

# A Visual Interface for Multivariate Temporal Data: Finding Patterns of Events across Multiple Histories

Jerry Alan Fails

Amy Karlson

Layla Shahamat

Ben Shneiderman

Department of Computer Science & Human-Computer Interaction Lab  
University of Maryland

## ABSTRACT

Finding patterns of events over time is important in searching patient histories, web logs, news stories, and criminal activities. This paper presents PatternFinder, an integrated interface for query and result-set visualization for search and discovery of temporal patterns within multivariate and categorical data sets. We define temporal patterns as sequences of events with inter-event time spans. PatternFinder allows users to specify the attributes of events and time spans to produce powerful pattern queries that are difficult to express with other formalisms. We characterize the range of queries PatternFinder supports as users vary the specificity at which events and time spans are defined. Pattern Finder's query capabilities together with coupled ball-and-chain and tabular visualizations enable users to effectively query, explore and analyze event patterns both within and across data entities (e.g. patient histories, terrorist groups, web logs, etc.).

**CR Categories:** H.5.2 [Information Interfaces and Presentation]: User Interfaces; H.5.2 [Information Systems]: Information Search and Retrieval — query formulation

**Keywords:** Temporal query, information visualization, user interface

## 1 INTRODUCTION

Finding patterns of events is fundamental to understanding and reasoning in many domains. For example, in the medical field, time plays a critical role in assessing individual treatments based on personal medical history, as well as broader treatment success rates based on aggregate analysis of multiple case histories. Time also plays an important role in analyzing financial events that guide investment and in tracking travel events that could direct police intervention of terrorists. Despite a pervasive presence of temporal data, traditional database management systems support only SQL queries which are not convenient for temporal pattern search. Some research systems provide temporal access languages to support limited visual queries from end-users [1-4], but many of these suffer the same accessibility difficulties of SQL, namely that the languages are not known by nor are they welcoming to practitioners or they require an understanding of the underlying database structure. Simply stated, to many practitioners these query languages are obtuse.

In contrast to some systems that support time-series data [5], we consider the multivariate and categorical nature of events. We define a temporal pattern as a sequence of point events separated

---

A.V. Williams Building, College Park, Maryland 20742  
{fails, akk, laylas, ben}@cs.umd.edu

IEEE Symposium on Visual Analytics Science and Technology 2006  
October 31 - November 2, Baltimore, MD, USA  
1-4244-0592-0/06/\$20.00 © 2006 IEEE

by time spans. We describe PatternFinder and the types of temporal patterns it supports. A representative task supported by PatternFinder would be finding and displaying patients who had high blood sugar on two consecutive blood tests within ten days. A pattern such as this would be challenging for average users to formulate using a query language, or to interpret as a text-based tabular result. Combining temporal query specification and visualization, PatternFinder enables exploration and discovery of patterns in temporal data.

We approach this problem by focusing on query and exploration of visual temporal patterns within medical histories. Although our prototype and examples are grounded in the medical domain, these techniques apply equally well to other domains such as historical records (e.g., education registration, business), transaction-based data sets (e.g., web logs, finance), and police or intelligence investigation (e.g. money laundering, terror planning).

## 2 RELATED WORK

We divide the large body of related work into three general areas: time theory, databases, and visualizations. There is considerable overlap between these areas. Related research is vast, so only a few highlights are expressed along with their relevance to our work.

### 2.1 Time theory

Much of the seminal work in computer science relating to time stems from artificial intelligence, time reasoning, and early natural language processing [6-8].

Time can be characterized by point or interval events. Snodgrass defines instance events as being absolute and intervals as being relative distances between two instances [9]. Allen introduces time intervals as the primitive for automated reasoning over temporally structured data [7]. He argues that all seemingly instantaneous events can be decomposed and thus all events can be modelled uniformly as intervals. He defines 13 mutually exclusive relations that hold between any two intervals: Equal, Before, After, During, Contains, Overlaps, Overlapped-by, Meets, Met-by, Starts, Started-by, Finishes, and Finished-by. Allen also discusses how these can be hierarchically composed, however he does not implement an interface. PatternFinder supports intervals, including: start/stop event pairs, and as sets of events that share a user-defined characteristic, which together span an extent of time. Although overlapping queries are possible, PatternFinder supports temporal sequence patterns more directly. PatternFinder currently does not allow for hierarchical or recursive query definitions as Allen described, but it does implement a powerful interval-based query system.

### 2.2 Databases

Over the years databases have progressed from theory, to small text files, to visual representations, to presently include research in spatial-temporal databases. In situating our work, this section focuses on visual interfaces and how databases deal with time.

### 2.2.1 Visual Query Interfaces to Relational Databases

Traditional access to databases has been via SQL [10], a language designed specifically to create, organize and query databases. Due to the complexity of formulating SQL queries, several approaches have made database query more accessible to a broader spectrum of users. Query By Example (QBE) presents the structure of the database as skeleton tables, and is the visual query mechanism used in Microsoft's Access [11]. Simple queries in QBE are formulated by users placing a mark in the data column they wish to be returned. Users enter constants to specify desired attribute values and variables to bind results across columns. Although visual query languages facilitate database query by avoiding SQL syntax, users must still understand the relational tables and formulate queries, using variables and other difficult concepts. PatternFinder presents users with visual constructs for querying the temporal relationships among events.

