Event Sequence Queries by Example or by Filters: Design and Empirical Evaluation
Krist Wongsuphasawat∗ Catherine Plaisant† Ben Shneiderman‡
Department of Computer Science & Human-Computer Interaction Lab, University of Maryland, College Park, MD 20742

ABSTRACT
Specifying event sequence queries is challenging even for skilled computer professionals familiar with SQL. Most graphical user interfaces for database search use a query-by-filters approach, which is often effective, but applies an exact match criteria. We describe a new query-by-example interface, in which users specify a pattern by simply placing events on a blank timeline, producing a similarity-ranked list of results. Users customize the similarity measure by four decision criteria, enabling them to adjust the impact of missing, extra, or swapped events or the impact of time shifts. We describe an example of use with electronic health records based on our ongoing collaboration with hospital physicians. Then we report on a controlled experiment with 18 participants that compared query-by-filters and query-by-example features. We report on the advantages and disadvantages of each approach and conclude with recommendations for the design of a hybrid approach combining both interfaces.

Index Terms: H.5.2 [Information Interfaces and Presentation]: User Interfaces—[Graphical user interfaces (GUI)]

1 INTRODUCTION
Organizations are increasingly collecting event sequence records: health organizations have Electronic Health Record (EHR) databases containing millions of records of heart attacks, hospital admissions and treatments. Transportation systems generate logs of incidents and the timing of their management (first response, lane closures, arrival of equipment on the scene). Academic institutions keep detailed records of the educational advancement of their students (classes, milestones reached, graduation, etc.). Querying those large databases of event sequences to answer specific questions or look for patterns is an important activity.

Specifying temporal queries in SQL is difficult even for computer professionals specializing in such queries. Database researchers have made progress in representing temporal abstractions and executing complex temporal queries [28, 29, 10], but there is very little research that focuses on making it easy for end users such as medical researchers, traffic engineers, or educators to specify the queries and examine results interactively and visually.

To the best of our knowledge, most existing visual temporal query tools have used a query-by-filters approach, in which users apply filters, such as sliders or drop-down lists, to specify queries, such as event sequences. The tool returns only the records that match the query. LifeLines2 [35], a previous project from our research group, followed this approach (See Figure 1.) This approach works well when the users are fairly certain about their query (e.g. “find all patients admitted to the emergency room within a week after leaving the hospital.”) However, exploratory search [33, 37], in which users are uncertain about what they were looking for are common. Broad queries return too many results that are not relevant. Narrow queries miss records that may be “just off” (e.g. 7.5 days, or admitted but to another unit of the hospital). A more flexible query method could help exploratory searchers.

Other query tools follow a query-by-example approach, which allows users to sketch an example of what they are seeking directly and use a similarity measure to find similar results. Our early work on Similan [38], demonstrates a temporal query tool that could search for records that are similar to a selected existing record in the database. Similan employed a simple similarity measure called the M&K (Match and Mismatch) measure, which could give a score that tells how much two records are similar. The similarity scores between the target and all records were computed and records sorted by their similarity scores. This initial prototype was found promising by medical researchers but its usefulness limited for several reasons: it could only deal with six event types and events were binned into fixed intervals; it only allowed the users to select an existing record from the database as a target (not to specify an example of their choice); it took the entire range of the data into account (while users might be interested in only certain time ranges); the similarity measure considered only two types of difference.

We propose a new query-by-example approach for querying event sequences by developing a new version of Similan. (See Figure 2.) Users can specify a temporal pattern example by placing events on a blank timeline, and search for records that are similar to their example. We improve and extend the M&K measure to be faster and customizable by four decision criteria, responding to users’ need for rich and flexible definitions of similarity.

We report on a controlled experiment to compare the features of the query-by-filters and query-by-example interfaces using LifeLines2 and the new Similan. We summarize the advantages and disadvantages of each approach and propose the guidelines for combining the two approaches.

We first describe an example of use from our ongoing collaboration with a hospital physician and review the related work in more detail, we then describe our user interfaces, explain the similarity measure and report on a controlled experiment that compared the query-by-filters and query-by-example features. We conclude with general recommendations for the design of future event sequence query interfaces.

