. At that point, they recalled that "heparin flushes" were also recorded in the database, and needed to removed in order to proceed in the study. Thus, the physicians were able to make connections between the individual record visualization and their domain knowledge, and classify patients by the group visualization (temporal summaries) support. When interviewd by the Terp Magazine, Dr. Marchand commented on Lifelines2, "This technology saves time and gives us another important diagnostic tool" and "[it] will not only make for better care by doctors, but also help patients make healthier choices on their own." [111].

The physicians felt that these case studies, while initially thought appropriate, were not the best match for Lifelines2 because of the lack of numerical value support. They had decided to wait and think of additional, more appropriate scenarios for Lifelines2. Dr. Marchand had since then moved onto other things and was not able to guide the direction of the investigations.

6.3 Case Studies: Mature Adoption

When most of the features in Lifelines2 were implemented, I started conducting more mature case studies. During these case studies, no main features were added. The efforts were put into software stability, bug fixing, and minor feature additions. The goals of these studies are to evaluate the usefulness of Lifelines2 as a whole, gather participants' comments and suggestions. Chapter 5 presented the more mature heparin-induced thrombocytopenia study and the graduate student progress case study. Here I describe other studies that were conducted in the more mature adoption of Lifelines2.

6.3.1 Heart Attack in Relationship to Day Light Savings Time

A recent article published in the New England Journal of Medicine tied the day light saving time changes to an increase of heart attack incidents in the following week [44]. In particular, the study indicated that the spring time change commonly known as "spring-forward" seems to be associated with a higher number of heart attacks. However, "fall-back" does not seem to be associated with the phenomenon (in fact, the rate is lower). Dr. Mark Smith of the Washington Hospital Center was curious to see if by using Lifelines2, we could mirror the study and find out whether there is such a phenomenon in the clinical database in Washington Hospital Center.

He requested the database administrator David Roseman to find patient records of the following three indicators of heart attack from the years 2004-2008: (1) elevated troponin level (2) admission diagnosis of myocardial infarction (or related

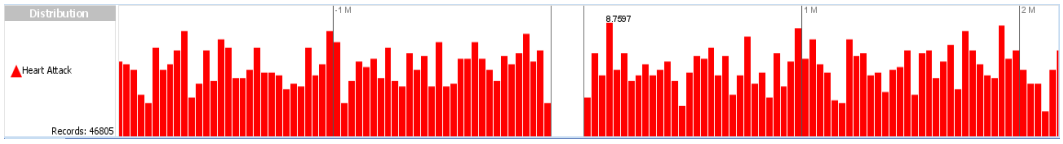

Figure 6.3: Distribution of heart attack events aligned by all "Fall-Back" daylight saving time change days.

forms), and (3) discharge diagnosis of the same terms. Roseman was able to find patients of (2) reliably from the database but not the other two, so I used those data to perform the mirror study.

I first converted the data into Lifelines2 format, and supplemented it with the day light saving time change events. There are three additional event types: *Spring Forward*, *Fall Back*, and the generic *Day Light Saving Time Change*. There are a total of 9361 records and 196581 events total.

Initially I performed align-by-all on *Day Light Saving Time Change* event, but no pattern in the temporal summaries can be visually detected. I then aligned-by-all *Fall-Back* and also aligned-by-all *Spring-Forward* events and showed the distribution of the heart attack occurrences. While there does seem to be a rise of heart attack incidents in first few days following the time change, the rise is not qualitatively different from other rises in the graph. The visual evidence is not overwhelming.

Despite the lack of visually compelling data, I investigated it further by following the the steps of the study. The study compared the daily incidents rates in the week following the time change to the average of the incidents rates of the same day 2 weeks before and 2 weeks after. Using Temporal summaries and focusing the

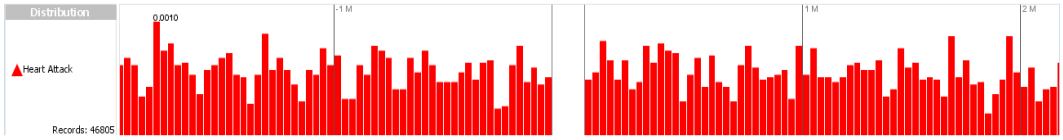

Figure 6.4: Distribution of heart attack events aligned by all "Spring-Forward" daylight saving time change days.

visualization to the granularity of days, I was able to quickly obtain the numbers. Putting these numbers into an excel spread sheet to further compute the comparisons. Similar to the original study, the incident rate of heart attacks are higher in the week following the Spring-Forward time change than the week following the Fall-Back time change by about 22%. Also similar to the original study, there is a large spike of heart attack incidents on the third day after the Spring-Forward alignment. While the original paper reported that the incident rates were lower in the week following Fall-Back, I did not observe such a trend in the Washington Hospital Center data.

The relatively small sample size and the nature of clinical data as opposed to high-quality, well-maintained data may be the cause of the difference. The sample size from the original data is about 2 orders of magnitude larger than the WHC data. The original study used data from the Swedish registry of acute myocardial infarction, which is known for its high quality. Nevertheless, the study with WHC was completed in 2 hours – from converting the data from Roseman to finishing analyzing the data in Excel. The speed comes from the flexibility of being able to specify the temporal granularities to days, and allowing the align-by-all to focus

on all *Spring-Forward*s or *Fall-Back*s. Combined with how temporal summaries aggregates the events, the case study was fairly straight forward and trivial. The uniqueness of this study was that there was no filtering or ranking operations. All the physicians cared about were the incidents rates, trends, and the final comparative numbers.

6.3.2 Communication Network in Drug Trafficking Ring

Chris Hutchins, the Information Technology Manager for the North Texas High Intensity Drug Trafficking Area (HIDTA) under the Office of National Drug Control Policy contacted me for a copy of Lifelines2 for his data analysis of those under surveillance. The bulk of his drug trafficking data consists of telephone records which are time-stamped. He had been using the Organizational Risk Analysis software from Carnegie Mellon, a MySQL Database, and MS Office software for data analysis. He also used Tomcat for active web component, php for scripting. Java and javascript are his choice of programming languages.

Ideally, Chris wanted analysis tools that can provide spatial-temporal-social analysis support, but it was not possible to find the appropriate tool. He turned to academic tools to supplement his needs. He also requested GeoDDupe [48] for geospatial analysis, SocialAction [78] and later NodeXL [100] for social data analysis, and finally, Lifelines2 for temporal sequence analysis from Maryland. He was spending some time aside from work to identify potentially useful tools for their daily analysis work.

One aspect of his analysis was to study the communication patterns of the involved parties. He believes the communication patterns will trace the necessary steps to move drugs and money around the country and that visualizing these calls on a time scale can help him find key participants. In addition, he thinks Lifelines2 can help him identify the sources and destinations of communications across the entire network and, hopefully, whether local netowrks "fire" from either supply-side forces or markeet-side forces.

Part of the study was to determine what was the best way to transform the network-natured data into temporal sequence records. Through a few e-mail exchanges, Chris and I determined that it was best to convert each phone call source-destination pairs as each event type, and then salt calls (by adding special events for them) where there is significance. For example, calls to the person with the highest eigenvector centrality can be salted so that Chris may align by it.

Having decided on this approach, Chris fetched the data and processed it. In a small set of data where there are 51 records and 628 phone calls, Chris looked for sentinel events that increased chatter. He found it by focusing on the calls made by the person with the highest eigenvector centrality and the number of calls between this person and another important player *From Ryan to SEAN H.* By quickly testing different sentinel events, he was able to find the call *From RYAN to ROMEY* as the sentinel event that seems to be the precursor to a disproportionately large number of *From Ryan to SEAN H* in the following month 6.5.

Chris later added the event "Interjection" to the dataset to study the communication trend before and after the law enforcement intervention. After detailed

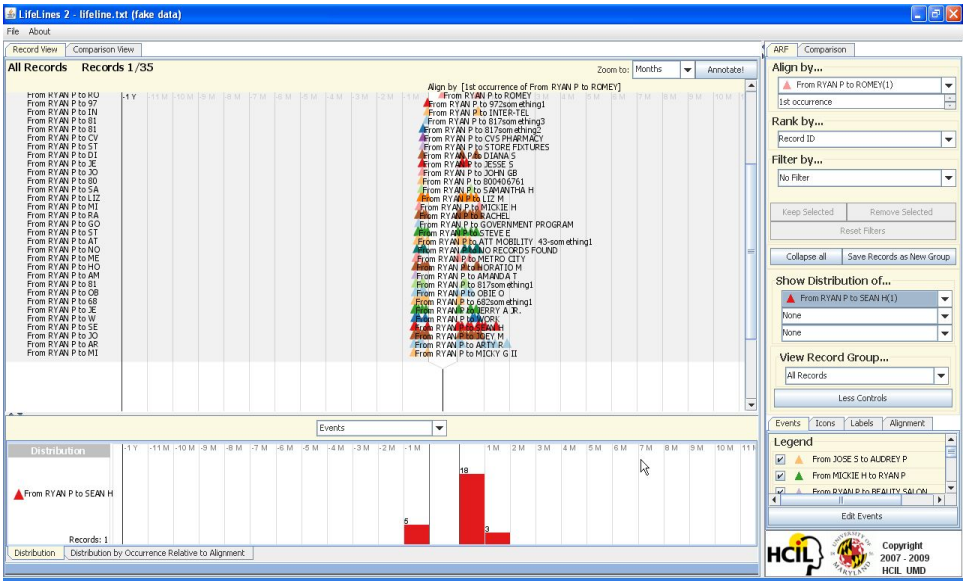

Figure 6.5: Telephone data in a drug trafficking ring is aligned by a phone call between the person with the highest eigenvector centrality and another. The analyst tried several sentinel events, and found this one led to an increased chatter between Ryan and Sean, two important players in the network.

analysis, the "Interjection" event is correlated to the increased number of phone calls by important actors in the network. He commented that "the distribution shows the impact of event Interdiction on communications between 2 selected parties." While performing these studies, he enthusiastically commented on the combination of alignment and temporal summaries, "the distribution panel shows an extremely good view of what happens to a selected call target in relation to the aligned call.'This is not done well by any other software I'm aware of!"

This case study was conducted without any face-to-face meetings or phone calls. The progress was reported by Chris via e-mail. Since the data were sensitive, I only received the anonymized data and screen shots from him. Chris also reported

a number of bugs that I fixed.

In a post-study e-mail interview, he responded that he had no more time to perform investigation with Lifelines2, "I used [Lifelines2] for several weeks, and stopped due to lack of time", "I haven't been using Lifelines2 recently and my investigations of it stalled, for the classic reason: I got no time to advance technology because there's no time available. Not my choice. There's a chance I'll get some graduate research assistants help from UT Dallas this year, but the project is in its infancy." However, he felt that he was getting some new perspective with the visualization provided by the tool: "[With regard to Lifelines2], I think I was approaching a point where it would be helpful. Since I was very interested in sequential event analysis, I found some of what I was looking for." He commented on the quality of the information shown in the visualization in Lifelines2 to him and the viability of sharing the data with his colleagues as follows: "Since I [...] ran short of time to fully analyze it, I didn't try to share the results with very many people. The few I did show thought highly of the information available from the software." When asked about the biggest bottleneck for his case study, he said it was data extraction and conversion. He said that "Law enforcement (LE) is riddled with siloed data sources, and lack standardization. [...] There is no standardized training for analysts across the country."

6.3.3 Exploring Book Reading Sessions on ICDL

International Children's Digital Library (ICDL) is a project that began and is maintained in the Human-Computer Interaction Lab (HCIL) at the University of Maryland. It hosts over 4000 children's books translated into over 50 different languages [1]. Lab visitor Rong Chen was interested in analyzing how the books are accessed, how this information can be used to help the site understand how people are reading the books, and ultimately how to design features to support the reading habits.

Rong used the server web logs to track each visitor's session in ICDL. The logs identify each visitor by IP, and catalogs the visitor's reading pattern – when the visitor clicked on a certain page, how is the page represented (single page format or double page format), what language is the visitor using to view it. She had used Spotfire [2] as an analysis tool to explore the various dimensions of the log data. However, the book page visitation patterns were of particular interest to her, and Spotfire did not have good visual representation for temporal categorical records. She turned to Lifelines2 for her visualization needs.

Initially, she chose to visualize one of the most popular books "The Blue Sky" from the library to get an understanding of what the data feel like. Each visitor identified by an IP is transformed into a short ID number. Each visitation to a book page was recorded as a separate event, *e.g.*, event *001* for visiting page 1. Since every visitation is associated with time, these events are time-stamped. While

[1] http://www.childrenslibrary.org/

[2] http://spotfire.tibco.com/

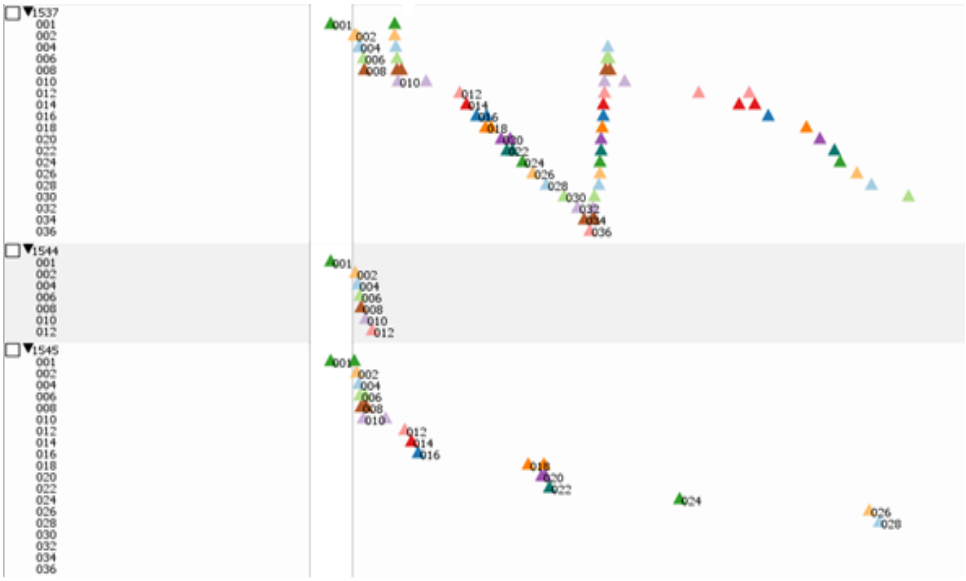

Figure 6.6: Three records representing three visitors' reading behavior in ICDL are shown. These records are aligned by the visitor's first visit to page 1: event *001*. These behaviors do not conform to the common expectation. In particular, the first visitor has bursts of fast backward seeking as if the visitor clicked on the "back" button in the browser quickly to get to a certain page.

she had expected page-by-page transitions as in the canonical reading behavior, not all visitors exhibited that pattern. Some visitors did not start with the first page. Some visitors did not finish the book. Some visitors jumped around pages, while still others seemed to be turning many pages very quickly without reading. Some of the patterns can be found in Figure 6.6.

One particularly interesting class of patterns that was discovered was that some visitors would have a series of many visitations to consecutive pages previously viewed in the reverse order. We had conjectured that these visitors used the "back" button on their browser to quickly get back to a page they previously visited. This raised the question that why the visitors chose to use the back button instead of the

book page index page where all pages of the book are in one view, and the visitor can choose one with the guidance of thumbnail pictures. Was it because the visitors did not know how to access the index easily? Or do the visitors perceive it was easier/faster to use the "back" button? While we were not able to pursue these questions further (we could not link IPs to contact information of the users since ICDL does not require registration to use), this may have revealed a ninteresting usability trend by the visitors to ICDL, and may influence later site design.

One other interesting insight that was gained was that in books where there is no text in the initial pages (as many children's books often have rich illustration instead of text in some pages), the most frequently visited first page for those books is the first page with text. Some visitors are observed to read the books more than once in one session – perhaps twice very thoroughly, or once casually and once thoroughly, or once thoroughly and carefully re-read several other pages.

While these results were interesting, the most important thing was that the visualization revealed several previously unanticipated reading behaviors by visitors. Because of this, Rong and her colleagues Bederson and Rose realized that analyzing the ICDL usage required some structural definition – what does it mean to read a book? From the web log data, how can we define that a visitor has read a book (or most of the book)? They later used a number of variables to define what a Real Book Reading Session is, and used the definition to mine visitor logs that fit the definition. The results of their findings were published in the 13th European Conference on Digital Libraries [13].

While Lifelines2 played a role in helping the analysts visualize on-line book

reading behaviors, some shortcomings of Lifelines2 were also revealed. It was not possible to compute some general but useful statistics. For example, in the book reading example, it would be nice to compute the average time spent between pages, or average time spent between certain pages. Secondly, it was difficult to generalize event sequences. Spotting reading patterns that are very distinct from others was not very difficult, but spotting the difference among similar event sequences was. A better aggregation and visualization support for event sequences would go a long way for this case study. However, there can be challenges in designing an effective sequence aggregation for this task – the analysts are interested in *reading* patterns, which is different from *event* patterns . That is, the analysts are interested in reading patterns that are represented in the web log as a class of event patterns. They care about how book is read, which is a higher-level generalization than the sequential access to pages of the book. Unless the sequence aggregation is able to flexibly further generalize specific sequences, the problem will remain unresolved.

6.3.4 Hospital Room Transfer

Physicians in the Washington Hospital Center are interested in patterns of patient transfers between hospital rooms. The rooms in a hospital can be classified into several categories: *ICU* (intensive care unit, typically the highest level of care), *Floor* (normal hospital beds, typically house the patients with no life-threatening conditions), *IMC* (intermediate medical care, which typically houses the patients that need elevated level of care, but not serious enough to be in ICU), and *Special*

(emergency rooms, operation rooms, or other rooms). In these studies, these events represent the entering time of the rooms, not the exiting time. The physicians are interested in two particular pattern sequences:

1. **Bounce-Back**: In the bounce-back scenario, physicians are looking for patients who were discharged from a higher-level care room to a lower-level care room, and then back to the higher-level care room. For example, ICU -> Floor -> ICU will be one such sequence (where -> denotes "followed by"). These cases represent patients who may have been released from the higher-level-care rooms too soon.

2. **Step-Up**: The step-up cases are the patients who were initially triaged into IMC rooms, but then immediately required escalation to higher-level care rooms. These patients are the ones who may have been mis-triaged. That is, they should have been sent to ICU as opposed to IMC in the first place.