### 2.2.2 Time and Databases

Numerous extensions to the relational model have been proposed to incorporate time, such as TSQL2 [4], an extension to the SQL-92 [10] language standard. TSQL2 provides a surrogate data type which can be used to support history identity and generalized range variables. A hybrid between QBE and Extended Entity-Relationship diagrams (EER) represents queries visually as EER objects for which attributes and variables can be instantiated [12]. In order to support temporal databases, each EER object and relation is associated with a temporal object describing its valid times (when the event happened) and transaction times (when it was recorded). Temporal queries are supported by temporal operator objects in the diagram including the qualifiers: before, during, after, start and end. This work was expanded by a conceptually equivalent approach which allows users to manipulate EER type objects and specify time constraints using a rich set of menus and toolbars [13]. This hybrid system allows users to look at snapshots (data representing a given time) or slices (data over time).

Although both QBE-EER and hybrid approaches support queries over temporal events using a visual, direct-manipulation language, the temporal operators are an extension to a general-purpose database query language. Thus, neither attempt to visually encode temporal aspects and relationships in a succinct or orderly way to take advantage of the strictly ordered nature of time. MQuery is another approach for visual query via entity-relation style specification [14]. MQuery targets various types of streaming data, such as video footage and medical histories. Temporal features can be captured by before/after date specifications and left to right positioning of query objects, but it does not provide a higher level representation of the temporal aspects of the data or the query itself.

## 2.3 Related Visualizations

This section discusses three different aspects of related visualization: visual query by time intervals, other temporal visualizations, and other related work.

### 2.3.1 Visual Query by Time Interval

Chittaro and Combi proposed three alternative visual metaphors for querying temporal intervals [15]. The authors based the expressivity of their visual language on Allen's 13 relations between two intervals. Three semantically equivalent representations (elastic bands, springs and paint strips) depict horizontal bars whose ends can be constrained in such a way as to capture Allen's interval relationships.

Hibino and Rudensteiner introduced a direct manipulation Temporal Visual Query Language (TVQL) for specifying interval endpoint constraints [16] to support Allen's 13 relational primitives. Four double-sided sliders allow users to express the relationship between each pair of endpoints among two intervals. Users interact exclusively with the sliders, while a visual representation of the interval interaction is dynamically updated to provide users feedback on the meaning of the defined query.

Although both of the above systems implement all 13 of Allen's temporal relationships, the interfaces are difficult to conceptualize. We believe the simple timeline layout of queries and visualizations implemented in PatternFinder enable users to more readily create, understand and discover temporal patterns.

### 2.3.2 Visualizing Temporal Patterns

Interestingly, none of these proposals address the visualization of the returned results. However, applications such as TimeSearcher [5], Spirals [17], DataJewel [18], KNAVE [2] and LifeLines [19] offer visualizations that cluster results and highlight temporal patterns. TimeSearcher allows users to explore ordinal data by specifying queries using TimeBoxes, rectangular query operators that specify the regions in which the users are interested. Spirals uses each ring of the spiral to represent a periodic section of a time series. Color and line thickness are used to distinguish the data values. DataJewel tightly couples a familiar calendar visualization with database and algorithmic components for exploring temporal data patterns. Each calendar day displays the frequency of events using horizontal histograms such that month views offer a compact representation for users to visually detect patterns.

### 2.3.3 Other Related Work

In health care, patients may be assessed individually by nurses and physicians, or *en masse* by clinical researchers, public health officials, auditors, etc. LifeLines provides a compact hierarchical timeline visualization for personal histories organized by facets, such as doctor visits, lab tests, and medications [19]. LifeLines supports zooming, adjusting the time scale, filtering records and accessing details on demand. LifeLines supports both discrete time events, displayed as icons, and interval events, displayed as lines. Line thickness and color encode event attributes such as significance and relationship to other events. Lifelines primarily supports directed browsing with textual search, but does not offer a higher-level query mechanism for discovery across multiple records.

Many systems have built on LifeLines. Bade et. al. presented different temporal visualizations for medical data including a temporal mural that captures high-frequency data while still allowing browsing of time and data [20]. Another patient history system, CareView, used enhanced visualizations with the goal of increasing the visibility of temporal trends in clinical narratives [21]. While both of these systems contribute informative new temporal visualizations, neither allow for temporal query nor provide views of multiple entities (i.e. patients).

Although not directly related to the medical field a set of visualizations that build along the same vein as PatternFinder are those of Chen et. al [22]. Chen et. al. presented STV (Spatio-Temporal Visualization) and CAN (Criminal Activities Network). STV uses coordinated visualizations including a geo-spatial representation (a map), a timeline, and a spiral periodic visualization, all of which change as a time slider is adjusted. CAN provides a visualization for network changes over time using spring-embedded block algorithms. Both systems

incorporate time and use coordinated views as does PatternFinder, but do not allow query of temporal patterns.

Shahar proposes a Knowledge Based Temporal Abstraction model RÉSUMÉ [23] to enable domain knowledge sharing and summarization in a context sensitive manner. Knowledge-based Navigation of Abstraction for Visualization and Explanation (KNAVE) is the visualization and navigation module that operates over RÉSUMÉ to support physician decisions about treatment protocols. KNAVE offers semantic navigation for three types of tasks: domain ontology traversal, presentation adaptation according to changes in temporal granularity (e.g., aggregating daily blood sugar outcomes when viewed by year versus day), and context switching. While KNAVE supports clinicians by summarizing individual patient conditions in a disease protocol, it does not provide cross-patient query and discovery within a specified protocol nor context-free record overview or exploration.

### 3 PATTERNFINDER: QUERY AND RESULTS VISUALIZATION

We emphasize that none of the systems discussed above enable query or visualizations of patterns across multiple entities (e.g. cross-patient query). This is one of the major contributions of PatternFinder, the combined power of temporal queries with graphical visualizations.

#### 3.1 Data Set Description

In the tradition of LifeLines, we created a mock data set of over 26,000 medical events for 950 patients. Each event has a type and a value. The type supports up to three levels of a hierarchy. For example a visit to the doctor's office for a check-up would be specified by the three level hierarchy: Visit → Doctor Office → Checkup. Event values can be numeric, such as for a systolic blood pressure reading, or categorical, such as normal/abnormal blood sugar. For simplicity, in this version events are point events at the day granularity.

#### 3.2 Temporal Patterns

We define a temporal pattern as a sequence of events separated by time spans (Figure 1). In PatternFinder, users define patterns by setting constraints on Events and Time Spans. PatternFinder's query interface maps Events and TimeSpans to form-fill-in elements that support a rich set of pattern queries, presented

formally in Section 4. Pattern matches are then displayed for visual exploration. To support pattern discovery, the PatternFinder interface is divided into two main panels: the pattern design panel and the results visualization panel.

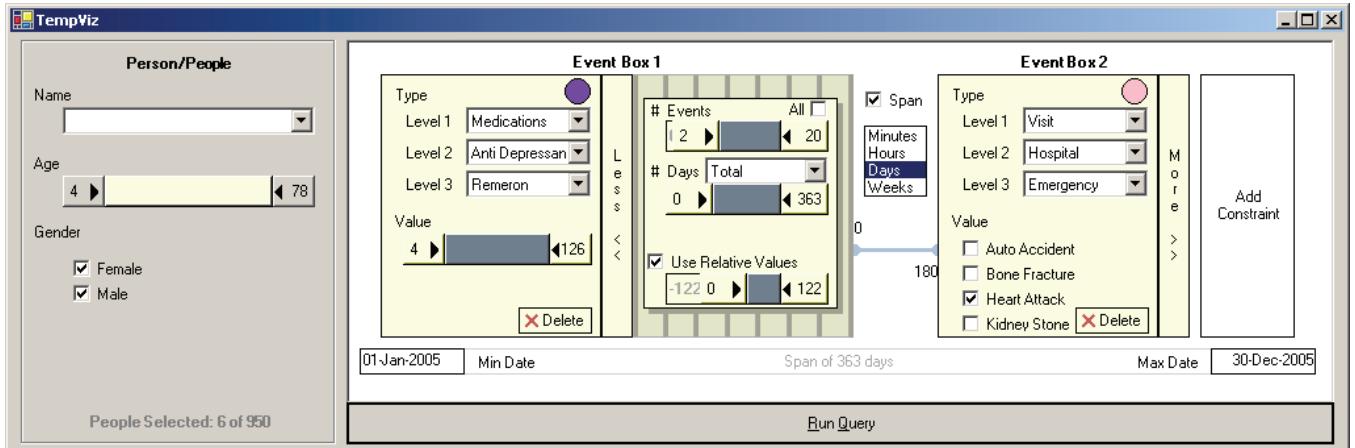


**Figure 1:** A Pattern is a sequence of events and time spans.

#### 3.3 Pattern Design

The top half of the PatternFinder interface supports pattern design and specification (Figure 2). Users consider elements of the pattern design panel from left to right. Because patterns are applied within individual patient histories, the leftmost panel allows users to restrict the types of patients in the results set in the Person/People panel. Users can restrict patients by name, by selecting from a list of patients, or by entering a text string. Users can also restrict patient age range and sex. Changes to the Patient/People panel are dynamic queries [24] that update results immediately in the results visualization panel (Figure 4). The number of patients that meet the current patient selection criteria and design pattern (if specified) is also displayed in the panel. All double-sliders not only allow adjustments via the mouse, but clicking on the end buttons (or using key strokes to select it) allows for rapid textual entry of precise data values.

The temporal pattern panel lies to the right of the Person/People panel, and allows users to chain Events together to form a pattern query. The timeline constraints and the left-to-right layout of Event boxes echoes cultural intuition that time flows in a left to right manner (e.g. Event Box 1 specifies an Event that occurs before the one in Event Box 2). Absolute start and end dates can be specified for the temporal query; by default the absolute dates reflect the maximal TimeSpan of Events in the underlying data set. Event Boxes initially present only the widgets for setting non-temporal attributes. As users select attributes from top to bottom, they define increasingly strict constraints on the types and values of related Events. Pull-down lists contain the labels that define a three level type hierarchy: the Event Type, Event Classification, and Event Name as defined within the underlying data set. By default only a top level Event Type is selected. If



**Figure 2:** Pattern query panel. The Person/People panel at the left allows users to specify the types of patients to consider. The white temporal pattern panel on the right allows an arbitrary number of Events and TimeSpans to be defined (Section 4 discusses patterns details).

This pattern specification is any patient who received increasing dosages of Remeron followed by a heart attack within 180 days (along with the events constituting the temporal pattern match).

users select a specific Event Name, such as Systolic Blood Pressure, the values associated with that Event will be made available for further selection and query specificity. Once all three event levels are specified, events with numeric values are presented with a double ended slider that defaults to the complete Event value range, while Events with categorical values are presented with a checkbox for each category.

By default no Event boxes are displayed – only an “Add Constraints” widget, which adds a new Event to the right of the chain when clicked. Users can delete Events by selecting the delete button. Also, by default, each Event is assigned a unique color marker, shown as a circular color chip in the Event Box. This allows users to visually map each Event in the results visualization to the associated Event in the pattern definition panel. Users can customize the assigned color by clicking on the color chip.