These scenarios represent one way the hospital can evaluate its quality of care. Performed longitudinally, the hospital can assess whether there is a drop or an increase in quality of care. The numbers can be a measure of quality assurance. The physician working with me is Dr. Phuong Ho. Aside from being a full-time physician, Dr. Ho also works under Dr. Mark Smith in MedStar Institute for Innovation (MI2), which seeks to find new technologies that can potentially help physicians provide better care and hospital administrators to assess the quality of care in the hospital system.

Dr. Ho said that, currently, the way Washington Hospital Center traditionally finds these indicator numbers is either incomplete or time-consuming. First, standard report of patients admitted to IMC or ICU is fetched quarterly. This is an automatic service in the database system, and the query was written to include patients admitted in that time frame and stayed in IMC or ICU. However, patients who were admitted prior to the time frame and transferred to IMC or ICU would be missed. This makes the quarterly report routinely miss a handful of patients every time. This automatic system was, however, not amenable to change, so the physicians could not change the query it issues. The way the physicians cope with this is to write their own SQL query to fetch all patients in a certain time period, overshooting the quarter by months to guarantee as many patients are included.

After they have put the data (either the under-reported automatic service or the over-reported SQL query) in Excel, they would write rudimentary scripts to perform this kind of temporal pattern search. However, they have found it difficult to correctly specify the pattern in Excel. It was particularly difficult for them to specify the temporal constraints and how far along the data should the query look for (ideally it should be for all data, but the physicians felt they were at the limit of their expertise with Excel and could not get to a satisfactory state.) The Excel script typically take half a work day to implement, but would only take a few minutes to return the numbers. However, the numbers typically represent grossly lower-bounds – the error produced from the Excel scripts can be larger than that of the automatic reporting service that fetches the data – and the physicians do not find the numbers reliable.

The other way of performing the analysis is to have a resident or a nurse look at the patient records (whether digital or on-paper) manually and flag those that fit these patterns. While this approach usually gives higher quality results than the Excel approach, it is extremely time-consuming. Dr. Ho said that it typically would take at least 5 to 10 minutes to browse a record. When there are as many as 500 patients in a quarter, this is simply not a good use of their time (not to mention it could also be very error-prone). Therefore, they only do this sparingly. Altogether, these figures that supposedly measure the quality of care are not high-quality because the process of obtaining them is riddled with problems. When Dr. Ho first saw Lifelines2, he thought he could leverage on a number of features that help him get to more reliable numbers. For example, Lifelines2 can help select the data from the results of SQL (*e.g.*, selecting all patients who were admitted to ICU or IMC in a quarter regardless when they were admitted), which would reduce the data from the SQL query, but also be more reliable than the automatic data service. He commented that, "[These are] not the sexiest scientific questions, but very important clinical question for us."

6.3.4.1 Bounce-Back

The bounce-back patients represent those who may have been moved from a higher-level care room prematurely. The canonical list of sequences Dr. Ho was looking for are the following:

1. ICU -> Floor -> ICU

2. ICU -> Floor -> IMC

3. IMC -> Floor -> IMC

4. IMC -> Floor -> ICU

5. ICU -> IMC -> ICU

In addition, Dr. Ho requires that the escalation to the higher-level care rooms (the third events) needs to occur within 48 hours of the lower-level care room (second events). There are no temporal constraints between the first and the second event. These queries are slightly different from the the ones in the contrast-induced nephropathy and the heparin-induced thrombocytopenia cases. It is not as clear what the alignment event should be that would be the fastest way to get to the final answers. At the end, we decided to align by the middle events so that the temporal constraints can then be directly applied in the temporal summaries.

For example, the process to find the patients that fit the first pattern "ICU -> Floor -> ICU", we did the following steps.

1. Perform a sequence filter on *ICU* -> *Floor* -> *ICU*, and save the results as a new group named `ICU-Floor-ICU`.

2. Align by 1^{st} *Floor*, the middle event.

3. Temporally select *ICU* events within 48 hours after the alignment.

4. Keep the selected records, and save as a new group as `ICU-Floor-ICU Align1`.

5. Export this new group as a file, and return to the group `ICU-Floor-ICU`.

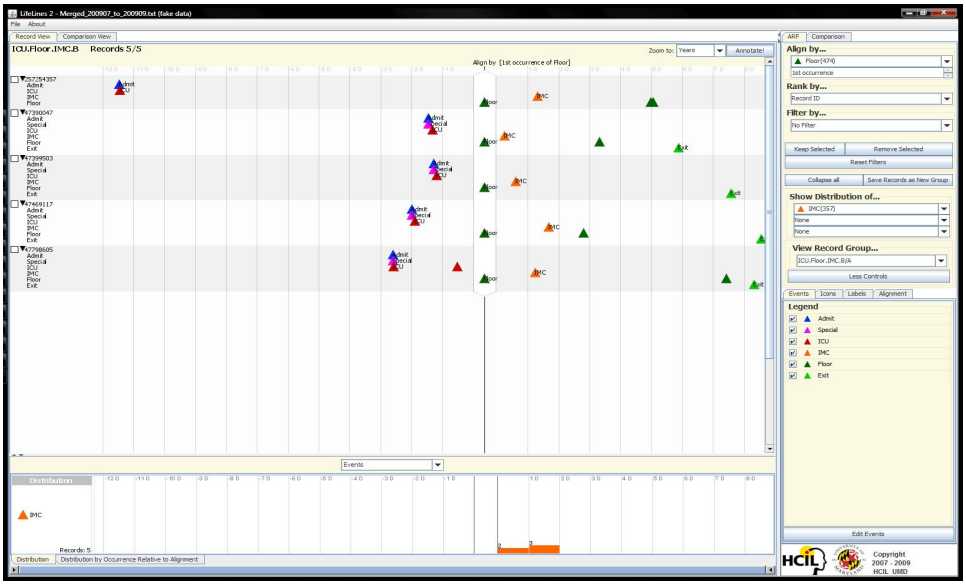

Figure 6.7: Screen shot representing the search results of one of the five patterns for the Bounce-Back Study (ICU → Floor → IMC).

6. Repeat Steps 2-5 but align by the 1^{st} *Floor* event.

These steps were repeated for the other 4 patterns. Figure 6.7 shows the search results for the pattern ICU -> Floor -> IMC. By the end of the process, there are a number of text files that represent the patient records that fit one of these patterns. We then used a text editor to merge all these files into one for a final count. Although these steps looks fairly involved, they allowed Dr. Ho to visually examine the results after each step and decide if some patients should additionally be included, like those who may have missed the 48 hours constraint by one minute or two.

For the quarter of July to September 2009, we found 19 patients out of 576 that exhibited this pattern. The pattern "ICU -> Floor -> ICU" is the most prevalent

pattern, with 10 patients. "ICU -> Floor -> IMC" had 5, "IMC -> Floor -> ICU" with 4, "IMC -> Floor -> ICU" with 1, and "ICU -> IMC -> ICU" with 0. One patient had two of these patterns, and that is why these numbers add up to 20 instead of 19. Out of the 576 patients admitted either in IMC or ICU in that time frame, this represents 3.3%, and this is well within the bounds expected by the WHC.

6.3.4.2 Step-Up

In the Step-Up Case, the Washington Hospital Center wanted to study the trends of patients who were initially sent to IMC, but escalated to ICU within 24 hours. This is indicative of insufficient initial triaging – sending patients too sick to IMC instead of ICU. Unlike the Bounce-Back cases, Dr. Ho was interested in looking at the numbers with respect to two hypotheses he has had. First, the nurses had been complaining about the increase of Step-Up cases. They had given Dr. Ho anecdotal evidence that the initial triage had not been adequate recently. Dr. Ho wanted to look at the numbers compared to historical numbers to see if this hypothesis holds was supported. It that is the case, then WHC would consider changing the way triage is performed.

Secondly, because an influx of new physicians enter the hospital in the third quarter (fall semester) of every year, Dr. Ho thinks it may be the case that the rate of Step-Up cases may be higher for these months due to the less-experienced residents. A competing hypothesis may also be the case: the less experienced residents can be

over-compensating for their lack of experience by being overly cautions and sending more patients to ICU than is necessary, thus reducing the number of Step-Up cases. Dr. Ho wanted to find out whether there is evidence that supports either hypothesis.

The original query seemed simple: align by all patients' *IMC* event, and select all *ICU* events that occur within 24 hours after the alignment. However, as Dr. Ho and I looked at the data together and tried to perform the searches, he realized some issues. For example, there should not be *Floor* events between *IMC* and *ICU* (patient going from *IMC* to *Floor* then to *ICU*) because this suggests the escalation from *Floor* to *ICU* is not due to an earlier triage. Similarly, there should not be an *ICU* prior to the *IMC* in question. If there were, the patient was already in *ICU*, and this would be considered a Bounce-Back instead of Step-Up. These nuances can be handled and easily verified in Lifelines2 in the following procedure:

1. Perform a sequence filter on *IMC* `->` *No Floor* `->` *ICU*, and save the results as a new group named `IMC-No Floor-ICU`.
2. Align by 1^{st} *IMC*.
3. Temporally select *ICU* events that occur any time prior to the alignment, and remove the selected records.
4. Temporally select *ICU* events that occur within 24 hours after alignment, and keep the selected records.
5. Save as a new group and export this new group as a file.
6. Return to group `IMC-No Floor-ICU`.

7. Repeat steps 1-6 by changing the 1^{st} *IMC* to the n^{th} *IMC*. Stop when there are no records with n IMCs.

We conducted this study for every quarter from January, 2007 to December, 2009. A sample screen shot of the process is shown in Figure 6.8. Figure 6.9 shows the number of patients admitted to IMC during that period. It is clear that the number had been on a rising trend. Figure 6.10 shows the number of patients who exhibited the Step-Up pattern in the same period. This is a subset of all patients admitted to IMC. The line graph is more jagged than Figure 6.9, however, its upward trending is also clear. Given that the number of IMC patients are not constant throughout the year, we plotted the percentage of patients who exhibited the Step-Up phenomenon out of the IMC patients (Figure 6.11). It showed that the percentage of Step-Up patients peaked in the middle two quarters of 2008 (at nearly 7%), and had actually been in decline since then. This suggested that the anecdotal evidence Dr. Ho saw was, in fact, merely anecdotal, and did not represent a larger trend. However, these numbers also explained why the nurses felt otherwise – the number of total IMC patients had been steadily on the rise, and so the Step-Up cases were also growing, reaching as many as 20 in the last quarter of 2009. Dr. Ho explained, "The nurses must have gotten the impression that mis-triaging occurred more often because they have encountered more Step-Up cases. They felt the increased number of cases was due to errors in the triaging, while the real reason is more likely the increased of IMC patients." He also said, "The reason why there was an increase of IMC patients was not due to the increase of diseases or injuries.

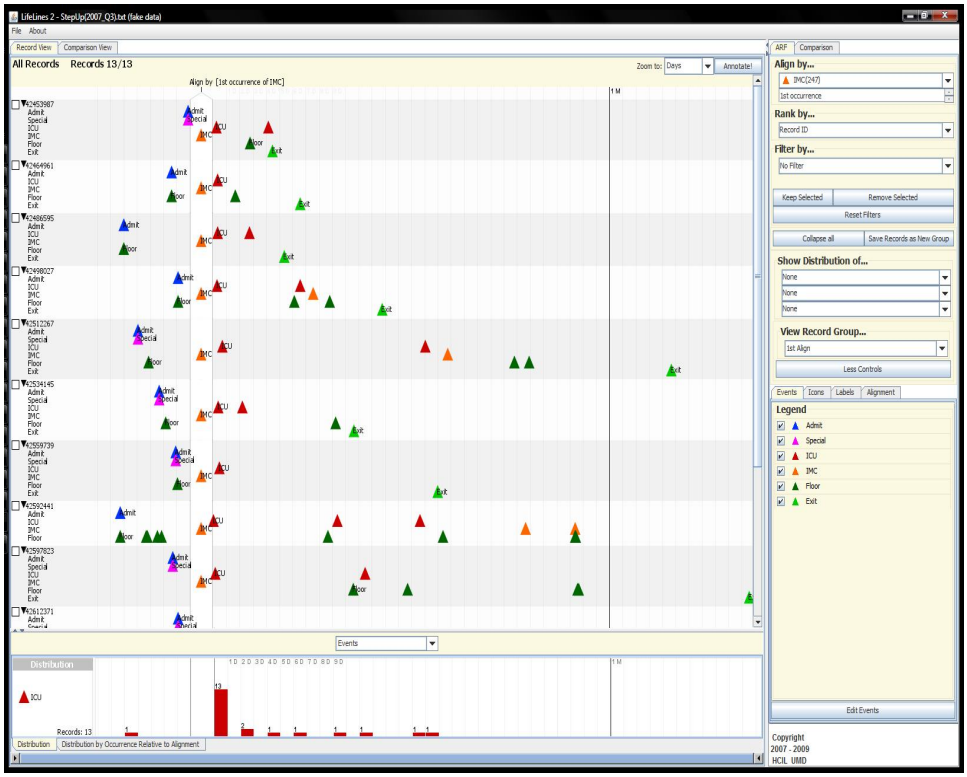

Figure 6.8: Sample screen shot of the process in Lifelines2 to identify the Step-Up patients

Instead, it was merely a reflection on the expansion of IMC care in the hospital."

I re-ordered the percentage of Step-Up patients in a Figure 6.12 to better show the differences across quarters in the past three years to investigate the second hypothesis: "In quarter 3 (Jul-Sep), because of the increase of fresh, less-experienced doctors, the number of Step-Up cases would be higher." The chart indicated that it was not quite the case. Of the four quarters, the second quarter has the highest number of Step-Up cases altogether, and the first quarter has the fewest by a significant margin. Dr. Ho commented that, "The attending physicians (supervisors of the residents) must have been doing a good job reviewing the results of the resident

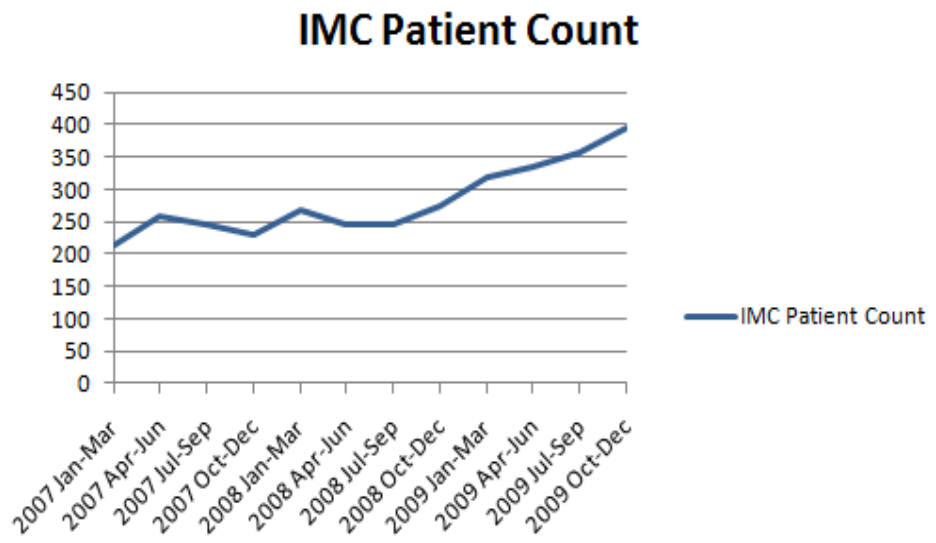

Figure 6.9: Number of patients admitted to IMC from the start of 2007 to the end of 2009.

triaging process." He did not offer an explanation for why the numbers in the first quarter are so much lower than the others.

Throughout the studies, we discovered a few bugs and a number of usability issues. First, *align-by-all* can be conceptually confusing, especially when filtering is applied. Analysts may have trouble interpreting the temporal summary when *align-by-all* is used. When filters and save selected/remove selected are applied, analysts expected the *instances* to be saved/removed instead of *records*, which Lifelines2 focused on. This is a major design problem because the visual representation differs from what is happening in the data. Since the first meetings on the Bounce-Back and Step-Up studies, I have made the changes so that filters operate both on records *and* instances – so the representation reflects the data faithfully. Secondly, the Bounce-Back study required the merging of a number of preexisting groups. We did this

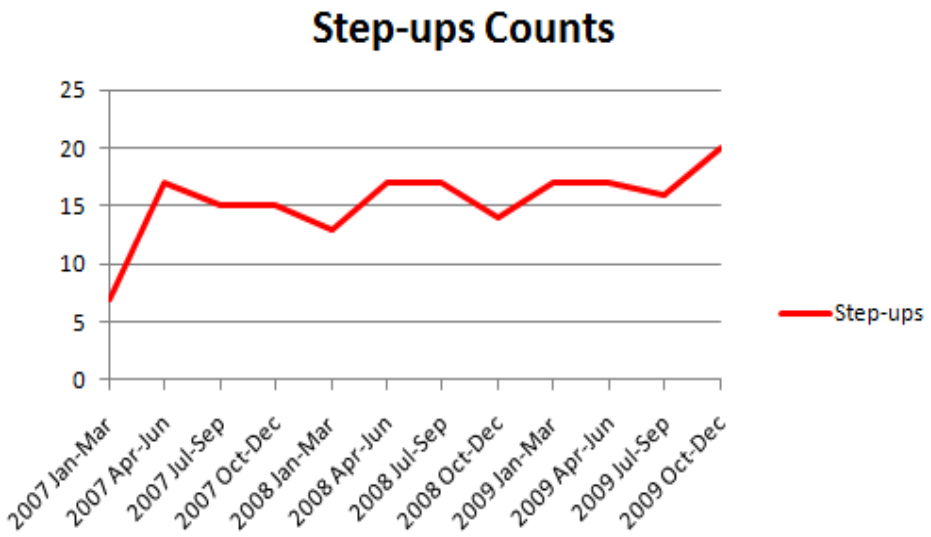

Figure 6.10: Number of patients who exhibited the Step-Up pattern from the start of 2007 to the end of 2009.

outside of Lifelines2 by merging the exported data from the groups. Ideally, there should be group operations in Lifelines2 that enable analysts to better manage the groups. Finally, I have implemented a small feature that allows analysts to jump to a specific patient record by ID. This was important as Dr. Ho would want to look at a specific patient when the data have been re-ranked.