Event Box 2 (Figure 2) is an example of an EventSet, which provides users the ability to define constraints on multiple events of the same type. To define an EventSet, users select the “More >>” tab, which opens to reveal widgets for setting window, cardinality and relative value constraints. Selecting the Span checkbox between two Event Boxes places a time restriction between adjacent Events. The minimum TimeSpan in days is positioned at the upper left of the TimeSpan bar that connects Events, with the maximum span in days positioned at the lower right of the bar. For example, by opening the “More >>” tab a user could easily specify the following query: three blood pressure tests occurring within a one month period, each time the systolic pressure increase by at least 10 points. Another example query is: all patients who were prescribed decreasing dosages of the heart medication Plaxin over a six month period and later had a heart attack. For more details on the types of queries that can be specified and how to specify them in PatternFinder, see Section 4.

### 3.4 Result Visualization

Although others have explored temporal query interfaces, most have ignored the display of the resulting matches. Conversely, medical systems [3, 19] have provided interfaces for browsing temporal patient data, but have rarely provided support for querying patient patterns, nor for comparison of patterns across patients. PatternFinder provides both pattern formulation and result exploration. The lower half of the interface (Figure 4) shows result visualization, designed according to the well-established information-seeking mantra: overview first, zoom and filter, then details-on-demand [25]. Although the visualization is not unique, its coupling with the pattern specification panel along with other features described below, make for a powerful combination enabling temporal pattern discovery.

```

SELECT P.*
FROM Person P, Event E1, Event E2,
Event E3, Event E4
WHERE P.PID = E1.PID
AND P.PID = E2.PID
AND P.PID = E3.PID
AND P.PID = E4.PID
AND E1.type = "Medication"
AND E1.class = "Anti Depressant"
AND E1.name = "Remeron"
AND E2.type = "Medication"
AND E2.class = "Anti Depressant"
AND E2.name = "Remeron"
AND E3.type = "Medication"
AND E3.class = "Anti Depressant"
AND E3.name = "Remeron"
AND E2.value > E1.value
AND E3.value >= E2.value
AND E2.date > E1.date
AND E3.date >= E2.date
AND E4.type = "Visit"
AND E4.class = "Hospital"
AND E4.name = "Emergency"
AND E4.value = "Heart Attack"
AND E4.date >= E3.date
AND 180 <= (E4.date - E3.date)

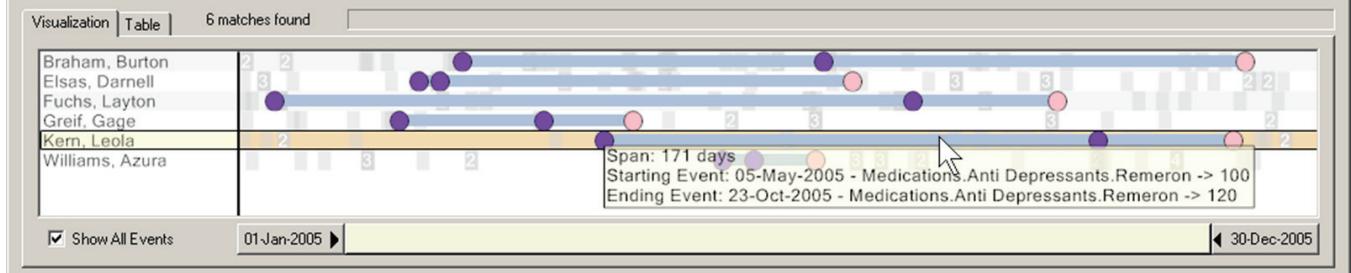
```

**Figure 3:** Limited SQL version of query in Figure 2, patients who received one or two increasing dosages of Remeron followed by a heart attack within 180 days (corresponding events are not returned).

#### 3.4.1 Overview

PatternFinder’s results visualization (Figure 4) shows a graphical table of pattern matches. Each row represents a single pattern match for a single patient. Rows are ordered first by patient, then within a patient in order by earliest Event. A patient for whom the pattern has matched multiple times will be associated with multiple rows. To distinguish one patient’s results from another’s, the row background alternates between white and light gray from one patient to the next and row headings display the name of the patient for whom the row is associated. Rows which do not fit within the vertical constraints of the display can be accessed by scrolling the display panel. Columns represent days.

Pattern matches are depicted as a timeline in a ball-and-chain fashion, with Event points represented by circles and TimeSpans by blue bars between the circles. The color of an Event point within the visualization corresponds to the color assignment of the associated Event in the pattern query definition. The Event point colors can be especially helpful for distinguishing Events from two or more EventSets, or comprehending the pattern match when constraint specifications allow the order of events in the match to



**Figure 4:** Pattern Result Visualization. Each row represents a single pattern match for a patient. Gray blocks indicate an event occurred on that day, with numbers indicating that multiple events occurred. Mouse over provides details about all events.

The results show four matches of people who received increasing dosages of Remeron followed by a heart attack within 180 days.

differ from the left-to-right layout of Event Boxes in the pattern definition. By default, the “Show All Events” checkbox is selected, which displays the matched patterns in the context of the patients’ entire event histories; each event is shown as a light gray rectangle and numbered if multiple events occurred on the same day. Event context can be useful for identifying interesting patterns or suggesting pattern causality.

As an alternative to the pattern visualization, a Table tab (Figure 5) displays the result set as a table, aggregating data by person, which users can expand to see the individual matches. This view supports rapid sorting, allowing users to quickly discover minimum and maximum values, dominant event types, event names, patient characteristics, and number of matches per patient.

### 3.4.2 Coordinated Views

The table view (Figure 5) is tightly coupled with the visualization view (Figure 4) allowing users to view a tabular summary by person, perform a sort by the number of matches and return to the visualization to view the matches in order of the number of matches per person. This coupling adds yet another approach for interactively finding the most important and relevant trends and for testing a wider range of hypotheses. As an example, a user might specify a pattern query, go to the table view, sort by age and return to the default graphical visualization to explore effects correlated to patient age.