When Dr. Ho drove the application alone, I observed that he had no problem using alignment, group selection/creation, rank, temporal summaries, and selections on temporal summaries. However, he did have a problem formulating his query into a sequence filter. He mentioned that he did not see the sequence UI immediately on screen so it did not remind him how to get to the filter. Other than the sequence filter, Dr. Ho was able to perform the Case Study for Step-Up in a matter of minutes by himself. He commented that, "The good thing about Lifeliens2 is its visual power

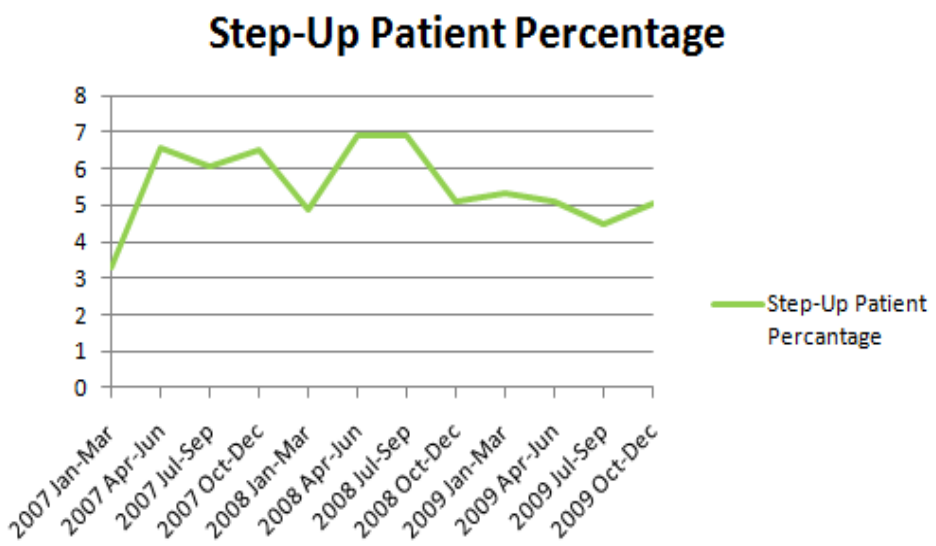

Figure 6.11: Percentage of patients who exhibited the Step-Up pattern from the start of 2007 to the end of 2009.

[...] I can visually look quickly to see if there anything amiss," and "I could perform the pattern-finding in matter of minutes, and feel that data is far more reliable at the same time." The dream feature for him would be to record the study and automatically repeat it for every quarter. He wanted it to be automatically done, and allow him to visually inspect the results so that he can manually change the results if he needs to. If his review of the process revealed something wrong, he would like to allow for different branches of the analysis to take place.

Dr. Ho had taken the results produced from our collaboration using Lifelines2 (numbers, Excel spread sheets, graphs, and Lifelines2 screen shots) to a physicians' meetings in the hospital. The consensus was that these percentages were well-within the boundaries – especially that it seems to be improving recently –, and changing triage procedures was probably not a necessary course of action at this point. They

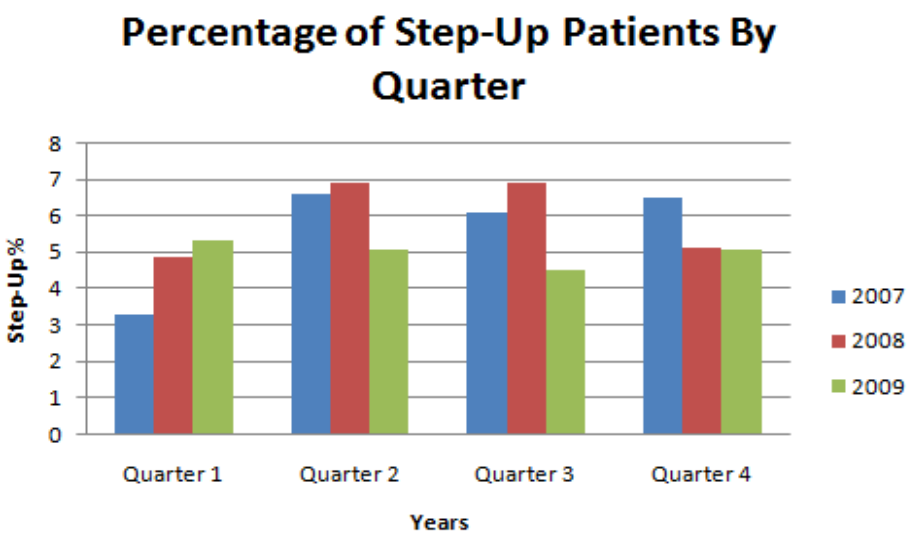

Figure 6.12: Percentage of patients who exhibited the Step-Up pattern from the start of 2007 to the end of 2009 by quarters.

noted that the analysis is interesting, and would love to keep performing the same analysis for as long as possible to build an historical baseline and so every quarter in the future can be evaluated the same way. Furthermore, the data can be used to compare to the numbers from other hospital care systems, for the purpose of hospital metrics.

Dr. Ho had been working with me for five months. Because part of his job at MI2 was to discover and test new technologies, he was able to spend more time (roughly once every two weeks) meeting with me and evaluating Lifelines2, which is far more frequent than the other physicians. Throughout these months, he was able to come up with many case study scenarios where Lifelines2 could be used. However, not all of them had data that were easy to obtain. We started and obtained some results for the Step-Up and Bounce-Back cases, but we also started on a number

of other scenarios, which I do not have preliminary results to report. He said that, "Lifelines2 would save me so much time to deal with all the different scenarios.", "I would never have to spend hours to write broken Excel scripts that produce low-quality data ever again!". Spending months working with Lifelines2 also changed how he views clinical data. In particular, when I asked him about Lifelines2, he said that, "Yeah, alignment was very different idea, yet so natural. I think about clinical problems in terms of alignment now, but none of my coworkers does the same."

6.4 Lifelines2 Feature Usage

These case studies showed how Lifelines2 has been beneficial to analysts in a variety of different domains. In the mature adoption scenarios (the heart attack incident in relation to day light savings time study and the drug trafficking communication network studies), the analysts found the features of temporal summaries in conjunction with alignment the most useful. In the drug trafficking case, the analyst was able to try different sentinel events while focused on the communication frequency of important actors to discover an important actor in the dataset. In the heart attack incidents in day light savings time change scenario, the situation was more complex. Because of the small data size and a lack of automatic aggregation in the granularity of weeks, temporal summaries did not show strong evidence that the original study claimed. Despite this, the analyst was able to get a sense of what the distribution of heart attack events looks like, something the original study never demonstrated. Instead, the most important feature that allowed the analyst

to mirror the published study was the automatic tabulation of heart attack events in temporal summaries. The fact that the temporal summary was configurable to different temporal granularities made the counting very easy. In these scenarios, comparison features and filters were not used.

In contrast, in the on-line book reading study of ICDL logs, the most important feature was the visual representation of the records. The separation of different event types to different horizontal space and the temporally placed events gave the analyst a good sense of the overall pattern. The fact that the event types (page numbers) also have an order made the visualization very clear, and conveys several different reading behaviors saliently. Alignment was useful in aiding the visual exploration of the data much as the experiment in Section 3.4.1. The studies in the mature heparin-induced thrombocytopenia (Section 5.3.1), graduate student academic progress (Section 5.3.2), and hospital room transfers (Section 6.3.4) used a richer set of tools in Lifelines2. ARF, temporal summaries, and grouping operators were used regularly. The first two studies also used group comparison features. The last study used filters and selections in temporal summaries extensively.

As part of the evaluation process, I had kept a detailed log on the usages of Lifelines2 operations. Some of the case studies had no logs because the data are confidential such as the drug trafficking case or the graduate student academic progress case. The collaborators did not want to de-identify the logs and send to us. However, all of the case studies with the hospital data are included. The Lifelines2 log is designed so the log itself is in Lifeline2 format, and can be read by Lifelines2 for analysis purposes.

6.4.1 Log Data in detail

Since September of 2008, most of Lifelines2 features were complete. Since then, Lifelines2 had been logging all the user actions in a log directory. There were a total of 2477 logs. However, some of the logs were not due to exploring the data or demonstrating the exploration results to collaborators. Many of the logs were short, and no additional data was opened aside from the default file. These are indicative that the session was due to testing and debugging uses. After removing the very short sessions, those that do not open at least one other file, and those that had logged no operators, I was left with 426 sessions (Figure 6.13). These logs contain all of the case studies in which we collaborated with the Washington Hospital Center. The temporal summaries in the figure show the number of events of *Align*, *Rank*, and *Filter*. The minimal amount of activity in 2009 Jan represented winter break. The lack of any activity in July and August of 2009 represented my time spent in an internship. The peculiar spike in October of 2009 represented when Dr. Ho and I started working on the hospital transfer data. The amount of operations in Lifeliens2 was reflective of the fact that the study involved many steps, and over 15 different datasets.

Table 6.1 (Page 185) summarizes the logs of the usage of Lifelines2. The operations are broken down into the following five main categories: ARF, Temporal Summary, Comparison, Data Operations, and Navigation. The raw number of counts, counts per session, and percentage of sessions that logged such operations are presented.

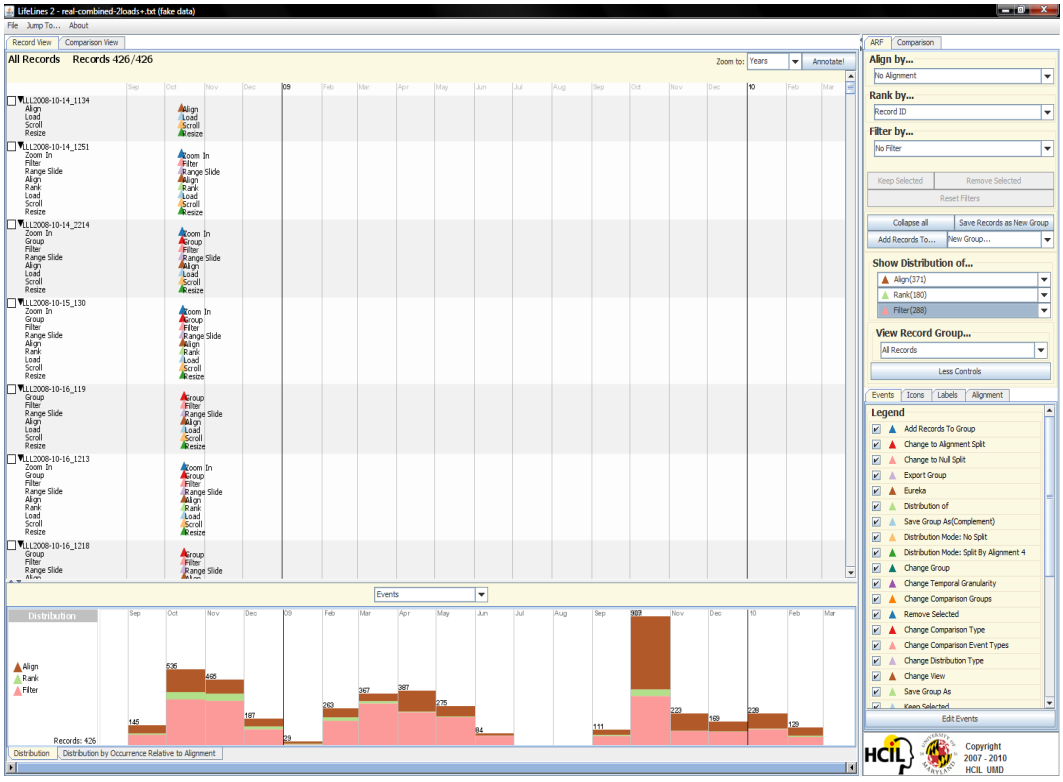

Figure 6.13: Logs of Lifelines2 usage are analyzed in Lifelines

With respect to the align-rank-filter framework, *Filter* was the most-frequently used operator. *Alignment* was second, and trailed by *Rank*. However, a larger percentage (87%) of sessions recorded at least one use of *Align*, while only 68% had any *Filter*. While *Rank* was useful to reorder the records by their event counts, it was ultimately not a vital operator in the studies. When pairs of the *Align*, *Rank*, and *Filter* were looked at as sequences, *Align* -> *Filter* and *Filter* -> *Align* occurred in 250 (59%) and 211 (50%) sessions respectively. *Align* -> *Rank* and *Rank* -> *Align* occurred at 162 (38%) and 123 (29%)sessions respectively. Finally, *Rank* ->

Filter occurred in 148 (37%) sessions, and *Filter* -> *Rank* occurred in only 48 (11%) sessions. When looking at a sequence of three, the break down (number of sessions that had the contained the sequence) is as follows: ARF (*Align*->*Rank*->*Filter*): 123, AFR: 41, FAR: 37, FRA: 27, RFA: 104, and RAF: 87. These numbers indicate although *Rank* is the least popular of the three operations, when it is used, *Rank* is typically used prior to *Align* or *Filter*, or both. On the other hand, *Align* and *Filter* are used fairly frequently, and which one tended to be used before the other depends on the analysts' tasks.

30% of the sessions used temporal summaries, and 24% used selections in temporal summary. The average number of these operations across all sessions were over 1 per session. This means that in sessions that these operations were used, they were used many times, so much that the average count per record is brought up.

The operations under the Comparison feature only occurred in 11-13% of all sessions. However, analysts tended to change the event types in the comparison and the groups in the comparison heavily. The type of comparison (Between Group/Within Group/Both) or the type of distribution type (Event/Reocrd/Event Normalized By Record Count) were less frequently used.

In these case studies, analysts tended to use *Keep Selected* as opposed to *Remove Selected*. *Save Group* occurred in 29% of the sessions while *Change Group* occurred only in 23%. This indicates that sometimes analysts would save a group, but not change the group otherwise. This occurs not because the analysts do not look at the newly created group. Instead, this is because Lifelines2 automatically

brings the analysts to the newly created group without having them perform the group change themselves. *Change Group*, as expected, is used more frequently than *Save Group*.

The first thing to notice in the navigation operations is that *Scroll* is a dominant operation. Every session involved scrolling, and on average, each session has more than 16 scrolls. Each scroll is computed only when an analysts lets go of the mouse on the scroll bar or the up/down keys. Changing the *Time Range Slider* was a distant second in usage in this category. In contrast, *Change Granularity* (temporal granularity) was not as popular. This may be attributed to the fact that Using the *Time Range Slider* analyst can control the temporal range more finely. Even after using the cruder *Change Granularity*, an adjustment in the *Time Range Slider* was often necessary. *Zoom In* was used more often than *Zoom Out*. *Collapse Record* and *Expand Record* were the least used features.

6.5 A Process Model for Exploring Temporal Categorical Records

Combining the log data, observations with the collaborators, and interviews and comments with the analysts, I constructed the following process model for exploring temporal categorical records.

1. Examine data in visualization for confidence (overview/browse)
2. Exploratory Search
    (a) Iteratively applying visual operators

(b) Evaluate Results of manipulation

(c) Deal with unexpected findings

3. Analysis, Explanation

(a) Examine path of search as a whole

(b) Determine to what extent the questions are answered

i. At the limitation of the system

ii. At the limitation of the data

(c) Refine existing questions

4. Report results to colleagues

(a) Document discovery

(b) Disseminate data

5. Move onto new questions

The exploratory process model presented here does not include the data acquisition stage, which occurs before the exploration begins. However, it is worth mentioning it because sometimes the exploratory process leads analysts back to data acquisition. In the data acquisition stage, analysts decide the scope of the data that they want to examine. When analysts are not satisfied with the data, found systematic errors in the data, or want to incorporate more data for deeper analyses, they return to this pre-exploratory stage. In my case studies, this stage often takes a long time. The reason for the lengthiness in acquiring data lies in the complexity

of the data, infrastructural or organizational barriers. The requisite data may reside in different databases, named using different IDs or codes. They may be difficult to search, and may require first finding someone who knows where it is.

One of the most common results of analysts looking at their own data through a visualization technique for the first time is the surprise that there are artifacts in the data (systematic errors, lack of consistency, etc.). This is because they have never seen it in an effective format before. As such, my collaborators would cursorily browse and sometimes examine in detail the data to make sure the data reflects what they know. Sometimes they would find strange artifacts. For example, when the data in mature heparin-induced thrombocytopenia case study was first converted, my physician collaborators found patients who were given drugs *after* they had been discharged dead! In this particular case, the discovery was made by playing with the Lifelines2 interface by showing the distribution of drug prescriptions and aligning by discharged dead events. The summary showed suspicious timing, and when we tracked the patients down, we found 7 of them were prescribed drugs after they had been discharged dead. All 7 cases occurred within one hour of their discharge dead events, and we decided that we could safely assume that this occurred because the drug database must have recorded the prescription event with some delay. This also raised questions on how reliable the time stamp was, and whether subsequent case study would be affected. We eventually found better data to circumvent the problem.

After the analysts have gained confidence in the data and the visualization (that the visualization was not giving false impressions somehow), they move onto

Stage 2 – exploratory search. They would start seeking answers to their preconceived questions or finding evidence for their hypothesis. However, in the process of seeking answers to one question, new questions often spawn when they notice interesting or unexpected data. At this point they would utilize their domain knowledge to try to explain what they see (for example, narrate about certain patient records to aid their reasoning), or they would write down the new question for later exploration (for example, noticing that a characteristic of the event pattern and needing a way to handle them).

During the exploration, there may be different strategies. I have observed that analysts would apply alignment on different sentinel events in the same exploratory session to look at the data in different views. By using different alignment while showing distribution of certain events they care about, they aimed to find useful or telling sentinel events (Chris on the drug trafficking communication network case study reported this same strategy). Some analysts would use a more traditional way for exploration: actively manipulate the display by ranking, filtering iteratively, or changing the temporal summary. Regardless of the strategy they used, alignment remained the strongest indicator on what focus they have on the data. The changes in sentinel event indicate a change of exploratory focus. Sometimes when the collaborators had aligned by one event, and realized that alignment would not lead them to the information they wanted to see, they would reformulate the question and use a different alignment. This had been observed with Dr. Ho, and in meetings with Dr. Marchand and Dr. Smith (where they dictated changes of sentinel events).

Another important aspect of the exploratory search was that at each step of the search, the analysts were very keen on what happens to the data. The Lifelines2 log data showed how often the analysts used *Scroll* to view records in the dataset. There are two hot spots where many scroll operations are performed in the observation. The first is when the analyst is checking the data to make sure it does not contain errors or missing data. The second is after align, rank, or filter had been applied. Scrolling to see records in a new order by *Rank* is common. However, my collaborators found it was just as important to examine the data after *Align* and *Filter* operations. These operators can change the amount of data that is being visualized, and my collaborators want to keep an eye on the results of these operators.