FullName	LastName	FirstName	Age	Gender	TotalNumEve	TotalNumMat
Braham, Burt	Braham	Burton	23	Male	41	1
Elsas, Darnell	Elsas	Darnell	17	Male	44	1
Fuchs, Layton	Fuchs	Layton	27	Male	29	1
Greif, Gage	Greif	Gage	43	Female	24	1
Kern, Leola	Kern	Leola	52	Female	31	1
Williams, Azu	Williams	Azura	78	Female	34	1

**Figure 5:** Table result view, each row represents a person with one or more pattern matches.

### 3.4.3 Zoom and Filter

Initially all pattern matches for a patient are scaled to fit horizontally within the window, providing an overview of the results. A double ended query slider below the visualization indicates the endpoints of the time period displayed. Either end of the slider may be manipulated to change the time period presented in the visualization, in effect zooming the display area, and filtering out uninteresting time periods.

### 3.4.4 Details-on-Demand

Visually distinguishing Events from the TimeSpans between Events has two main benefits. The first is that the distinct objects present distinct targets for tool tips. Placing the cursor over any Event or TimeSpan triggers details-on-demand in the form of a tooltip (as shown in Figure 4). Another advantage to depicting Events and TimeSpans separately is that short TimeSpans can be more easily compared.

## 3.5 Implementation

PatternFinder was developed in Visual C# .NET with the Piccolo.NET development toolkit for scalable and zoomable user interfaces [26]. Queries are evaluated using a sequential scan of a memory-resident data set. We do not use a relational database nor SQL to perform query evaluation. The queries we support could conceivably be issued using standard SQL, however, our emphasis on supporting arbitrarily long patterns in which Events

can be specified by time window, cardinality and/or relative value constraints, are extremely difficult to express in SQL. Figure 3, shows an example that will return only the patient’s names that fit a temporal pattern — not the actual events. We believe that PatternFinder’s visual query interface provides simplicity not accessible in SQL and a level of power beyond that which can be offered by a tabular or EER-based representation. PatternFinder may be considered a specialized, mixed visual and form-fill-in approach to temporal query specification and visualization.

## 4 PATTERN LANGUAGE

The interface elements of our query language were designed to support a rich set of patterns which we formalize here using a modified Bachus Naur Form (BNF): optional constructs are enclosed in brackets [], constructs repeated zero or more times are in braces {}, and constructs repeated one or more times are in braces followed by a plus {}+. Parentheses () are used to enclose groups of alternative items, each separated by a |. Non-terminals are shown in **bold**.

**Pattern** = **AbsoluteDate** **EventSeq** **AbsoluteDate**  
**EventSeq** = **Event** {[**TimeSpan**] **EventSeq**}

A pattern is bounded by a global time frame – a minimum and maximum calendar date (**AbsoluteDate**) between which an event sequence is defined. As a base case, a pattern can be a single **Event**. More interestingly, however, patterns are specified as an **Event**, **E1**, followed by at least one additional **Event**, **E2**, where **E1** and **E2** are separated by a **TimeSpan**. This structure allows for the composition of chains of Events related to one another by time constraints, which together form a Pattern. This basic template for a pattern is displayed graphically in Figure 1, and is supported in PatternFinder as the ability to add an arbitrary number of Event Boxes and intervening **TimeSpan** definitions.

When forming patterns, there is a trade-off between the specificity of the pattern description and the generalizability of the result set. While assigning values to all attributes of a pattern can be useful for some tasks (“Find all patients who had cholesterol above 200 and who were later hospitalized for a heart attack”), it can also be overly restrictive, and fail to return results that are of general interest. By relaxing constraints systematically across Events and **TimeSpans** in the pattern, users can tailor their pattern-finding to support different types of tasks.

Hochheiser & Shneiderman outline types of temporal queries that are of interest for single-attribute time series within the financial domain [5]. We propose a complementary taxonomy (Tables 1 and 2) of temporal patterns for multivariate and categorical temporal data, cast in terms of the Events and **TimeSpans** of our pattern model. The taxonomy is hierarchical in the sense that each pattern, other than the root **Events** (**E**) pattern, can be defined in terms of another pattern which has been modified in one of three ways: the constraints on the **Event** have been relaxed, the constraints on the **TimeSpan** have been relaxed, or the values of one or more events have been constrained to depend on one or more other events. By systematically applying these rules, we define a taxonomy of increasingly complex patterns supported by PatternFinder (Tables 1 and 2).

### 4.1 E Pattern Queries

As a base case, **E** patterns require users to specify all non-temporal attributes of **Events** (**E**) — the three type levels and a value. Numeric **Event** values are specified as an inclusive range with a **MinValue** and **MaxValue** to support exactly, above, below, and between constraints. Setting **MinValue**=**MaxValue** enforces an exact value, while setting **MinValue** to the lowest value in the

data set or MaxValue to the highest value in the data set is equivalent to  $\geq \text{MinValue}$  or  $\leq \text{MaxValue}$ . One or more items in a categorical enumeration can also be specified.

**Event** = EventType EventClass EventName [**ValueSpec**]  
**ValueSpec** = (MinValue MaxValue | {EnumVal}+)

Using PatternFinder, E patterns are formed by specifying values for all graphical elements of a single unexpanded Event Box (see Event Box 2 of Figure 2).

#### 4.2 E-FT and E-VT Pattern Queries

An E-FT pattern has two Events, related by a relative TimeSpan of fixed (FT) length, while an E-VT pattern allows a TimeSpan of variable (VT) length between Events. Relaxing the TimeSpan constraint from fixed to variable allows matches to be returned that agree in the number, type and order of Events, but which have bounded variability in time between Events. Especially in the medical field, precise time constraints between Events may be impractical since, for example, medication response and surgery recovery times vary widely across patients. E-FT and E-VT patterns are both supported by defining TimeSpans.

**EventSeq** = Event {[TimeSpan] EventSeq}

**TimeSpan** = MinDays MaxDays

To describe a TimeSpan, users specify a minimum number of days (MinDays) and a maximum number of days (MaxDays) between two Events. To construct an E-FT-Pattern, users specify that MinDays = MaxDays, and for an E-VT-Pattern, that MinDays < MaxDays. Leaving MinDays and MaxDays unspecified places no constraints on the TimeSpan between Events except that for

any pair of Events, the left Event occurs before the right Event.

In PatternFinder, no span is defined initially between events. If desired, users can click on the Span checkbox to activate it, then use the tab button to move between the min and max day specifications or use the mouse to click on them and change them.

#### 4.3 E\*-VT Pattern Queries

E\*-VT patterns lift the restriction that all non-temporal attributes be specified, effectively allowing wildcard query (\*) on event attributes. Our hierarchical event structure makes this pattern especially powerful. Users can query by event type, by event classification (restricting the event type), by event name (restricting the event type and classification) or by event outcome (restricting all levels). Thus, E\*-VT patterns allow users to describe events at four levels of specificity:

**Event** = EventType [ClassSpec]

**ClassSpec** = EventClass [DetailSpec]

**DetailSpec** = EventName [ValueSpec]

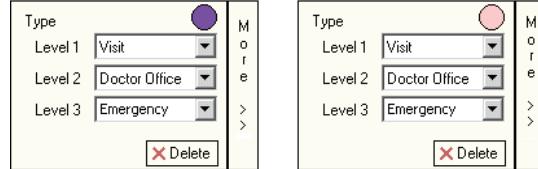
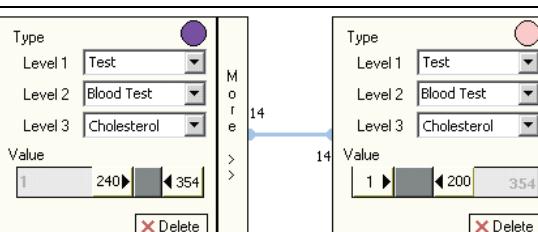
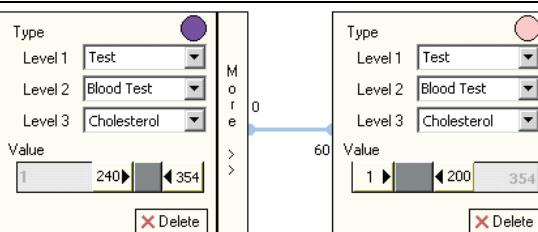
**ValueSpec** = MinValue MaxValue | {EnumVal}+

The advantage of E\*-VT-Patterns over other patterns is that they allow discovery of interactions between classes of events rather than only specific outcomes. Our faceted Event architecture enforces structure upon the pattern matches, even as specificity constraints are lifted.

#### 4.4 E\*WC-VT Pattern Queries

PatternFinder's pattern language allows users to construct Event patterns of arbitrary length and specificity. While this approach supports queries of the form "Find patients who had a

Table 1: Temporal Query Taxonomy: Simple Events and TimeSpans

Pattern	Restrictions	Example
E Events Only	<b>Event</b> : Specify all non-temporal attributes of one or more events.  <b>TimeSpan</b> : No span. Left to right ordering implicitly denotes "any time later".	 <p>Find patients who have had at least 2 emergency doctor visits.</p>
E-FT Fixed TimeSpans	<b>Event</b> : Specify all non-temporal attributes of <u>two or more events</u> .  <b>TimeSpan</b> : Relative TimeSpan of <u>fixed size</u> .	 <p>Find patients whose cholesterol was above 240 but two weeks later was below 200.</p>
E-VT Variable TimeSpans	<b>Events</b> : Same as E-FT.  <b>TimeSpan</b> : Relative TimeSpan of <u>variable size</u> .	 <p>Find patients whose cholesterol was above 240 but fell below 200 within 2 months.</p>

Hemoglobin Test between 25 and 50 during January 2004” and “Find patients who had two Heart Attacks within 6 months of each other”, it does not scale well to pattern queries of the form “Find patients whose Cholesterol was above 240 for a week”, which would require chaining 7 identical Events together with a TimeSpan of 1 day between each pair. The query might be further complicated for events with finer granularity than a day. To address this problem, we extend the pattern model to include E\*WC-VT patterns, which define pattern EventSets whose non-temporal attributes are identically constrained but whose temporal attributes vary within a specified time window (W). A time window can be defined to contain all Events in the EventSet using the Total constraint, or instead between pairs of consecutive Events in the EventSet using the Between constraint. The EventSets are also constrained by the number of elements that must be present in the set, or its cardinality (C):

**Event** = EventType [ClassSpec] [EventSetSpec]  
**EventSetSpec** = [WindowSpec] [Cardinality]  
**WindowSpec** = (Total | Between) MinDays MaxDays  
**Cardinality** = (All | MinNumEvents MaxNumEvents)