Aside from manual scrolling, my collaborators would also keep an eye on the distribution of their favorite events in the temporal summary and the number of records in view. I observed that when align, rank, filter and group operations are applied, the analysts would focus on these two things. They give the analysts a global feel of how the data is changing when they apply a variety of operations. In fact, in cases where heavy exploration was required, as in the early adoption heparin-induced thrombocytopenia case study (Section 6.2.1), I noticed that the physicians, including Dr. Marchand, kept their eyes fixed on the temporal summary as a variety of filters are applied. When the Dr. Mukherjee and I worked on the mature heparin-induced thrombocytopenia study (Section 5.3.1), he also showed the same tendency – focusing on the temporal summaries as the data is being sliced and diced. By fixing his attention on the temporal summaries, he could get a good

sense of what changed in the data, and how the operations he had chosen to perform changed it. He could then decide if he was on the right path of exploration. If he did not like a filter he previously applied, he would backtrack to that previous state (a previously created group), and rethink his approach. In fact, my collaborators would even use the comparison feature on several previously created groups to see if the paths of exploration seemed to be a fruitful one. Temporal summaries became an indispensable mini map to the thousands of patient records that cannot fit the screen. Although focusing on temporal summaries was quick, my collaborators would still examine the records individually when they had the chance, though not exhaustively.

When my collaborators encountered unexpected discoveries, they would save the current state as a new group, and make a mental note to they themselves to come visit the saved group. When they wanted to continue exploration the next time using the data, they would export the current data into a new file. When they found something noteworthy, annotation was used, sometimes in conjunction with the built-in screen capture in Lifelines2 – although annotation was recorded to only occur in 5% of all logged sessions (not in Table 6.1.

When my collaborators arrived at an interesting point where their questions might be answered, they would use their domain knowledge to analyze what they saw, and offer explanations (Stage 3: Analysis, Explanation). They would verify how they got there by looking at the groups they created before, or by examining them all at once in the comparison view, as if they were double-checking their work. When they were satisfied with what they saw, they would examine the data, and

decide whether their questions can be, or are, answered. Sometimes a dead-end was reached, and they would realize that more data was required. We would then stop the exploration and discuss how to get the additional requisite data, and how long it would take to get it, clean it, and assimilate it. However, sometimes the dead-end was encountered because we were at the limitations of Lifelines2. This occurred when the analysis in a case study required features Lifelines2 simply does not support (*e.g.*, numerical values, percentage drops). In these cases, the exploration came to a halt. Unless we had found a solution (a good way to bin the numerical values into categorical values, for example), the case study would discontinue. Sometimes the limitations of Lifelines2 could be compensated by other systems. As in the heart attack and daylight savings case study, we resorted to using Excel as a platform to compute our results. In these cases, the case study may come to a stop with fruitful results. As a result of the end of the exploration, our collaborators sometimes realized that they needed to refine their questions because they could not proceed with the original one with the data available, or that a different spin on the question is more exciting – and the spin would not have revealed itself without the prior exploration.

Finally, when a case study is completed to some satisfactory point, my collaborators would prepare their findings. Dr. Ho routinely asked me to send him subsets of patient records that represent the fruit of our labor, along with screen shots, annotations, and spreadsheets if we created any. He would take these data to show his fellow physicians or his supervisor to argue for the usage of technologies such as Lifeines2. Overall, I had not encountered that the screen shot needed to be

modified for other non-collaborators' consumption. Our collaborators are capable of explaining what is happening in a Lifelines2 screen shot. When an exploration ends, a new one often begins. Though the new one may just be a branch from the earlier exploration!

6.6 Recommendations

The case studies, Lifelines2 logs, and observations revealed some interesting user behaviors when dealing with records of temporal categorical data. They also revealed strengths and weaknesses of Lifelines2. I generalize these into the following recommendations for future developers of visualization tools for multiple temporal records.

1. **Use Alignment** The usefulness of alignment was evident in the Lifelines2 logs, the case studies, and from observations. The user logs corroborate the findings of alignment in the controlled experiment – alignment is useful, users recognize it, and as a result they use it often. When dealing with a large number of temporal categorical data, the ability to use alignment to impose a strict relative time frame was important to my collaborators. It allowed for better visual scanning of the data, and the dynamism of alignment allowed the analysts to quickly switch perspectives and focus if they need to. The idea of "anchoring" data by data characteristics for exploration had been successful in other visualization systems, and alignment seems to be one natural version of it for the temporal domain. Developing future visualization systems for

records of temporal data should leverage on alignment for its power, flexibility, and wide range of applicability. However, there may be other "anchoring" techniques in temporal visualization in perhaps different situations, and it is worth pursuing them.

2. **Show Details** One surprise finding was that the analysts liked to look at the details of the records. *Scroll* was the most frequently-used operation, and it is indicative of how much the analysts liked to view records in detail. Seeing the details and being able to compare the detailed records that are close to one another seem to reassure the analysts that no data are missing, broken, or lost along the visualization pipe or the analysis process. My second recommendation is that detailed depiction of the records is important. Even if the primary view of the data was to be in an overview display, the analysis tool should always make the details accessible to analysts – and preferably for many records at once.

3. **Overview Differently** Lifelines2 provides an overview in the form of temporal summaries. While it is perhaps the most important feature after ARF (and one of the most frequently used features), more ways to overview the data may be beneficial to analysts. The observations indicated that for iterative filtering tasks, analysts tend to focus on the overview most of the time, despite looking at records in detail as well. They would use the comparison view as a heavyweight overview to examine their exploration process, but a lightweight overview would be better (more accessible, and without having to

take them away from the visualization of record details). Additionally, other types of overviews would give analysts a better idea of how their data changed since they applied an operator. For example, a temporal summary provides a "horizontal" overview. A "vertical" overview that represents the records would complement temporal summaries, and be a good way to augment the analysts' understanding of the data. Furthermore, a good vertical overview design may reduce the amount of *Scroll* necessary.

4. **Support Branching Exploration** Analysts naturally create different sets of data as they explore. They also naturally perform backtrack and branching in their exploration. The features in Lifelines2 that support branching in exploration are *Save Group* and *Change Group*. This rudimentary support for branching exploration and backtrack was both used fairly frequently (averaging almost one use per session in the log). Better branching support and history keeping for backtracks may be one of the most important practical feature.

6.7 Summary

I report on the details of both the early adoption and more mature case studies of Lifelines2. My collaborators were able to identify interesting temporal patterns and event trends. They were able to obtain concrete results that were publishable in the ICDL case study. In the drug trafficking case study, my collaborator was able to find novel data trends. In the case studies on hospital room transfers,

my collaborators were able to debunk theories based on anecdotal evidence, and prevent lengthy and expensive overhaul in how they triage patients. Despite the identification of additional usability problems and features that can strengthen the capabilities of Lifelines2, my collaborators continue to be enthusiastic about using Lifelines2 and incorporating it as part of their daily analysis tool. The anecdotal evidence from my case studies provide support for the value of the features I have designed and implemented in Lifelines2.

With observations from these case studies, logs of analyst actions in the Lifelines2 system, and interviews and comments, I generalized and presented a process model. Furthermore, since the evaluations and observations revealed much of the strengths and weaknesses of Lifelines2, I made several design and feature recommendations to future temporal categorical data visualization designers.

Operation	Count	Count Per Session	% of Sessions
ARF			
Align	1680	3.94	87%
Rank	260	0.610	42%
Filter	2564	6.019	68%
Temporal Summary			
Show Summary	623	1.46	30%
Temporal Selection	531	1.25	24%
Comparison			
Event Type Change	406	0.95	11%
Comparison Type Change	79	0.18	12%
Group Change	406	0.95	12%
Distribution Type Change	179	0.42	13%
Data Operations			
Keep Selected	400	0.94	30%
Remove Selected	96	0.23	18%
Save Group	409	0.96	29%
Change Group	687	1.61	23%
Navigation			
Zoom In	646	1.52	30%
Zoom Out	157	0.37	13%
Time Range Slider	1865	4.38	29%
Change Granularity	217	0.51	14%
Scroll	6840	16.1	100%
Collapse	55	0.13	8%
Expand	30	0.07	5%

Table 6.1: Operator usage in Lifelines2

Chapter 7

Conclusions

With an increasing growth of data in the form of temporal categorical records today, it also becomes increasingly important to design effective ways to allow analysts to interact with such data. This dissertation describes four main contributions. Three are technical supports for different aspects of searching, and one that describes a process model and design/feature guidelines for future visualization designers. A summary of the core chapters and the main contributions is provided below.

1. **(Align-Rank-Filter Framework)** Alignment was designed to allow relative visual comparisons of temporal categorical data. By focusing on a specific sentinel event, analysts can better understand the data trends immediately before, after, and around the alignment. The effectiveness of alignment is evaluated in a controlled experiment, which showed overwhelming advantage for certain tasks when alignment is used.

2. **(Temporal Pattern Search Algorithm)** A common task in interactive visualization is to filter. A common filtering task in dealing with temporal categorical data is to filter by patterns. I designed an algorithm (TPS) that uses the combination of separate arrays of event data and efficient binary search to perform this kind of pattern search. The TPS algorithm performs better than the NFA or the Shift-And approaches using real data and patterns. When ex-

perimental random data is used, Shift-And performs better in short patterns, but loses to TPS when the patterns are longer. Unlike Shift-And, however, TPS is extensible, and can be made to handle additional constraints. TPS was designed to suit analyses tools such as Lifelines2 because each sequence filter does not require a preprocessing stage, and can properly handle cases where there are simultaneous events.

3. **(Temporal Summaries)** I present temporal summaries, a visualization technique that allows dynamic aggregation of event types over time in different temporal granularities. It is used to support overview of event types over time, and subsequent comparison among different record groups. Temporal summaries are also used to allow analysts a visual way to perform temporal constraints that had otherwise been intimidating to analysts when faced with UI widgets. The value of temporal summaries was evaluated in a series of case studies.

4. **(Process Model and Future Design Guidelines)** I then detail the case studies of Lifelines2 and discuss in detail how the features of Lifelines2 are utilized in these cross-domain real scenarios. From the experience of working with collaborators, Lifelines2 logs, and analyst interview and commentary, I generalize the exploratory analysis process for temporal categorical records. I also present some recommendations for future visual analytical system design.

With the important national health care reform on the horizon and health-care related fields moving towards electronic health records (EHRs) for maintainability

and affordability, EHRs have been a focus of my collaboration efforts. While many have devoted their time and energy in designing better storage, indexing , retrieval, and data conversion systems, not nearly as much effort has been devoted to adding value to the information that is stored. Interactive search techniques presented in this dissertation present one of the several starting efforts to add value to the existing infrastructure, bringing the value of the information to analysts who want to study them.

7.1 Future Work

There are a number of possible future directions. Some are more intellectually more interesting, while others are pragmatically important.

7.1.1 Scalability

The Lifelines2 system has been tested with up to 100000 records and over 1.5 million events as is. The system can hold the data all in main memory while generating the visual representations and executing the various visual operators. However, these numbers are not enough if Lifelines2 were to support live databases in a hospital, where millions of patient records may reside. There are, however, a few ways to extend the scalability of the system.

7.1.1.1 Using Fewer Shapes

Lifelines2 creates one independent shape to represent each individual event. However, the shapes often overlap when the visualization is zoomed out sufficiently. Instead of using multiple shapes which uses up precious memory, a single shape would conserve memory, and perhaps allow faster rendering. This will require computation before rendering to see if events are overlapping. However, with position of events and size of shapes available at time of rendering, this computation would be fairly straight forward. This new rendering method, however, would create a small drawback as it would require additional computation to decide which tooltip to see – since tooltips are no longer tied to specific event shapes. A related issue is the completely overlapped events. When the visualization is zoomed out sufficiently, the shapes that completely overlap do not show any differently than other shapes that represent only single events. Some indication, perhaps size adjustment with possible a numerical designation for the number of overlapping shapes, would be worth pursuing.

7.1.1.2 Dealing with Massive Amounts of Live Data

In order to deal with data larger than what the RAM can handle, Lifelines2 will need to manage the records in indexable page files on the hard disk, and will only cache what is necessary in memory. The page files will be in a format that can be easily read back quickly into the data structure Lifelines2 uses, as to avoid additional data processing when it is being read/written. One possibility is to Java Objects

which can be read in/written out efficiently. The access to the page file will need to be asynchronous to the main interaction thread as so the interaction experience is not disturbed by the potentially long-running action. By the same token, each asynchronous action should be cancelable by the user. As the access to the data is asynchronous, most of the operators such as *Align*, *Rank*, *Filter*, *Summarize*, will also be asynchronous. Caching of records that are recently on-screen in memory can help the performance. Dealing with live data presents another challenge. Assuming there is a database that accepts changes and updates to the records Lifelines2 is visualizing, every time the data being visualized is changed, the data in Lifelines2 and its page files need to be updated. If the changes occur so frequently that it can impair the normal interactive experience of Lifelines2, the updates can be updated in batches in set intervals, or when the updated data is being accessed by Lifelines2.

7.1.2 Extension to Alignment

I have presented alignment by single event types and its effectiveness as a visual aid for relative comparison tasks. However, alignment can be further extended in a few different ways. Although they were not critical in the case studies, they are worth exploring. These extensions are ranked in the order of ease of implementation.

7.1.2.1 Tight Integration with Sequence Filters

Integrating alignment as a part of search has been implemented in *temporal summaries* in Lifelines2. However, it is restricted to the events in the summaries,

and no additional temporal patterns can be imposed. One possible way to extend alignment is to integrate it tightly with the sequence filter. In addition to all the event types and their absences, an *Alignment Event* option can be available when an alignment is in effect. This will allow analysts to specify temporal patterns in the context of the alignment. For example, if the data is aligned by the 3^{rd} *Heart Attack*, a normal sequence filter can specify *High Blood Pressure* `->` *Heart Attack* `->` *Stroke*, but the *Heart Attack* is not necessarily the 3^{rd} one. With this extension, analysts can use *Alignment Event* to specify exactly the 3^{rd} *Heart Attack* in the sequence filter. This gives more expressiveness in specifying temporal pattern filters, and this is similar to the extensions to alignment done in other projects [120, 58]. A straightforward implementation in the Temporal Pattern Search (TPS) algorithm is easy. The *Alignment Event* option is first translated to the actual event type by which all the records are aligned. Then in TPS, whenever a potential match to the alignment event is found, another check is performed (in the `NEXT(...)` function) to see if the matching event is indeed the event by which the record is aligned. Only if the event is the alignment event will `NEXT(...)` return it to continue the matching process.

7.1.2.2 Alignment on Sequences of Events

A direct extension to aligning by a single event is to align by a sequence of events. Two different designs can be imagined. First, the sequence of events to be aligned will occupy the same vertical strip just as the normal alignment. The strip that contains the sequence, however, will not represent a temporal durations

(just as the current alignment design). For a given sequence and the events that match the sequence, let the first matching event be m_1 and the last matching event m_m. All other events in the record that occur between m_1 and m_m and m_1 and m_m themselves will be fitted in the strip. All events before and after the sequence will be laid out the same way as the current alignment design. By warping different temporal duration into the same vertical strip across records, the actual amount of time that take place between m_1 and m_m will be difficult to tell, and this may create a lot of visual noise in the alignment strip and, worse, give analysts wrong impressions about the temporal durations.

A less temporally disturbing model is to display all records that contain the sequence that the analyst wants to align by. Instead of aligning all the elements of the sequence simultaneously, the analyst is given a choice to align by a chosen element of the sequence, much like the current alignment design. Subsequently, analysts an also choose a secondary alignment by choosing another element from the sequence, which will make the same kind of warping behavior as described in the previous method. By making the time-warping an explicit action, hopefully this can make the disturbance on the time less hostile, and provides more control to the analysts. Finally, temporal distances to an alignment by a particular matching event on the sequence can be used to sort the matching records for better view.

7.1.2.3 Automatic Discovery of Useful Alignments

One interesting observation from the case studies was that analysts sometimes try a variety of alignments to see if one changes the over all distribution of some other important event types. The *Alignment* control combo box makes this fairly easy to do. Once a selection is made, analysts can simply use the up/down keys on the keyboard to try different sentinel events to align by. While there is value in trying different sentinel events – one may see something that is unexpected or informative – for the purpose of finding "interesting" sentinel events, an automatic approach will be easier. The difficulty lies in how to define "interestingness." For example, rise in the distribution of event "A" prior to the alignment within certain time constraint followed by the sentinel event and then followed by a rise in the distribution of event "B" may be considered interesting, but there are many more. Finding an interesting alignment with respect to any event type may be too difficult a problem to start with. However, by choosing some important events ahead of time, the algorithm for automatic discovery may be more feasible. For example, the temporal summaries of events can be converted into a form that Shape Language proposed by Agrawal et al. [3] can apply. Then by defining a *Sharp Rise* in "A" followed by alignment and then followed by *Sharp Drop* of "B" and other interesting patterns, they can be searched and results organized and ranked by a heuristic function that measures "interestingness." The results can then be presented to analysts as a ranked list.

7.1.3 Information Density and Alternative Summaries

One complaint about the record visualization in Lifelines2 is that there is much wasted space on screen. There is an argument that the lack of events in certain time periods conveys important information as well. However, the Lifelines2 log suggests that analysts tend to scroll to examine the records visually. A more densely-packed information display or effective alternative summary design could reduce the need to scroll.

7.1.3.1 Better Utilization of the Vertical Space

One cause for the lack of information density is the design to place only events of the same type on the same horizontal space. While this design avoids the occlusion of different types of events, particularly for medical events that can occur very frequently in a short amount of time, it is also spatially expensive. One way is to merge event types that can be merged in the same horizontal space with minimal occlusion. And if occlusions occur, a visual representation for the occlusion will be important. For event types that cannot be merged onto the same horizontal space, lessen the vertical space between each event type rows, so the event themselves can partially overlap is one possible solution. Designing an algorithm that detects what event types can be merged is an interesting path of future work. However, these space-saving strategies present some usability challenges. The merging of horizontal space means that the color and spatial cues for different event types are reduced to a single one (color). Analysts will only be able to discern one event type from

another by its color. Furthermore, the title for the event types to the left of each record will not be able to accommodate the merged event types, and the reliance on the legend itself becomes more important. The lack of labels combined with the reliance on color to discern event types can be very problematic when there are many event types and some have effectively the same color to the human eyes. One concern that this approach would not work very well is that Lifelines2 has the collapse feature where analysts can collapse the records into smaller representations. To conserve space, the collapsed records do not have title labels for event types, and events can partially overlap a previous event type. The Lifelines2 log indicates, however, that analysts are not fond of using this feature. Before the space-saving strategies are considered, a study needs to investigate whether the lack of labels and the overlapping are the primary reasons for the lack of the usage. If they were, then different approaches need to be taken. Instead of improving the information density of the individual records, other approaches rely on aggregation of selected features of the records. By directly supporting the tasks that the analysts are interested in when they scroll down to see the records, these strategies may be more effective.

7.1.3.2 Vertical Summaries

Temporal Summaries have provided a way to aggregate events over time (horizontally). A different approach is to summarize the records *vertically*. For example, the *Rank* operator ranks records by the number of events of a specific type. However, the distribution of the events across records is very difficult to see. A vertical sum-

mary can show the distribution, and allow a more informative cut-off point should analysts favor one.

7.1.3.3 Sequence Summaries

One primary reason for analysts to look at the records in detail is trying to get a sense of the predominant patterns, or see the variations of a particular pattern. Lifelines2 would be a more effective tool if the summary of temporal sequences (especially the ones relative to an alignment) can be summarized effectively. This is an example of vertical summary. The summary will help analysts answer the following questions (1) What are the predominant sequences of events (from a certain start point such as an alignment)? (2) How "popular" are the predominant sequences as compared to less popular ones? (3) What are the variations in the sequences? The variations can depend on what events tend to occur in-between the matching sequence, or how much time generally pass between the matching elements of the sequence. This summary can save analysts work in generalizing the data in terms of sequences, and provide affordance to select records based on their event sequences and temporal variations on the sequences. One caveat in providing this feature is that it is not clear how events that occur at the same time can be supported if events on a record are viewed only as sequences.

7.1.4 Practical Concerns

These future work items have less research value, but are identified through my case studies as important components that will make a system like Lifelines2 more effective in real scenarios.

7.1.4.1 Extension to Other data Types

As a dissertation project, Lifelines2 has focused solely on categorical data. In the real world, other data are important in decision-making. The early adoption case studies revealed that physicians would like to include numerical data in the visualization tool. Instead of viewing the high/normal/low segments of blood pressure, the actual numbers should be plotted, and searchable. The physicians would like to have ways to search only by categories, only by numbers, and by a mixture of the two. A second extension is to include attributes of the records (*e.g.*, patient attributes in the domain of medical records). *Gender*, *age*, *ethnicity* are some canonical attributes in patient records that may be important factors to clinical decisions. Making these attributes filterable and rankable would be useful to analysts.

7.1.4.2 Scripting for Automation

As a number of the case studies are about measuring the quality of health care in an institution, it goes without saying that a longer-term monitoring will be more useful. To perform the same kind of analysis, some of the actions can be automated – such as what sentinel events to choose, and kind of sequence filter to apply. As

long as the results can be reviewed and explored manually, and actions overwritten by analysts, the automation would be a time-saver.

7.1.4.3 Automatic Conversion from Database Schema

Perhaps the largest bottleneck with my case studies is the gathering and the refinement of data. The iterative process of getting the right data can be long. Ideally there would be a component that allows the search on the database and converts the search results to Lifelines2 format for visual analysis. While this is done with our collaborators in the i2b2 project [70] (Figure 3.8), the integration was largely easy because their system already outputs formatted XML documents with rigid schema. In general, an automatic conversion from the database search results will require significant amount of tailoring to the individual database schema. This is no easy task, since there are often numerous databases, each with its own schema in the same system. This can be a large undertaking, but this will allow the exploratory process to be faster, and let the analysts drive the exploratory search from end-to-end.

7.1.5 Process Support

One interesting lesson I learned from the case studies is that as analysis tools become more and more complex in terms of features, optional configurations, and operations, analysts need more and more support to deal with their data, findings, and the entire process. While different data domain and different types of analysis

have different features, there are certain commonalities. I envision a standard set of process support features in all future analysis tools (particularly visualization analysis tools), and here I list a few items to serve as a start of the line of research.

7.1.5.1 Knowledge Provenance

As analysts gather, view, and slice and dice data in order to gain the insight on that they seek, the process of how they get to that point should be preserved. By preserving the path taken, other analysts can follow the path and decide if the conclusions drawn by the original analyst is correct. For example, others may discover that one action taken by the original analyst reflects an assumption that was not reported. Or that the data used for the analysis is somehow flawed. The preservation of the exploratory analysis path can be used to invite external reviews, obtain external validation, increase discussions. Furthermore, the path tells others how certain conclusions or insights are gained. I think preserving how knowledge is gained from analysis tools is an important first step. A history keeping component such as the one in Photoshop coupled with a temporal visualization and annotation would provide the most basic needs. A more sophisticated one would include the ability to playback an existing analysis path.

7.1.5.2 Extending Grouping Support to Analysis Branching

A next step would extend the preservation of analysis path to allow analysis branching. An analyst can follow an original analysis path and branch off a new

analysis of his/her own independent of the original one. The grouping support in Lifelines2 is but a rudimentary support for analysis branching, and it does not have a complete history keeping to allow analysts to roll back to particular analysis states. The collection of analysis paths can then be taken as an aggregate to see what interests users. One can potentially compare the different analysis paths and investigate which part of the data is over-analyzed, and which part is under-analyzed, and finally determine whether such discrepancies may represent a bias in the analyses. Finally, the entire analysis framework can be deployed on the Web, much like ManyEyes [112] and sense.us [36], to promote shared, collaborative analysis of data and elicit comments and feedback.

7.2 Summary

The main results of this dissertation are presented. Each technical contribution is presented with its evaluation results. I present a list of several potential future work directions. The breadth and the depth of these directions suggest that the research presented in this dissertation merely opened the door for more innovation on the topic of searching temporal categorical data. The many opportunities for better visualization and interaction support indicates that this topic remains an intellectually interesting and rewarding research direction for the future.

Appendix A

Lifelines2 Software

Lifelines2 is written in Java. It requires Java 1.5 or above to compile and run. It also requires the 2D graphics libraries `piccolo.jar` and `piccolox.jar` [10] [1]. An optional package `automaton.jar` is used as part of the TPS algorithm evaluation (Chapter 4), but it is not essential for running Lifelines2.

Lifelines2 contains a total of 27597 lines of code in 36 packages. There are three main packages: `external`, `lifelines`, and `test`. The `external` and `test` are non-essential packages of Lifelines2. The `external` package contains scripts for converting raw data from my collaborators to Lifelines2 format, specific XML reader for the outputs of the i2b2 system [70], and automatic random data generator for stress tests. On the other hand, the `test` package contains the codes that test and evaluates different parts of the Lifelines2 software. For example, the set of code for evaluating the TPS algorithm in Chapter 4 is in this package.

The `external` and `test` packages consist of 8580 lines of code (`external` contains 2922 lines), but are not essential to Lifelines2. Removing them will not prevent the compilation or execution of Lifelines2. They are kept as part of the Lifelines2 source code because they refer code in the `lifelines` package, and this organization simplifies the management of the code repository.

I will describe the core `lifelines` package in detail.

[1] Piccolo: `http://www.piccolo2d.org/`

A.1 Lifelines2 Data Model

I first describe the Lifelines2 input data formats, secondly the internal data structures, and finally, the various possible outputs from Lifelines2.

A.1.1 Input Data

Lifelines2 accepts two formats of data. The first is in plain text (`.txt`), and the second is in XML (`.xml`). The second format is specifically tailored to the XML outputs of the i2b2 system [70] for the ease of our collaborations. The XML reader is not meant for general purposes. In contrast, the first format is a multi-column format that can be easily created using spreadsheets or other database output mechanisms. I present only the first format as it is more widely used and I will refer to it as the "Lifelines2 format" below.

A.1.1.1 Lifelines2 Format

The basic Lifelines2 format consists of three columns. Each column describes an event. The first column contains the ID of records (which record does the event belong to), the second column contains the event type (of the event), and the third column contains the time stamp (of the event). Table A.1 shows a sample data of athletes with their sports injuries[2]. Columns are separated by tabs (not shown), and there are no column headers. Events do not need to be sorted (Tom). IDs

[2]A similar sample is downloadable from `http://www.cs.umd.edu/hcil/lifelines2/pages/simpleExample.txt`

Tom	achilles tendinitis	2002-07-08
Tom	jammed finger	2002-06-24
Jill	jammed finger	2002-04-18
Jill	torn ACL	2004-11-07
Alice	jammed finger	2005-12-01
Bob	achilles tendinitis	2001-02-07
Alice	jammed finger	2006-02-07
Bob	broken foot	2002-04-22

Table A.1: Sample data in Lifelines2 format.

do not need to be sorted (Alice). In addition to this 3-column format, Lifelines2 also accepts a fourth column as annotation for that event (any text). Note that all events in an input file must be in a 3-column or 4-column format, input files that contain both 3-column events and 4-column events may fail during read-in or miss data.

The Lifelines2 format file parser is `lifelines.PlainTextFileParser`, and the i2b2 XML praser is `lifelines.i2b2.I2b2XMLParser`. `lifelines.TextFileParser` is the parent class of the two input file parsers. It is designed to be extensible to other plain text data file formats. Lifelines2 does not currently have facilities to handle other types of data such as streaming data or networked data (*e.g.*, connection to a database). Well-known software-engineering design patterns such as the Singleton and the Factory patterns are used to make the automatic parser code clean and extensible. When users choose to open a dataset via Lifelines2 file browser (which only shows `.txt` and `.xml` files), Lifelines2 decides which file parser to use, and instantiates a `lifelines.TextFileParserFactory` to create either a `lifelines.PlainTextFileParser` or a `lifelines.i2b2.I2b2XMLParser` to handle the input.

Format	Examples
YYYY-MM-DD hh:mm:ss.SSS	2001-11-5 12:43:12.543
YYYY-MM-DD hh:mm:ss	2005-1-1 21:4:21
	2003-12-14 2:43:11
YYYY-MM-DD hh:mm	2007-3-26 21:10
	2006-12-22 3:22
YYYY-MM-DD	2001-11-24
	1997-1-6

Table A.2: Acceptable date/time formats in Lifelines2 format

Lifelines2 format accepts several time stamp formats (Table A.2), where Y, M, and D stand for Year, Month, and Day respectively. Similarly, h, m, s, S stand for hour, minute, second, and millisecond. The length of string describes how many digits are acceptable. A detailed description of the date/time format and the Lifelines2 format can be found online[3].

A.1.1.2 Customization File Format

In addition to the core input data, Lifelines2 also uses a plain text customization file (`.cstm`) to allow customization of a dataset across different sessions of Lifelines2. When users close an input file (by exiting Lifelines2 or opening a new file), Lifelines2 updates the input file's customization file. If there is not an existing customization file, Lifelines2 creates one – which records a variety of user preferences, for example, the size, color, shape of event representations – and stores the customization file in the same directory as the input file with the same names, *e.g.*, `inputData.cstm` for `inputData.txt`. Most of the customization settings are stored

[3]Lifelines2 format: `http://www.cs.umd.edu/hcil/lifelines2/pages/lifelines2InputFormat.html`

Variable Name	Utility
Dataset-Specific customizations	
`currentDetailedEventShape`	Shape for events (expanded)
`currentDetailedEventSize`	Size for events (expanded)
`currentCollapsedEventShape`	Shape for events (collapsed)
`currentCollapsedEventSize`	Size for events (collapsed)
`currentLabelPosition`	Position for event type labels
`currentDetailedShapeDrawer`	Drawer for events (expanded)
`currentCollapsedShapeDrawer`	Drawer for events (collapsed)
`inputDataIntegrity`	Is data fake, real, or de-identified?
`currentAlignmentStripeHalfWidth`	Width of the alignment strip
`currentShowCalendarMark`	calendar mark visibility when aligned
Event-Type-Specific Customizations	
`eventColorMap`	Event types' colors
`legendUIMap`	Event types' GUI in the legend
`aliasMap`	Event types' aliases
`visibilityMap`	Event types' visibility
`intervalManager`	Event Types' intervals of validity
Semantic Customizations	
`customization`	Organization of event types

Table A.3: Customization Variables in the class `lifelines.Settings`

in `lifelines.Settings` (`Settings`) , and the variables for customization are listed in Table A.3.

The `Settings` class uses a singleton design and is accessible globally within Lifelines2. When Lifelines2 loads a dataset, it attempts to locate a corresponding `.cstm` file. If it exists, it is also loaded. Lifelines2 uses the class `lifelines.CustomizationReader` to read its preference data and populate the singleton `lifelines.Settings` instance for the dataset in the Lifelines2 session.

Most of the preference data are modifiable directly from Lifelines2 via the Preferences tab in the ARF tab. However, the optional customization variable (last line in Table A.3) which describes the semantic grouping of the event types in a

dataset is not editable from Lifelines2. Instead, users must edit the customization file directly. The customization variable describes how events are related to one another, and, subsequently should be displayed in Lifelines2. For example, users can use an Order customization to describe semantic relationship of the the following event types: High Blood Pressure, Normal Blood Pressure, and Low Blood Pressure. Order tells Lifelines2 that these three event types should be ordered that way vertically (High followed the Normal, followed Low) to reflect how users typically perceive the data instead using the default alphabetical ordering. Furthermore, the semantic constraint forces Lifelines2 to display these three events in a group (not interrupted by other events such as Hip Replacement, which would appear between High and Normal in alphabetical ordering). Multiple semantic groups can be created for a dataset, where no one event type should belong to multiple semantic customizations (such as multiple Orders).

The syntax for the semantic grouping in the `.cstm` is a 3-column format: Semantic Grouping Type (*e.g.*, "Order"), Name of the Semantic Group (*e.g.*, "Blood Pressure"), and a comma-separated list of event Types ("High blood Pressure, Low blood Pressure, Low Blood Pressure"). The columns are tab-separated. Figure A.1 shows part of the `.cstm` file, which includes the semantic grouping customizations, for the Contrast-Creatinine dataset presented in Chapter 5.2. Color highlighting was added to facilitate the description. Lines that start with % are comments and are ignored by the customization reader `lifelines.CustomizationReader`.

Most of the code for customization reside in the following packages: `lifelines.customization` and `lifelines.customization.drawers`. The package `lifelines.`

```
% event type semantics customization
Order   CREAT   CREAT-H, CREAT-, CREAT-L
Order   24H UCREAT      24H UCREAT-H, 24H UCREAT-, 24H UCREAT-L
Order   BL CREAT        BL CREAT-H, BL CREAT-, BL CREAT-1

% settings customization
View    Detailed        Shape   Triangle
View    Detailed        Size    18
View    Label   Font    11,Tahoma,0
View    Label   Position        8,4
View    Label   Visibility      true
View    Collapsed       Shape   Triangle
View    Collapsed       Size    6
View    Alignment       HalfWidth       13
View    Alignment       CalendarMark    false
File    deidentified data
Event   CREAT-H Color   255,0,0
Event   CREAT-  Color   102,153,255
Event   CREAT-L Color   102,204,0
```

Figure A.1: Partial screen shot of the customization file for the Contrast-Creatinine dataset in Section 5.2. Orange portion shows the semantic customization. Green portion shows the view-related customization. Red portion shows the descriptor for the type of input data. Blue portion shows the display preferences of each event type.

`customization` (332 lines of code) contains the customization file reader and the abstract super class `lifelines. customization.CustomRecord` for all customization grouping types such as `lifelines.customization.Order` (`Order`). The class (`Order`)is the only fully implemented grouping type. It allows users to constrain certain event types to be ordered in a certain way. Its two subclasses `lifelines. customization.CompactOrder` and `lifelines.customization.NumericOrder` are intended to, respectively, squeeze multiple event types into one single horizontal space (*e.g.*, put all High, Normal, and Low blood pressure events in the same horizontal space), and to display numerical data points (such as the actual reading of blood

pressure). These two subclasses are not fully implemented. However, the infrastructure is set up. Each Semantic grouping type such as `lifelines.customization.Order` is paired up with a drawer in the package `lifelines.customization.drawers`. When Lifelines2 draws events in the main display, it consults with the semantic grouping customization to see if a special drawer is to be used. Custom Drawers such as `CompactOrderDrawer` or `NumericOrderDrawer` in the same package would then draw these semantic grouping accordingly (*e.g.*, in a single horizontal space or displaying line graphs for numerical data points). Although the drawers are currently unimplemented, this infrastructure demonstrates the extensibility and adaptability to a wider range of data types and more customizable display of events.

A.1.2 Internal Data Structures

There are three types of data structures in Lifelines2. An immutable core data structure represents the records. A temporary, intermediate data structure represents the graphical data in the display. Finally, there are globally accessible data structures for a variety of usages, *e.g.*, customization settings and the internal representation of event types. I cover the first and the third types of data structures, and discuss the second type of data structures in Section A.3 together with drawing.

A.1.2.1 Core Data Structure

The main classes for data structures in Lifelines2 reside in the package `lifelines.data` (1118 lines of code). The basic unit of data in Lifelines2 is a `lifelines.data.Event`

(`Event`), which roughly corresponds to a line of input in the Lifelines2 format. An `Event` contains an event type, a time stamp, and an, optional, annotation.

A `lifelines.data.Record` (`Record`) contains many instances of `Event`. Chapter 4 describes how `Event`s of the same type are sorted by time stamp in an array, indexed by event type in a `Record` so that the Temporal Pattern Search (TPS) algorithm can work. However, arrays are not the only data structure that would work with TPS. Any sorted data structure that can support efficient binary search can be used. Although arrays were used in the earliest versions of Lifelines2, Lifelines2 currently uses the standard Java data structure `java.util.TreeMap` instead of arrays. The `java.util.TreeMap`s in Lifelines2 automatically sort `Event`s by timestamps, and supports efficient binary search just as sorted arrays do. Furthermore, the `java.util.TreeMap` data structure can allow the detection of duplicate events (same event type and time stamp). However, its performance is expected to be slightly worse than plain arrays because it is a more complex data structure, and requires more maintenance. All evaluations and results reported in Chapter 4 are performed with `Record`s that are implemented using `java.util.TreeMap`s. Chapter 4 uses the more common data structure *array* as an example to make the narrative clearer.