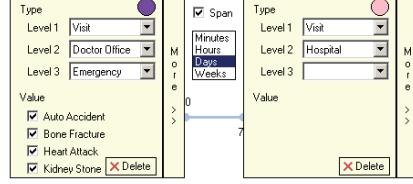
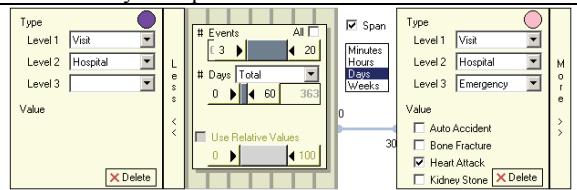
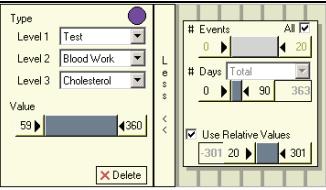
These simple additions greatly increase the power of PatternFinder and the types of queries supported. Now, queries of the form “Find patients whose Cholesterol was above 240 for a week” can be compactly specified as a single EventSet by constraining the event window to a Total of MinDays=MaxDays=7 to denote that the tests must fall within a window of exactly 7 days and the cardinality criterion to “all”; these settings translate to finding patients for whom all cholesterol tests within a week’s time frame were above 240. Note that “All” is a special case. Instead of users committing to a specific number or range of events, “All” translates to “For any events that meet the window criteria, all must meet the Event attribute criteria in

order to be considered a match”.

Perhaps more importantly, E\*WC-VT patterns support a type of trend query and discovery, in which event sets that share local characteristics can be compared to one another, such as to discover that a particular medication had a positive or negative impact. Without window and cardinality constraints for an event set, only weaker queries would be possible. For example, without such patterns one might be able to say that a patient’s cholesterol was above 240 on a single day and below 200 on another day two months later, but would now be able to make the stronger statement that a patient’s cholesterol remained above 240 for a period and then remained below 200 for a subsequent period, which is more convincing evidence of the efficacy of a cholesterol-lowering medication.

Support for E\*WC-VT patterns allows users to specify other helpful queries, such as finding patients that have had at least 2 heart attacks in the last year (returning *all* that occurred), or patients with chronic high cholesterol who have had no heart attacks. The astute reader will notice that our definition of E\*WC-VT patterns can lead to a multiple Cartesian products representing an explosion in the size of the search space and the resulting pattern matches, negatively impacting system response time and user comprehension of results. We suspect, however, that many such queries do not reflect realistic user tasks. We can easily constrain our query interface in subsequent iterations to prevent pathological queries, but refrain from doing so until we better understand typical usage scenarios. At present, our visualization displays a pattern match (row) for every pair of events that satisfy the E\*WC-VT. In the future we envision that matches could be collapsed into a single row to simplify user comprehension.

**Table 2:** Temporal Query Taxonomy: EventSets and TimeSpans

Pattern	Restrictions	Example
<b>E*-VT</b> Variable Events Variable TimeSpans	<b>Event:</b> Specify a <u>subset</u> of non-temporal attributes of one or more events.  <b>TimeSpan:</b> Same as E-VT	 <p>Find patients who had an emergency doctor's visit followed by a hospitalization within a week.</p>
<b>E*WC-VT</b> Linked EventSet and Event	<b>Event:</b> Specify a subset of non-temporal attributes of one or more event <u>sets</u> . The elements of an event set are defined by a relative time <i>window</i> during which all events in the set must occur, as well as <i>cardinality</i> - the number of events that must occur within the window.  <b>TimeSpan:</b> Same as E-VT.	 <p>Find patients who had 3 or more hospitalizations within two months followed by a heart attack hospitalization within a month.</p>
<b>f(E*WC)-VT</b> EventSet	<b>Event:</b> Same as E*WC-VT plus a functional condition that must hold across all members of the set.  <b>TimeSpan:</b> Same as E-VT	 <p>Find patients whose cholesterol increased by at least 20 points each reading for 3 months.</p>

#### 4.5 $f(E^*WC)$ -VT Pattern Queries

$f(E^*WC)$ -VT patterns follow directly from  $E^*WC$ -VT patterns. Since  $E^*WC$ -VT patterns define event sets, it is natural to impose functional ( $f$ ) constraints among the elements of the set, which leads to  $f(E^*WC)$ -VT patterns. For example, instead of requesting all cholesterol events within a week to have a value above 240, a user might request that all cholesterol events within a week fall within some 20 point range above 240. This differs from specifying a value range with a double slider, since we do not care which 20 point range above 240, but we do care that no event has a value that is more than 20 points different than any other event in the set. This is a constraint among all matching Events. Rather than specifying a range of values that EventSet elements must satisfy, we might specify other functional relationships among values, such as monotonically increasing or decreasing, which would be a constraint between consecutive matching Events. This type of query is analogous to interval trending [5]. Both types of event set value constraints can be captured as follows:

**EventSetSpec** = [WindowSpec] [Cardinality] [RelativeRange]  
**RelativeRange** = (Among | Between) MinValue MaxValue

PatternFinder 1.0 supports only the “Between” constraint.

## 5 EVALUATION

A class project [27] conducted a usability test with 6 college students and 2 physicians performing 9 tasks each using a portion of a hospital’s patient data containing 24,819 events for 552 patients. One physician was a practitioner, the other a researcher. The physician practitioner expected he would mostly use the simpler Event-only queries, while the physician researcher anticipated using the expanded EventSet queries more frequently. All users were successful with simple queries, but had some difficulty using the Span feature as well as formulating queries with more than 3 events, leading to suggestions for redesign and improved training. These changes have been made and a trial deployment is planned for the Washington Hospital Center, supporting both physician practitioners and researchers.