Finally, a `Record` keeps track of its graphical representations `Facet` in the Lifelines2 display. When multiple alignment is used, a `Record` can have multiple `Facet`s (Section A.3).

A `DataManager` contains a set of `Record`s and a variety of statistics. For example, it keeps track of a set of `Record`s and their `Facet`s, the number of `Record`s

in the `DataManager`, the minimum and the maximum time stamp, total number of events, and the current Align, Rank, Filter operators.

A.1.2.2 Global Data Structures

I already described the globally accessible `lifelines.Settings` class. Another data structure is the `lifelines.data.GlobalEventData` (`GlobalEventData`) class. `GlobalEventData` contains both the forward and the backward mapping of event data types to integers. When a new event type is read in from text, Lifelines2 creates an integer to represent it internally. This allows Lifelines2 to store only one copy of the event type `String`s (as opposed to the expensive one copy per `Event`), saving large amount of memory. Programmers can use `GlobalEventData` to access the event type `String` by the event type integer identifier, or use the identifier to access the `String` (as when the text needs to be displayed on screen).

A.1.3 Output Data

There are four kinds of outputs from Lifelines2: customization files, record files, screen shots and annotations, and logs I already presented the nuances of the customization files in Section A.1.1.2. Lifelines2 lets users to export current group of records into an external file (either via menu or via the group creation dialog box). The results of the export are in Lifelines2 format. If annotations are present, a 4-column format is used. Otherwise a 3-column format is used. When a group is exported, a `.cstm` file is automatically created to preserve the same

preferences as the original `.cstm` file. The exported file and its `.cstm` file reside in the `exportedGroups` directory by default.

Users can, at any time, take a screen shot of the current state of Lifelines2, and can optionally provide textual annotation with it. Screen shots are saved as `.jpg` files, and annotations are saved as `.txt` files. The screen shot and the annotation are saved in the `annotations` directory by default. The annotation code can be found in `lifelines.utils.ui.EurekaDialog`.

Lifelines2 keeps detailed logs of users' operations whenever it is launched. Logs are written and saved when data files are closed or if users normally close Lifelines2. The log files are saved in the `logs` directory in `.txt` files. These logs are purposefully written so that it conforms to the Lifelines2 format. This means that these log files can be directly fed into Lifelines2 for temporal analyses. The main code for logging reside in the `lifelines.replay` package (158 lines of code). The package is so named because the ultimate goal for logging is to allow replays of past analyses, given the logs and the input data. In the same package, the utility `lifelines.replay.LogAggregator` can be found to aggregate a large number of individual Lifelines2 log files into one for multi-session analysis using Lifelines2.

A.2 Align, Rank, and Filter

The classes that represent Align, Rank, and Filter reside in the `lifelines.arf` package (760 lines of code) . There are three main abstract classes: `AbstractAlignment`, `AbstractRank`, and `AbstractFilter` in the package. All Align, Rank, and Filter

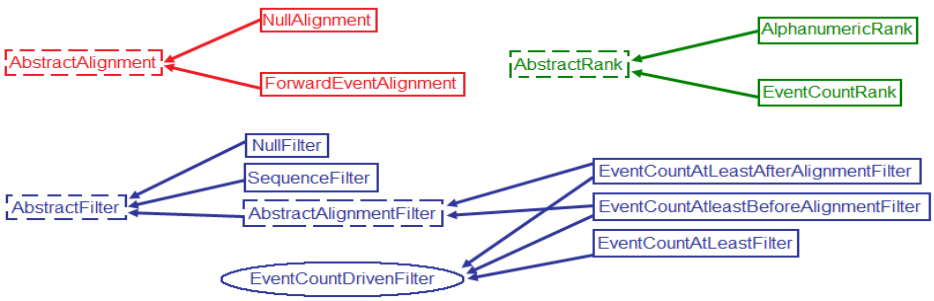

Figure A.2: Class inheritance diagram of the Align (red), Rank (green), and Filter (blue) operators. Dashed rectangles indicate abstract classes. Solid rectangles indicate concrete classes. Ovals indicate interfaces. Arrows indicates either an `extension` of a class or an `implementation` or an interface (depending on the context).

classes are, respectively, subclasses of these abstract classes. A detailed diagram of the class structures is presented in Figure A.2.

The `NullAlignment`, `AlphanumericRank`, and `NullFilter` are the default operators in Lifelines2. They correspond to align-by calendar time, rank-by record ID, and filter-by nothing. `ForwardEventAlignment` deals with relative alignment by the n^{th} event in the data, where n can be positive, negative (the n^{th} last event), or zero (all events). `EventCountRank` ranks records by the number of events of a selected event type. `SequenceFilter` filters records by event sequences, *i.e.*, using TPS. The three concrete classes that implement the interface of `EventCountDrivenFilter` allows the filtering by the number of events of a selected event type. The differences lie in the scope of filtering. `EventCountAtLeastFilter` looks at events at all times. The `EventCountAtLeastAfterAlignmentFilter` looks only at events after the alignment. Similarly, `EventCountAtLeastBeforeAlignmentFilter` looks only at events before the alignment. Users cannot perform the last two filers when no

alignment is active.

The inheritance design allows for modular implementation of the current Align, Rank, and Filter operators and is open to future operators using different criteria.

When users make a choice in Align, Rank, and Filter, Lifelines2 instantiates the newly selected operator and saves it in the `DataManager` that represents the set of `Record`s the users are current viewing. The `DataManager` then applies the ARF to its `Record`s. There are four results from the application of ARF operators. First, each `Record` is designated a set of time stamps. The set of time stamps indicate how big a shift needs to occur to align the particular `Record` with the others. When multiple alignment is activate, some `Record`s will have more than one shifts in the set. This set of time stamps are stored locally in the `DataManager`. Secondly, only `Record`s that can be aligned by the Alignment remain in the current set (maintained by the `DataManager`), the rest are "discarded". `DataManager` also maintains a original set, which can be used to undo the effects of Alignment. Thirdly, `Record`s that satisfy a particular Filter are highlighted. Fourthly, `Record`s are ordered by the current Rank operator. Code Listing A.1 shows the pseudocode that applies ARF in the `DataManager`. A, R, F in the code stand for the current Align, Rank, and Filter operators. myCurrentSet is the `DataManager`'s current set of Records. These four results are taken into account when rendering takes place.

```
0   ApplyARF(A, R, F)
1     for (each Record r in DataManager)
2        if (A.align(r))            //aligns
3          myCurrentSet.add(r)
4          if (F.isIncluded(r))     //filters
5        r.highlight()
6     R.sort(myCurrentSet)          //ranks
```

Code Listing A.1: Pseudocode for `DataManager.ApplyARF(...)`, where A, R, F stand for the `DataManager`'s current Align, Rank, and Filter operators.

A.3 Intermediate Data Structures and Rendering Behavior

Intermediate and temporary data structures for the graphical representation of data are created and destroyed Lifelines2's rendering. I describe the rendering details and present the design of these graphical elements in this section.

A.3.1 Drawing Records

There are many graphical elements that make up the main display of Lifelines2. They all reside in the `lifelines` package (7093 lines of code). `lifelines.Facet` (`Facet`) represents a `Record` visually. When alignment-by-all-instances is used, several `Facet`s may collectively represent a `Record`, where each `Facet` represents a shifted instance. Inside each `Facet`, there is a `lifelines.SubFacet` (`SubFacet`). The `SubFacet` is a legacy data structure from the original Lifelines. It had been used to represent the different facets of a patient record (*e.g.*, medication, complaints, treatments, etc.). In Lifelines2, however, the `SubFacet` structures are containers for graphical representations of `Event`s and the header information for each record.

SubFacets also drives the rendering of the events and headers. lifelines.EventNode (EventNode) class represent Events objects graphically. Each EventNode links to the actual Event object and creates a tooltip for mouseover events.

These graphical elements all subclass the edu.umd.cs.piccolo.PNode (PNode) class, and they are displayed onto a lifelines.TimeLine (TimeLine), which is a subclass of edu.umd.cs.piccolo.PCanvas. While Facets and SubFacets use default rendering from PNode, EventNode overrides the default. EventNodes render by first obtaining the current instance of the interface lifelines.drawers.shape .ShapeDrawer (ShapeDrawer) from the global instance of Settings. The concrete classes that implement ShapeDrawer has one method that draws the current shape with the current size for each EventNode as specified in Settings. There are four concrete classes that implement ShapeDrawer: DiamonDrawer, SquareDrawer, StripDrawer, and the default TriangleDrawer. These drawers can be found in the lifelines.drawers.shape package (111 lines of code).

After Align, Rank, and Filter are applied, DataManager creates a list of Facets and passes them to TimeLine for display. These Facets contain only the skeletal information. No SubFacet or EventNode objects are instantiated. Instead, these graphical elements are generated and added to the scene only when rendering is performed. Rendering takes place in the TimeLine.drawFacet method. Each Facet's bounds are set and its placement on the TimeLine is determined. If the Facet is in the current view port (as controlled by the scroll on the right), then its SubFacet and EventNodes are added for display. If a Facet is not in the view port, then its graphical elements are removed to reduce memory usage. That is, only elements

that can be seen by the users are kept in memory.

The early versions of Lifelines2 did create all graphical elements when new data is loaded. It would take a long time at the initial data load, and was not scalable to larger datasets because of its prohibitively high memory requirements. By dynamically creating and destroying the graphical elements of `Facet`s in the new implementation, Lifelines2 becomes more scalable. Having to create graphical elements on-the-fly is slower than having them ready all the time. However, this optimization enables Lifelines2 to handle larger datasets, such as the initial Heparin-Induced Thrombocytopenia (over 70000+ `EventNode`s when align-by-all instances of Heparin is used), that were not possible before.

Aside from drawing all visible `Facets` on screen, the `TimeLine` object also draws the background and the tick marks on the time line. The tick marks are managed by the class `lifelines.TimeRulerNode` (`TimeRulerNode`). It computes what the major tick marks are (*e.g.*, tick marks that represent year) and what the minor ticks are (*e.g.*, month) dynamically. It computes how many ticks are drawn, and where they are placed on screen. `TimeRulerNode` uses a herusitic to determine how visible tick marks should be (determined by alpha channel) by looking at how many tick marks can fit. If there are many, then they are less visible (more transparent). If there are many, then they are more visible. The aim is reduce to visual clutter that may detract from analyses. Unlike the `Facet`s and their subsidiary graphical elements, which are re-computed when different ARFs are applied, the `TimeRulderNode` needs to be re-computed whenever zooming and panning occurs. `TimeRulderNode` is implemented as a "sticky" in the `piccolo` pattern. It sits on

top of all the other visual elements in view.

A.3.2 Drawing Temporal Summaries

The code for drawing and managing temporal summaries are in the package `lifelines.summary.temporal` (2452 lines of code). For the ease of narration, I will refer to the classes only by their class name below, although readers should take the package name into account. Temporal summaries provide interesting design challenges. The summaries must dynamically deal with multiple event types. The histogram bars need to be linked with the actual `Event`s, `Record`s, and `Facet`s they represent. Temporal summaries need to deal with multiple temporal granularities. Finally, aggregating all `Event`s across all `Record`s can be an expensive operation. The general idea temporal summaries work is by first creating an array of size $n+1$, where n corresponds to the number of ticks in the time scale in the main Lifelines2 display. Each element in the array represents a histogram bar in the summary, keeps the information about how many `Event`s, `Facet`s, and `Record`s contribute to the bar. The histogram bar class `SummaryBar` handles multiple events, and stacks up bars to represent the aggregation of multiple events.

As users zoom in and zoom out, the temporal granularity may change. When the temporal granularity changes, temporal summaries recompute the aggregation. The recomputation is performed in a need-only manner. If users pan the time line. All previously out-of-view histogram bars are created dynamically, and the appropriate events aggregated. These bins are cached, so once they are created,

they do not need to be re-created again, unless new Align or Filter are applied, new dataset is opened, or different events are to be aggregated. Similarly, when temporal granularity changes due to zoom-in or zoom-out, additional bins can either be created (zooming in from year granularity to month granularity makes 12 new bins for each year) or collapsed (zooming out from month granularity to year would combine 12 month bins into 1 year bin), temporal summaries need to re-aggregate and recompute the bins.

The class `TemporalSummaryModel` is the top level data structure that represents a temporal summary. It has a set of `MultiGranularityTemporalSummaryModels`, which each has several `SingleTemporalSummaryModels`. Each `SingleTemporalSummaryModel` represents a temporal summary at a different temporal granularity. As temporal granularity changes due to user interactions, temporal summaries dynamically chooses the aggregation of the right granularity to display. A `SingleTemporalSummaryModel` keeps the aggregation of `Events`, `Facets`, and `Records` for the same granularity. Each aggregation is indexed by the time stamp of the time line ticks.

The abstract class `AbstractTemporalSummary` represents the root class for all temporal summary classes. It contains a `TemporalSummaryModel` and provides methods for basic accessor and mutator methods to the model structure. There are five subclassses of `AbstractTemporalSummary`: `AfterOnlyTemporalSummary`, `BeforeOnlyTemporalSummary`, `BothSidesTemporalSummary`, `DistributionTemporalSummary`, and `NoSidesTemporalSummary`. They represent each the four types summaries shown when "Distribution by Occurrence Relative to Alignment" tab is

used and one generic temporal summary (`DistriutionTemporalSummary`). These subclasses allow slightly different visual representation and labeling.

A `TemporalSummaryManager` manages the temporal summaries in Lifelines2. It performs the aggregation and decides what portion of the aggregation goes to which temporal summary. For example, if the "Distribution by Occurrence Relative to Alignment" tab is clicked, `TemporalSummaryManager` determines which `Facet`s belong to "Occurrence Only Before Alignment", and aggregates their `Event`s for the `BeforeOnlyTemporalSummary`. `TemporalSummaryManager` does the same for other three other occurrence types. When "Distribution by Occurrence Relative to Alignment" is not active, `TemporalSummaryManager` aggregates all data into the `DistriutionTemporalSummary`. Each temporal summary is bundled with a `SummarySelector` to perform direct selection. When aggregation has been done, all temporal summaries are placed onto a `TemporalSummaryCanvas`.

An analogous set up for the temporal summaries for the purpose of group comparison can be found in the `lifelines.comparison` and `liefelines.comparison.summary` packages (784 lines of code altogether). Comparisons use `NullSplitComparisonTemporalSummary`, a subclass of `DistributionTemporalSummary` to allow slightly different visual displays.

A.4 GUI Layout

The main GUI components are in the `LifelinesPanel` in the `lifelines` package (7093 lines of code). The `LifelinesPanel` was created in place of a

javax.swing.JFrame so that Lifelines2 can be more easily used as a component to be integrated in other outside tools such as the i2b2 [70] platform. The current Lifelines2 creates an LFrame as a container to the LifelinesPanel, but LFrame has limited functions. Virtually every interaction users make will be directly done to LifelinesPanel.

A LifelinesPanel has two main areas. The first is the main display area to the left, and a control panel to the right. The areas are split via a javax.swing.SplitPane. On each side of the SplitPane, there is a javax.swing.JTabbedPane. The two JTabbedPanes are synchronized so that when users select the first tab on the left side, the tab on the right side also becomes active and vice-versa. The left JTabbedPane is in a lifelines.VisualizationPanel, which manages the main display area, while the control panel area is directly managed by the LifelinesPanel.

In the VisualizationPanel, There are two main views. The first is the "Record View", and the second view is the "Comparison view". Depending on which view is active, the VisualizationPanel displays different content. When "Record View" is active, the typical display of records, events, and temporal summaries are displayed. When "Comparison View" is active, comparisons of groups using temporal summaries are displayed.

In the "Record View", there are two additional areas. The top part shows the records in the time line, while the bottom portion shows the temporal summaries, if they are active. The two portions are synchronized temporally via a double range slider at the bottom or the zooming controls in the top view. The VisualizationPanel has many roles. It first acts as a platform where chosen data

can be displayed. Groups of records can be loaded in and out of the `VisualizationPanel`. The `VisualizationPanel` interfaces between the data and the visual and the graphical elements. The `VisualizationPanel` also manages the `TimeLine` object, which manages the record visualization on the top, and the `TemporalSummaryManager` object, which manages the display of temporal summaries at the bottom. Finally, `VisualizationPanel` passes along user interactions received from `LifelinesPanel` to the data and visualization. Similarly, when the "Comparison View" is active, `VisualizationPanel` manages the visualization and interaction components of the comparisons.

A.5 ARF Time vs. Rendering Time

The rendering optimization techniques described in Section A.3.1 led to a more detailed examination to the overall performance of Lifelines2 – that is, does performing ARF operators on the dataset take more time the rendering? Which side should receive more attention for further optimziation? On the one hand, since ARF operators are intentionally invoked and that they apply to the entire dataset, users may be more patient with it. So optimization for ARF operators may not be as critical (though they should not take excessively long). On the other hand, rendering is invoked implicitly whenever scrolling, zooming, panning, and application of ARF are invoked. In normal circumstances, rendering should be kept under 160 milliseconds to deliver a fluid interaction experience.

I used the five real datasets presented in Chapter 4 to perform the performance

analysis. I instrumented Lifelines2 so that it records the time it takes to perform a ARF operation and to perform a rendering operation. I loaded each dataset into Lifelines2 and perform a variety of align, rank, and filter operations. At the end of operation, the timing for ARF operation and rendering operations were reported. Each dataset had a set of 10 comparable ARF operations: align by events, align to calendar time, 4 sequence filters, rank by most prevalent event (so the screen is populated by the most number of events), one filter by most prevalent event, rank by record IDs, and filter reset. The sequence filters are chosen from the set of sequence filters physicians would use in their investigation. Because the Heart Attack case study required no sequence filters and had no appropriate sequences for this analysis, comparable filter sequences were created and used in this comparison.

The timing results and a linear fitting of the data are presented in Figure A.3. For the smallest data set (Heparin-Induced Thrombocytopenia (814 records)), the rendering time is more than five times the ARF time. As the number of records increase, both ARF time and rendering time increase. However, the time it takes to perform ARF grows much faster than rendering time. In the largest dataset, where there are 50000+ records, ARF time is about twice that of rendering time. The rendering algorithm, as described in Section A.3.1, dynamically generates detailed visual elements for a `Facet` only if it is on screen. This means that if the number of events in the screen are the roughly the same across the datasets, the rendering performance graph to be roughly constant. However, the rendering algorithm also adds all `Facet`s Facet to the canvas, regardless if they would be on or off the screen. This explains the growth of rendering as the number of records increases.

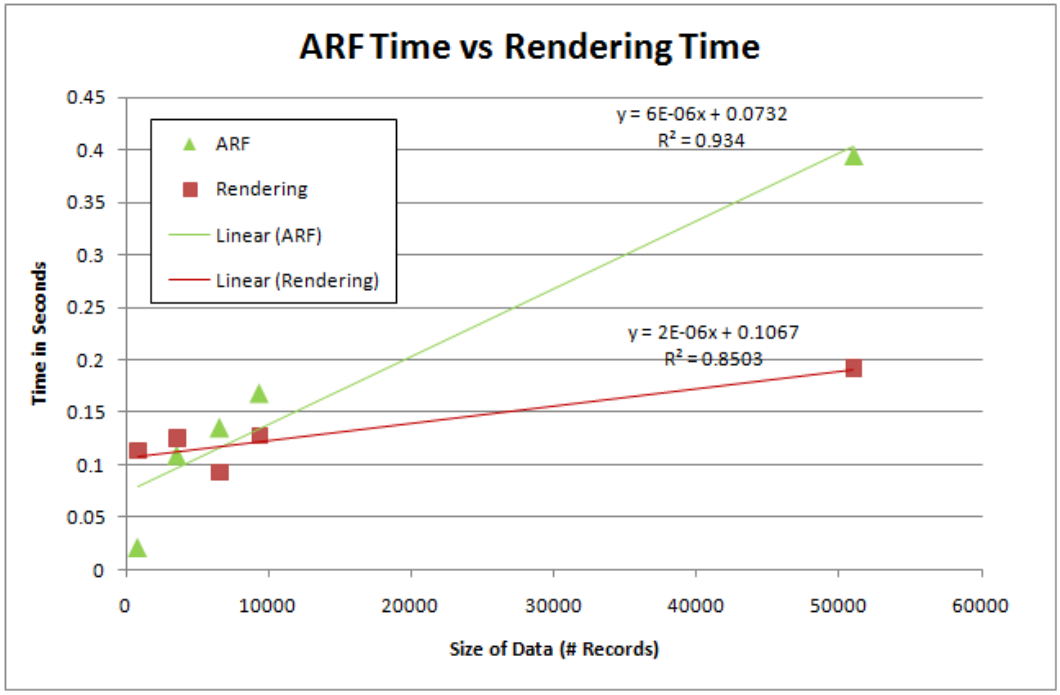

Figure A.3: Average execution time for ARF operators and rendering across five datasets and their linear regression.

The time it takes to perform ARF grows much faster than rendering because, in contrast to rendering, ARF is applied to every single record. Performance improves if filtering or alignment operations can be terminated early without having to scan an entire record, but every record must be scanned. As a result, the average standard deviation for ARF operations (0.0964) is more than twice that for the rendering (0.0411). Figure A.4 shows the individual timing results. The difference in standard deviation is apparent.

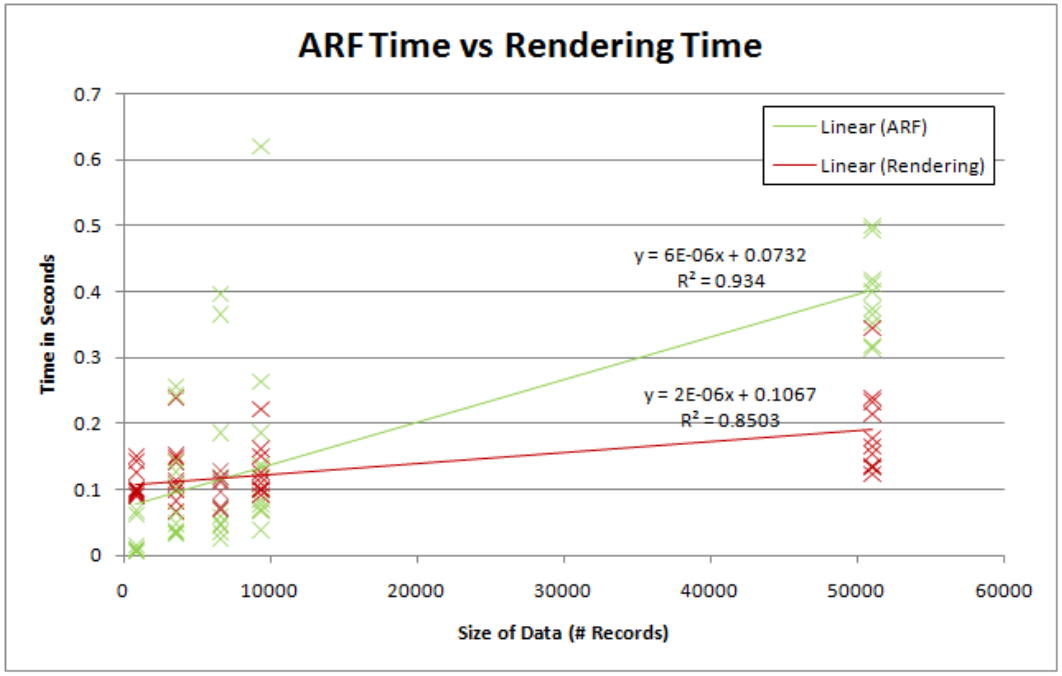

Figure A.4: Execution time details for ARF operators and rendering across five datasets and their linear regression. The difference in standard deviation between ARF timing and Rendering timing is apparent.

A.5.1 Summary

I present the engineering and the organization of the Lifelines2 software. I discuss the data model and the rendering pipeline in Lifelines2. I compare the rendering time and ARF time as they are now, and show that ARF time grows much faster than rendering time.

Appendix B

Temporal Pattern Query in Practice

Temporal pattern queries are important in medicine. However, most state-of-the-art medical informatics systems do not allow such queries. Favoring simplicity, learnability, and consistency with existing interfaces, these systems often only offer Boolean queries on the existence of events. In order to issue temporal pattern queries, domain experts typically need to circumvent the interface altogether and construct command line queries themselves. In this section, I present an example on how such a query can be constructed and what difficulties domain experts face in real situations. I present how generalizable the approach is, and discuss why an affordable interface is important.

B.1 An Example of Temporal Pattern Query

I use the commonly accessible SQL database mySQL as an illustration on what a domain experts might have learn in order to issue a temporal pattern query on a simple 3-column table (Table B.1) similar to the Lifelines2 format (Section A.1.1.1). This is only an example because different SQL database vendors have slightly different syntaxes and constructs that may allow for a simpler formulation of the query. However, the thought process in constructing such a query would be similar.

name	**type**	**date**
Alice	Proposal	2002-07-08
Alice	Defense	2004-06-24
Alice	Paper Submission	2005-12-22
Bob	Paper Submission	2002-04-18
Bob	Proposal	2002-11-07
Bob	Paper Submission	2004-01-07
Bob	Defense	2006-11-07
Bob	Paper Submission	2007-02-14
Charlie	Paper Submission	2003-02-07
Charlie	Paper Submission	2004-01-21
Charlie	Proposal	2004-05-30
...	...	...

Table B.1: The "student" SQL table.

B.2 Example Query

Imagine a domain expert is analyzing graduate student records using the data shown in Table B.1. She is interested in the rare situation where students proposed for a dissertation topic and defended, but did not submit any papers to conferences or journals during that period of time.

This pattern involves the "absence of" operator, complicating the strategy to construct the SQL. She needs to first find the set of all students who have the pattern [*Proposal*, *Defense*] (set 1). Next, she needs to construct a second set of students who have the pattern [*Proposal*, *Paper Submission*, *Defense*] (set 2). Finally, she needs to take a set difference, subtracting set 2 from set 1 (Figure B.1).

It is not a surprise that she needs to learn the syntaxes for the common `SELECT`, `FROM`, `WHERE`, `JOIN` keywords or the comparison operators `AND`, `<`, `>`. But she may be surprised to find that the set difference operation (the blue portion) is not directly supported in mySQL. mySQL (version 5.x, the latest free version when the example

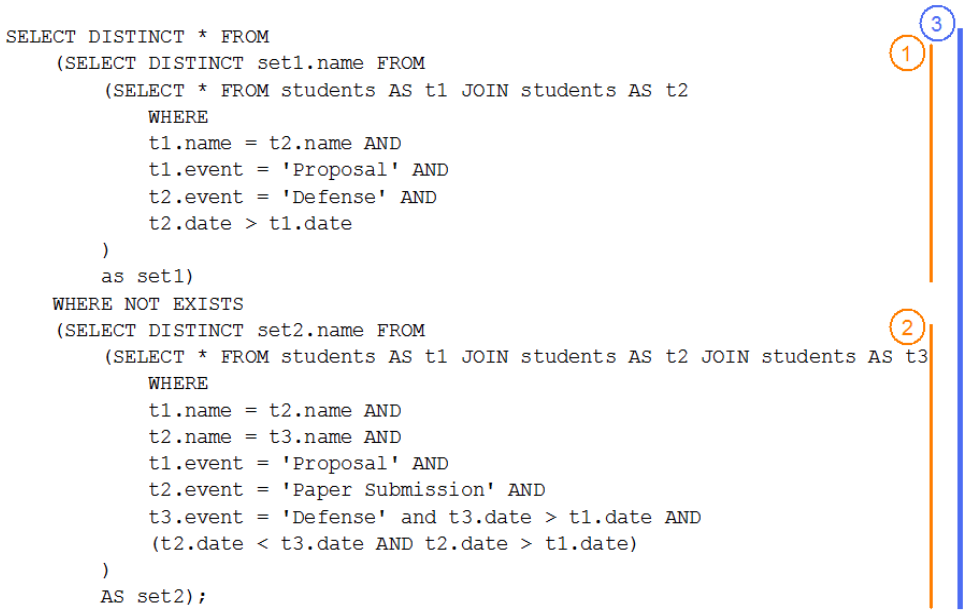

Figure B.1: SQL query in mySQL to find students who have the pattern [*Proposal*, *No Paper Submission*, *Defense*]. The orange portions show the construction of set 1 and set 2. Then set 2 is subtracted from set 1 (blue portion). Keywords in the mySQL syntax are capitalized.

was written) does not support set intersection or set difference operators (where as in standard SQL, `INTERSECT` and `EXCEPT` are supported operators). Instead, she has to use the combination of `DISTINCT` and `NOT EXIST` cleverly in a `WHERE` clause to get the ideal effect.

B.3 Analysis of the Query

The above example is a 3-part query that has an absence event in the middle. This means that the student table needs to be `JOIN`ed twice with itself in order to examine the pattern of three events. The number of `JOIN` operations increases linearly with the number of parts in the query. With an n-part query, number of

`JOIN` operations would be $n-1$. `JOIN` operations make a cross product of the tables in the operation, effectively making a table of size k to size k^2 if `JOIN`ing a table with itself. In this example query, a table of size k^3 is created.

Some database management systems offer a `JOIN` optimization for the common case where `JOIN` occurs only on the same ID. The hash join operation joins only the rows that share the same ID. In this example, only the rows that begin with "Alice" are joined, and the rows that begin with "Bob" are joined (separately), and so on, before putting all the joined rows together. This optimization prevents the k^n growth of tables, where k is the number of rows, and n the number of parts in the pattern query. Hash join is very effective when there are only a few events per student (say, 20 or fewer[1]), as in this example. However, when there are scores or hundreds of events per student (as in the case of patients in electronic health records), the time it takes to execute can be prohibitively long.

The query construction can become more tedious in more complex situations. In the 3-part example above, there is only one positive event pair (`Proposal`, `Defense`) that are relevant to the negative event. If there are multiple pairs, the query needs to identify each pair and force potentially different constraints. For example, the template for the above example would not work for the 6-part query [*A*, *No B*, *C*, *A No D*, *C*] because there are multiple pairs of `A` and `C`. The query will need to include additional constructs to differentiate which `A-C` pair cannot have `B` in between, and which `A-C` pair cannot have `D` in between.

[1]Source: private communication with professor Amol Deshpande

B.4 Additional Barriers

In addition to wrestling with issues in constructing SQL query, domain experts also struggle to deal with infrastructural difficulties. For example, in many institutions (hospitals, law enforcement institutions), data are not all stored in the same database. They are often siloed in many databases, and it relies on each user to decide which databases to connect to. Domain experts often do not know the which pieces of data reside in which databases, what each piece of information is named, and what each database is named.

B.5 Summary

The technical difficulties and the infrastructural difficulties prevent domain experts accessing the data they desire. The work presented in this dissertation, admittedly, does not help alleviate the infrastructural issues. However, the consistent GUI to allow domain analysts to specify temporal pattern queries reduces the amount of work they need to do to construct complex SQL queries, and in turn avoid the inconsistent approaches they must take to in different situations.

Bibliography

[1] Marc Abrams, Naganand Doraswamy, and Anup Mathur. Chitra: visual analysis of parallel and distributed programs in the time, event, and frequency domains. *IEEE Transactions on Parallel and Distributed Systems*, 3:672–685, 1992.

[2] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil immerman. Efficient pattern matching over event streams. *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 147–160, 2008.

[3] Rakesh Agrawal, Giuseppe Psaila, Edward L. Wimmers, and Mohamed Zait". Querying shapes of histories. *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB 95)*, pages 502–514, 1995.

[4] Christopher Ahlberg, C. Williamson, and Ben Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92)*, pages 619–626, 1992.

[5] Diane Lindwarm Alonso, Anne Rose, Catherine Plaisant, and Kent L. Norman. Viewing personal history records: A comparison of tabular format and graphical presentation using lifelines. *Behaviour and Information Technology*, 17(5):249–262, 1998.

[6] Paul André, Max L. Wilson, Alistair Russell, Daniel A. Smith, Alisdair Owens, and m.c. schraefel. Continuum: designing timelines for hierarchies, relationships and scale. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*, pages 101–110, New York, NY, USA, 2007. ACM.

[7] Aleks Aris and Ben Shneiderman. Designing semantic substrates for visual network exploration. *Information Visualization*, 6(4):281–300, 2007.

[8] Alex Aris, Ben Shneiderman, Catherine Plaisant, Galit Shmueli, and Wolfgang Jank. Representing unevenly-spaced time series data for visualization and interactive exploration. *Proceedings of the International Conference on Human-Computer Interaction (INTERACT '05)*, pages 835–846, 2005.

[9] Ragnar Bade, Stefan Schelchtweg, and Silvia Miksch. Connecting time-oriented data and information to a coherent interactive visualization. *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 105–112, 2004.

[10] Benjamin B. Bederson, J. Grosjean, and J. Meyer. Toolkit design for interactive structured graphics. *IEEE Transactions on Software Engineering*, 30(8):535–546, 2004.

[11] Robert S. Boyer and J. Strother Moore. A fast string-searching algorithm. *Communications of the ACM*, 20:762–772, 1977.

[12] J. Carlis and J. Konstan. Interactive visualization of serial periodic data. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 29–38, New York, NY, USA, 1998. ACM.

[13] Rong Chen, Anne Rose, and Benjamin B. Bederson. How people read books online: Mining and visualizing web logs for use information. In *Research and Advanced Technology for Digital Libraries*, pages 364–369, 2009.

[14] Luca Chittaro, Carlos Combi, and Giampaolo Trapasso. Data minding on temporal data: a visual approach and its clinical application to hemodialysis. *Journal of visual Languages and Computing*, 14:591–620, 2003.

[15] W. Chung, H. Chen, L.G. Chaboya, C. O'Toole, and H. Atabakhsh. Evaluating event visualization: A usability study of coplink spatio-temporal visualizer. *International Journal of Human-Computer Studies*, 62(1):127–157, 2005.

[16] Carlo Combi, Angelo Montanari, and Giuseppe Pozzi. The t4sql temporal query language. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management (CIKM '07:)*, pages 193–202, New York, NY, USA, 2007. ACM.

[17] Steve B. Cousins and Michael G. Kahn. Visualizing operations on temporal data. In *Proceedings of the First Conference on Visualization in Biomedical Computing*, pages 72–76, 1990.

[18] Steve B. Cousins and Michael G. Kahn. The visual display of temporal information. *Artificial Intelligence in Medicine*, 3(3):347–351, 1991.

[19] Russ Cox. Regular expression matching can be simple and fast. `http://swtch.com/~rsc/regexp/regexp1.html`, 2007.

[20] DataMontage. http://www.stottlerhenke.com/datamontage/.

[21] Marc Davis. Media streams: An iconic visual language for video representation. *Readings in Human-Computer Interaction: Toward the Year 2000*, pages 854–866, 1995.

[22] Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker White. Towards expressive publish/subscribe systems. *Proceedings of the 10th International Conference on Extending Database Technology*, pages 627–644, 2006.

[23] Micah Dubinko, Ravi Kumar, Joseph Magnani, Jasmins Novak, Prabhakar Raghavan, and Andrew Tomkins. Visualizing tags over time. In *Proceedings of the 15th international conference on World Wide Web (WWW 06)*, pages 193–202, New York, NY, USA, 2006. ACM.

[24] Jerry Fails, Amy Karlson, Layla Shahamat, and Ben Shneiderman. A visual interface for multivariate temporal data: Finding patterns of events over time. *Proceedings of the IEEE Symposium of Visual Analytics Science and Technology (VAST 2006)*, 2006.

[25] Anthony Faiola and Simon Hillier. Multivariate relational visualization of complex clinical datasets in a critical care setting: A data visualization interactive prototype. In *IV '06: Proceedings of the conference on Information Visualization*, pages 460–468, Washington, DC, USA, 2006. IEEE Computer Society.

[26] Domenico Ficara, Stefano Giodano, Gregorio Procissi, Fabio Vitucci, Gianni Antichi, and Andrea Di Pietro. An improved dfa for fast regular expression matching. *ACM SIGCOMM Computer Communication Review*, 38(5):29–40, 2008.

[27] Brian Francis and Mark Fuller. Visualization of event histories. *Journal of the Royal Statistical Society Series A*, 159(2):301–308, 1996.

[28] Anna Fredrikson, Chris North, Catherine Plaisant, and Ben Shneiderman. Temporal, geographical and categorical aggregations viewed through coordinated displays: a case study with highway incident data. In *Proceedings of the 1999 Workshop on New Paradigms in Information Visualization and Manipulation in Conjunction with the eighth ACM International Conference on Information and Knowledge Management (NPIVM '99)*, pages 26–34, New York, NY, USA, 1999. ACM.