## 6 CONCLUSION

We extend early systems for visual temporal query languages and navigable visualizations of temporal data sets. Our query taxonomy and interface offer ad hoc search and discovery of temporal patterns within multivariate categorical data sets. The taxonomy describes temporal patterns as sequences of Events and the TimeSpans which separate them. PatternFinder 1.0’s visual query interface allows users to define powerful patterns. The resulting pattern matches for all patients are shown in a single, dynamic overview to facilitate both within-patient and between-patient discoveries. The same techniques illustrated in PatternFinder with medical history data are applicable to the numerous domains that deal with temporal data. We believe that the taxonomy and initial implementation are major contributions in an important topic. This paper is designed to promote further work by others and gain feedback as we move on to implementation and algorithmic refinements, user testing, and validation in diverse application domains.

## REFERENCES

- [1] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini, "Visual query systems for databases: a survey," *Journal of Visual Languages and Computing*, vol. 8, pp. 215-260, 1997.
- [2] C. Cheng, Y. Shahar, A. Puerta, and D. Stites, "Navigation and visualization of abstractions of time-oriented clinical data" Stanford University Section on medical informatics technical report SMI-97-0688, 1997.
- [3] C. Cheng and Y. Shahar, "Intelligent visualization and exploration of time-oriented clinical data," *Topics in Health Information Management*, vol. 20, pp. 15-31, 1999.
- [4] C. S. Jensen and R. T. Snodgrass, "Temporal data management," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, pp. 36-44, 1999.
- [5] H. Hochheiser and B. Shneiderman, "Dynamic query tools for time series data sets: Timebox widgets for interactive exploration," *Information Visualization*, vol. 3, pp. 1-18, 2004.
- [6] B. C. Bruce, "A model for temporal references and its application in a question answer program," *Artificial Intelligence*, vol. 3, pp. 1-25, 1972.
- [7] J. F. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, pp. 832-843, 1983.
- [8] K. M. Kahn and A. G. Gorry, "Mechanizing temporal knowledge," *Artificial Intelligence*, vol. 9, pp. 87-108, 1977.
- [9] R. T. Snodgrass, *Developing time-oriented database applications in SQL*. San Francisco: Morgan Kauffman Publishers, Inc, 1999.
- [10] ANSI, *The database language SQL*: Document ANSI X3.135-1992, 1986.
- [11] M. Corporation, "Microsoft Access," 2000-2006.
- [12] V. Kouramajian and M. Gertz, "A graphical query language for temporal databases," in *Proc. of Object-Oriented and Entity Relationship Modeling*: Springer-Verlag, 1995, pp. 388-399.
- [13] S. F. Silva, U. Schiel, and T. Catarci, "Visual query operators for temporal databases," in *International Workshop on Temporal Representation and Reasoning*, 1997, pp. 46-53.
- [14] J. D. N. Dionisio and A. F. Cardenas, "MQuery: A visual query language for multimedia timeline and simulation data," *Journal of Visual Languages and Computing*, vol. 7, pp. 377-401, 1996.
- [15] L. Chittaro and C. Combi, "Visualizing queries on databases of temporal histories: new metaphors and their evaluation," *Data and Knowledge Engineering*, vol. 44, pp. 239-264, 2003.
- [16] S. Hibino and E. A. Rundensteiner, "User interface evaluation of a direct manipulation temporal visual query language," in *ACM Multimedia*: ACM Press, 1997, pp. 99-107.
- [17] M. Weber, M. Alexa, and W. Muller, "Visualizing time series on spirals," in *IEEE Symposium on Information Visualization*: IEEE Press, 2001, pp. 7-14.
- [18] M. Ankerst, D. H. Jones, A. Kao, and C. Wang, "DataJewel: Tightly integrating visualization with temporal data mining," in *ICDM Workshop on Visual Data Mining*, 2003.
- [19] C. Plaisant, B. Milash, A. Rose, S. Widoff, and B. Shneiderman, "LifeLines: Visualizing personal histories," in *CHI 1996*: ACM Press, 1996, pp. 221-227.
- [20] R. Bade, S. Schlechtweg, and S. Miksch, "Connecting time-oriented data and information to a coherent interactive visualization," in *CHI 2004*: ACM Press, 2004, pp. 105-112.
- [21] L. Mamykina, S. Goose, D. Hedqvist, and D. V. Beard, "CareView: Analyzing nursing narratives for temporal trends," in *Extended Abstracts of CHI 2004*: ACM Press, 2004, pp. 1147-1150.
- [22] H. Chen, H. Atabakhsh, C. Tseng, B. Marshall, S. Kaza, S. Eggers, H. Gowda, A. Shah, T. Petersen, and C. Violette, "Visualization in law enforcement," in *Extended Abstracts of CHI 2005*: ACM Press, 2005, pp. 1268-1271.
- [23] Y. Shahar, "Dynamic temporal interpretation contexts for temporal abstraction," *Annals of Mathematics and Artificial Intelligence*, vol. 22, pp. 159-192, 1998.
- [24] C. Ahlberg and B. Shneiderman, "Visual information seeking: Tight coupling of dynamic query filters with starfield displays," in *CHI 1994*, 1994, pp. 313-317.
- [25] B. Shneiderman and C. Plaisant, *Designing the user interface: Strategies for effective human-computer interaction*, 4th ed: Addison-Wesley, 2005.
- [26] "Piccolo.NET," University of Maryland’s Human Computer Interaction Lab, 2006.
- [27] J. Brennan, H. Song, and N. Negahban, "Hippocrates archive - comprehensive patient history search system interface design report," 2005, pp. www.cs.umd.edu/class/fall2005/cmsc434/projects/hippocrates.