[29] François Guimbretiére, Morgan Dixon, and Ken Hinckley. Experiscope: An analysis tool for interaction data. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*, pages 1333–1342, 2004.

[30] Lilian Harada and Yuuji Hotta. Order checking in a CPOE using event analyzer. *Proceedings of the Fourteenth ACM Conference on Conference on Information and Knowledge Management (CIKM '05:)*, pages 549–555, 2005.

[31] Lilian Harada, Yuuji Hotta, and Tadashi Ohmori. Detection of sequential patterns of events for supporting business intelligence solutions. *Proceedings of the International Database Engineering and Applications Symposium (IDEAS 04)*, pages 475–479, 2004.

[32] Beverly Harrison, Russell Own, and Ronald Baecker. Timelines: An interactive system for the collection and visualization of temporal data. *Proceedings of Graphics Interface '94*, pages 141–148, 1994.

[33] Paul K. Harter, Dennis Heimbigner, and Roger King. Idd: An interactive distributed debugger. Technical Report CU-CS-274-84, Department of Computer Science, University of Colorado at Boulder, 1984.

[34] Susan Havre, Elizabeth Hetzler, Paul Whitney, and Lucy Nowell. Themeriver: Visualizing thematic changes in large doeumcnet collections. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):9–20, 2002.

[35] Michael T. Heath and Jennifer A. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, 8:29–39, 1991.

[36] Jeffrey Heer, Fernanda B. Viégas, and Martin Wattenberg. Voyagers and voyeurs: supporting asynchronous collaborative information visualization. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1029–1038, New York, NY, USA, 2007. ACM.

[37] K.P. Hewagamage, M. Hirakawa, and T. Ichikawa. Interactive visualization of spatiotemporal patterns using spirals on a geographical map. In *Proceedings of Visual Languages 1999*, pages 296–303, 1999.

[38] Stacie Hibino and Elke A. Rundensteiner. Interactive visualization for temporal analysis: application to cscw multimedia data. *Intelligent Multimedia Information Retrieval*, pages 313–335, 1997.

[39] Stacie Hibino and Elke A. Rundensteiner. User interface evaluation of a direct manipulation temporal visual query language. In *MULTIMEDIA '97: Proceedings of the fifth ACM international conference on Multimedia*, pages 99–107, New York, NY, USA, 1997. ACM.

[40] K. Hinum, S. Miksch, W. Aigner, S. Ohmann, C. Popow, M. Pohl, and M. Rester. Gravi++: Interactive information visualization to explore highly structured temporal data. *Journal of Universal Computer Science (J. UCS) – Special Issue on Visual Data Mining*, 11(11):1792–1805, 2005.

[41] Harry Hochheiser and Ben Shneiderman. Interactive exploration of time series data. In *DS '01: Proceedings of the 4th International Conference on Discovery Science*, pages 441–446, London, UK, 2001. Springer-Verlag.

[42] W. Horn, C. Popow, and L Unterasinger. Support for fast comprehension of icu data: Visualization using metaphor graphics. *Methods of Information in Medecine*, 40(5):421–424, 2001.

[43] Weidong Huang, Peter Eades, and Seok-Hee Hong. Beyond time and error: a cognitive approach to the evaluation of graph drawings. In *BELIV '08: Proceedings of the 2008 conference on Beyond time and errors*, pages 1–8, New York, NY, USA, 2008. ACM.

[44] I. Janszky and R. Ljung. Shifts to and from daylight saving time and incidence of myocardial infarction. *New England Journal of Medicine*, 359(18):1966–1968, 2008.

[45] Matt Jensen. Visualizing complex semantic time-lines. Technical Report BMTR2003-001, NewsBlip, 2003.

[46] N. K. Jog and B. Shneiderman. Starfield visualization with interactive smooth zooming. In *Proceedings of the third IFIP WG2.6 Working Conference on Visual Database Systems 3 (VDB-3)*, pages 3–14, London, UK, UK, 1995. Chapman & Hall, Ltd.

[47] Ian Johnson. Putting time on the map. *GeoInformatics*, 2004.

[48] Hyunmo Kang, Vivek Sehgal, and Lise Getoor. Geoddupe: A novel interface for interactive entity resolution in geospatial data. In *IV '07: Proceedings of the 11th International Conference Information Visualization*, pages 489–496, Washington, DC, USA, 2007. IEEE Computer Society.

[49] Harry C. Karadimas, Christophe Chailloleau, François Hemery, Julien Simonnet, and Eric Lepage. Arden/j: An architecture for MLM execution on the java platform. *Journal of American Medical Informatics Association*, 9(4):359–368, 2002.

[50] Gerald Karam. Visualization using timelines. In *Proceedings of the 1994 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '94)*, pages 125–137, New York, NY, USA, 1994. ACM.

[51] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. Technical Report TR-31-81, Aiken Computation Laboratory, Harvard University, 1981.

[52] Denis Klimov, Yuval Shahar, and Meirav Taieb-Maimon. Intelligent selection and retrieval of multiple time-oriented records. *Journal of Intelligent Information Systems (Published Online)*, 2009.

[53] Donald E. Knuth, James H. Moris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[54] Aparna Krishnan and Steve Jones. Timespace: activity-based temporal visualisation of personal information spaces. *Personal Ubiquitous Comput.*, 9(1):46–65, 2005.

[55] Sailesh Kumar, Balakrishnan Chandrasekaran, Jonathan Turner, and George Varghese. Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia. *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, pages 155–164, 2007.

[56] V. Kumar and R. Furuta. Metadata visualization for digital libraries: Interactive timeline editing and review. In *Proceedings of the 3rd ACM Conference on Digital Libraries*, pages 126–133, 1998.

[57] Heidi Lam, Daniel Russell, Diane Tang, and Tamara Munzner. Session viewer: Visual exploratory analysis of web session logs. *Symposium On Visual Analytics Science And Technology*, 0:147–154, 2007.

[58] Stanley Lam. Patternfinder in microsoft amalga: Temporal query formulation and result visualization in action. `http://www.cs.umd.edu/hcil/patternFinderInAmalga/PatternFinderS-HonorsPaper.pdf`, 2008.

[59] Bonshin Lee, Cynthia C. Parr, Catherine Plaisant, Benjamin B. Bederson, V.D. Veksler, W.D. Gray, and C. Kotfila. Treeplus: Interactive exploration of networks with enhanced tree layouts. *IEEE TVCG Special Issue on Visual Analytics*, 12(6):1414–1426, 2006.

[60] Ted Lehr, Zary Segall, Dalibor F. Vrsalovic, Eddie Caplan, Alan L. Chung, and Charles E. Fineman. Visualizing performance debugging. *Computer*, 22(10):38–51, 1989.

[61] Wilhelm Hector Richard Albrecht Lexis. *Einleitung in die Theorie der Bevolkerungsstatistik*. K.J. Trübner, 1875.

[62] Jessica Lin, Eamonn Keogh, Stefano Lonardi, Jeffrey Lankford, and Donna Nystrom. Visually mining and monitoring massive time series. In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '04)*, pages 460–469, New York, NY, USA, 2004. ACM.

[63] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The perspective wall: detail and context smoothly integrated. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*, pages 173–176, New York, NY, USA, 1991. ACM.

[64] H. Mannila and P. Moen. Finding similar situations in sequences of events via random projections. *Data Warehousing and Knowledge Discovery (Dawak '99)*, pages 271–280, 1999.

[65] H. Mannila and P. Ronkainen. Similarity of event sequences. *Proceedings of Temporal Representation and Reasoning (TIME '97)*, pages 136–139, 1997.

[66] Charles E. McDowell and David P. Helmbold. Debugging concurrent programs. *ACM Computing Survey*, 21(4):593–622, 1989.

[67] Peter McLachlan, Tamara Munzner, Eleftherios Koutsofios, and Stephen North. Liverac: interactive visual exploration of system management time-series data. In *Proceeding of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*, pages 1483–1492, New York, NY, USA, 2008. ACM.

[68] Microsoft. http://www.microsoft.com/amalga/.

[69] Anders Møller. http://www.brics.dk/automaton/.

[70] Shawn Murphy, M. Mendis, K. Hackett, R. Kuttan, W. Pan, L. Phillips, V. Gainer, D. Berkowicz, J. Glaser, I. Kohane, and H. Chueh. Architecture of the open-source clinical research chart from informatics for integrating biology and the bedside. In *Proceedings of the American Medical Informatics Association Annual Symposium (AMIA '07)*, pages 548–552, 2007.

[71] G. Navarro. Pattern matching. *Journal of Applied Statistics*, 31(8):925–949, 2004.

[72] G. Navarro and M. Raffinot. Fast and flexible string matching by combining bit-parallelism and suffix automata. *ACM Journal of Experimental Algorithmics (JEA)*, 5(4):Article No. 4 (36 pages), 2000.

[73] G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings*. Cambridge Unversity Press, 2002.

[74] Chris North. Toward measuring visualization insight. *IEEE Comput. Graph. Appl.*, 26(3):6–9, 2006.

[75] Martin J O'Connor, Samson W Tu, and Mark A Musen. The chronus ii temporal database mediator. *Proceedings of AMIA Symposium*, pages 657–571, 2002.

[76] J.M. Overhage, B. Mamlin, J. Warvel, J. Warvel, W.M. Tierney, and C.J. McDonald. A tool for provider interaction during patient care: G-care. *Proceedings of Annuual Symposium on Computer Applications in Medidcal Care*, pages 178–182, 1995.

[77] Gultekin Ozsoyoglu and Huaqing Wang. A relational calculus with set operators, its safety, and equivalent graphical languages. *IEEE Transactions on Softwware Engineering*, 15(9):1038–1052, 1989.

[78] Adam Perer and Ben Shneiderman. Balancing systematic and flexible exploration of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):693–700, 2006.

[79] Adam Perer and Ben Shneiderman. Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis. In *Proceeding of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*, pages 265–274, New York, NY, USA, 2008. ACM.

[80] D. Phan, J. Gerth, M. Lee, A Paepcke, and T. Winograd. Visual analysis of network flow data with timelines and event plots. In *Proceedings of the workshop on Visualization for Computer Security '08*, pages 85–99, 2008.

[81] David S. Pieczkiewicz, Stanley M. Finkelstein, and Marshall I. Hertz. Design and evaluation of a web-based interactive visualization system for lung transplant home monitoring data. *Proceedings of the American Medical Informatics Association Annual Symposium (AMIA '07)*, pages 598–602, 2007.

[82] C. Plaisant, S. Lam, B. Shneiderman, M. Smith, D. Roseman, G. Marchand, M Gillam, C Feied, J Handler, and H. Rappaport. Searching electronic health records for temporal patterns in patient histories: A case study with microsoft amalga. *Proceeding of the American Medical Informatics Association Annual Symposium (AMIA '08)*, pages 601–605, 2008.

[83] Catherine Plaisant, Brett Milash, Anne Rose, Seth Widoff, and Ben Shneiderman. Lifelines: Visualizing personal histories. *Proceedings of CHI 96*, 1996.

[84] Catherine Plaisant, R. Mushlin, A. Snyder, J. Li, D. Heller, and Ben Shneiderman. Lifelines: Using visualization to enhance navigation and analysis of patient recordss. *Proceedings of American Medical Informatics Association Fall Symposium (AMIA '98)*, pages 76–80, 1998.

[85] Andrew R. Post and James H. Harrison. Protempa: A method for specifying and identifying temporal sequences in retrospective data for patient selection. *JAMIA*, 2007.

[86] S. Powsner and E. Tufte. Graphical summary of patient status. *The Lancet*, 344:386–389, 1994.

[87] Rolant Pressat. Lánalyse démographique: Méthodes, résultats, applications. *Population*, 16:505–508, 1961.

[88] SIMILE Project. http://simile.mit.edu/timeline.

[89] M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 2:114–125, 1959.

[90] Jun Rekimoto. Time-machine computing: a time-centric approach for the information environment. In *Proceedings of the 12th annual ACM symposium on User interface software and technology (UIST '99)*, pages 45–54, New York, NY, USA, 1999. ACM.

[91] Randy Ribler, Randy, Anup Mathur, and Marc Abrams. Visualizing and modeling categorical time series data. Technical report, Virginia Polytechnic Institute && State University, Blacksburg, VA, USA, 1995.

[92] Peza Sadri, Carlo Zaniolo, Amir Zarkesh, and Jafar Adibi. Expressing and optimizing sequence queries in database systems. *ACM Trans. on Database Systems*, 29(2):282–318, June 2004.

[93] Purvi Saraiya, Chris North, Vy Lam, and Karen A. Duca. An insight-based longitudinal study of visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 12:1511–1522, 2006.

[94] Jinwook Seo and ben Shneiderman. Knowledge discovery in high dimensional data: Case studies and a user survey for an information visualization tool. In *IEEE Transactions on Visualization and Computer Graphics*, volume 12, pages 311–322, 2006.

[95] Asaf Shabtai, Denis Klimov, Yuval Shahar, and Yuval Elovici. An intelligent, interactive tool for exploration and visualization of time-oriented security data. In *Proceedings of the 3rd International Workshop on Visualization for Computer Security (VizSEC '06))*, pages 15–22, New York, NY, USA, 2006. ACM.

[96] Yuval Shahar and Cleve Cheng. Intelligent visualization and exploration of time-oriented clinical data. *Proceedings of the 32nd Hawaii International Conference on System Sciences*, 4:4019, 1999.

[97] Ben Shneiderman. Dynamic queries for information seeking. *IEEE Software*, 11(6):70–77, 1994.

[98] Ben Shneiderman and Aleks Aris. Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics*, 12:733–740, 2006.

[99] Ben Shneiderman and Catherine Plaisant. Strategies for evaluating information visualization tools: Multi-dimensional in-depth long-term case studies. *Proceedings of BELIV '06*, pages 38–43, 2006.

[100] Marc Smith, Natasa Milic-Frayling, Ben Shneiderman, Cody Dunne, Tony Capone, Eduarda Mendes Rodrigues, Udayan Khourana, Annika Hupfeld, Jure Leskovec, Dan Fay, Vladmir Barash, Eric Gleave, and Adam Perer. Nodexl. `http://www.codeplex.com/NodeXL`, 2007.

[101] M. Smythe, J.M. Koerber, M. Fitzgerald, and JC Mattson. Financial impact of heparin-induced thrombocytopenia. *Chest Journal*, 134(3):568–573, 2008.

[102] Richard Snodgrass. The temporal query language tquel. *ACM Trans. Database Syst.*, 12(2):247–298, 1987.

[103] Richard T. Snodgrass. *The TSQL2 Temmporal Query Language*. Kluwer, 1995.

[104] Spotfire. http://spotfire.tibco.com/.

[105] Janice M. Stone. A graphical representation of concurrent processes. In *Proceedings of the 1988 ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging*, volume 24, pages 226–235, New York, NY, USA, 1989. ACM.

[106] Martin Suntinger, Hannes Obweger, Josef Schiefer, and M. Eduard Gröller. The event tunnel: Interactive visualization of complex event streams for business process pattern analysis. *Proceedings of PacificVis 08*, pages 111–118, 2008.

[107] Abdullah U. Tansel, M. Erol Arkun, and Gultekin Ösoyoğlu. Time-by-example query language for historical databases. *IEEE Trans. Softw. Eng.*, 15(4):464–478, 1989.

[108] K. Thompson. Regular expression search algorithm. *CACM*, 11(6):419–422, 1968.

[109] Hideyuki Tokuda, Makoto Kotera, and Clifford E. Mercer. A real-time monitor for a distributed real-time operating system. In *Proceedings of the 1988 ACM SIGPLAN and SIGOPS workshop on Parallel and Distributed Debugging (PADD '88)*, pages 68–77, New York, NY, USA, 1988. ACM.

[110] L. Tweedie and Robert Spence. The attribute explorer: Informatin synthesis via exploration. *Interacting with Computers*, 1998.

[111] Tom Ventsias. Health it an rx for health care. *Terp*, 7(1):20–23, 2009.

[112] Fernanda B. Viégas, Martin Wattenberg, Frank van Ham, Jesse Kriss, and Matt McKeon. Manyeyes: a site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1121–1128, 2007.

[113] Katerina Vrotsou, Jimmy Johansson, and Matthew Cooper. Activitree: Interactive visual exploration of sequences in event-based data using graph similarity. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):945–952, 2009.

[114] Taowei David Wang, Amol Deshpande, and Ben Shneiderman. A temporal pattern search algorithm for personal history event visualization. Technical Report HCIL-2009-14, Human-Computer Interaction Lab, University of Marland, 2009.

[115] Taowei David Wang, Catherine Plaisant, Alexander J. Quinn, Roman Stanchak, Shawn Murphy, and Ben Shneiderman. Aligning temporal data by sentinel events: discovering patterns in electronic health records. *Proceeding of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems*, pages 457–466, 2008.

[116] Taowei David Wang, Catherine Plaisant, Ben Shneiderman, Neil Spring, David Roseman, Greg Marchand, Vikrumjit Mukherjee, and Mark Smith. Temporal summaries: Supporting temporal categorical searching, aggregation, and comparison. *IEEE Transaction on Visualization and Computer Graphics*, 15(6):1049–1056, 2009.

[117] Martin Wattenberg. Baby names, visualization, and social data analysis. In *INFOVIS '05: Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, page 1, Washington, DC, USA, 2005. IEEE Computer Society.

[118] Marc Weber, Marc Alexa, and Wolfgang Müller. Visualizing time-series on spirals. In *INFOVIS*, pages 7–14, 2001.

[119] C. Williamson and Ben Shneiderman. The dynamic homefinder: evaluating dynamic queries in a real-estate information exploration system. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '92)*, pages 338–346, New York, NY, USA, 1992. ACM.

[120] K. Wongsuphasawat and B. Shneiderman. Finding comparable temporal categorical records: A similarity measure with an interactive visualization. *Proceedings of IEEE Symposium on Visual Analytics Science and Technology (VAST '09)*, pages 27–34, 2009.

[121] Fang Yu, Zhifeng Chen, Yanlei Diao, T.V. Lakshman, and Randy H. Katz. Fast and memory-efficient regular expression matching for deep packet inspection. In *Proceedings of the 2006 ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS '06)*, pages 93–102, New York, NY, USA, 2006. ACM.

[122] Moshé M. Zloof. Query-by-example: A database language. *IBM Systems Journal*, 1977.