2 EXAMPLE OF EVENT SEQUENCE ANALYSIS
Our physician partners at the Washington Hospital Center are analyzing sequences of patient transfers within the ICU (Intensive Care Unit) department for quality assurance. Two examples of sequences of interest are: 1) Patients who were transferred from the ICU room to the Floor (normal) room and then back to the ICU room 2) Patients who were admitted to the emergency room then transferred to the Floor room and then to the ICU room. Those patterns are known as bounce backs and correspond to a quality metric for the hospital that is very rarely monitored. Time constraints are associated with these sequences (e.g. the bounce back should occur within a certain number of days). The tool our partners have been using to find these patterns is an MS Excel spreadsheet. They exported data from the database and wrote formulas to express the queries.

∗e-mail: kristw@cs.umd.edu
†e-mail: plaisant@cs.umd.edu
‡e-mail: ben@cs.umd.edu
Figure 1: Query-by-Filter interface (LifeLines2) showing the results of a query for patients who were admitted to the hospital then transferred to the Intensive Care Unit (ICU) within a day, then to an Intermediate ICU room on the fourth day. The users has specified the sequence filter on the right selecting “Admit”, “ICU” and “Intermediate” in the menus, and aligned the results by the time of admission. The distribution panel in the bottom of the screen shows the distribution of “Intermediate”, which gives an overview of the distribution and has allowed users to select the time range of interest (e.g. on the fourth day) by drawing a selection box on the distribution bar chart.

Figure 2: Query-by-Example interface (Similan) with the same query as in Figure 1. Users specify the query by placing events on the target panel. To set the overall time range of interest and focus on events within this range users draw a box (red line) on the target. After clicking on “Search”, all records are sorted by their similarity to the target. The similarity score is represented by a number that is the total score and a bar with four sections. A longer bar means a higher similarity score. Each section of the rectangle corresponds to one decision criterion, e.g. the top two records has longer leftmost section than the third record because it has lower time difference so the Avoid Time Difference Score (AT) is high, resulting in longer bars. The four decision criteria are equally weighted by default but the users can adjust the weight manually. (See Figure 3.)
An interview with the physician who performed these tasks for his research revealed frustration with the approach because of its complexity and time-consuming aspects (it took many hours to create the formulas). We also asked about the possibility of performing these queries using SQL. He explained that SQL was even harder for him and he was not quite sure how to start (even though he had earned a Computer Science undergraduate degree in addition to his medical degree.)

To confirm the difficulty of specifying such queries in SQL we conducted a small pilot study. We gave 6 computer science or computer engineering students, who had completed a database course, the schema of a simple dataset and asked them to write a SQL query to find patients who were admitted to the hospital, transferred to Floor, and then to ICU (no time constraints were to be specified). We allowed them to use books or search engines to find the answer but set a 30-minute time limit. Only one participant returned a complete answer, which required multiple queries and a custom function to be written. Other participants produced incomplete queries and uncertainty, such as “Use join and group by, I think”. Three participants did not even return any answer. This pilot study confirmed that SQL is not appropriate for the most analysts that do not have strong computer knowledge.

Using a Multi-dimensional In-depth Long-term Case Study methodology [27], we worked with the physician by assisting him through the analysis of his data using both of our tools LifeLines2 (query-by-filters) and Similarn (query-by-example). The physician reported that he was able to specify his queries easily in much shorter time than with the spreadsheet, and that he discovered additional patients who he had missed using his old approach. He clearly stated that visualizing the results gave him a better understanding of the data, which could not have been achieved from his spreadsheet or an SQL query. He also hinted at advantages and disadvantages of both visual approaches. For example he felt that query-by-example made it easier to specify the pattern but that looking at the results ranked by similarity was difficult and sometimes frustrating as he was not always confident that the similarity measure was adequately computed to fit his needs. (The computation is explained in detail later in this paper.) Those contrasting benefits led us to design the controlled experiment to see if we could confirm those impressions and better understand which query method is better suited for different tasks.

3 Related Work

A time series is a sequence of data values, measured at successive times. Event sequences, e.g. ICU patient transfers, is one type of time series. However, unlike the numerical time series, e.g. stock indices, the data value is not a number (1, 2, 3, 5, ...), but a category ("Admitted", "ICU Room", etc.).

3.1 Query Languages

A traditional approach to query temporal data was to use database query languages. According to Chomicki [9] and Tansel and Tin [31], many research projects were conducted on designing temporal databases and extending standard query languages into temporal query languages. Some of the well-known languages were TQuel [28], TSQL2 [29] and HRDM (Historical Relational Data Model) [10]. However, these temporal query languages were built on top of their specific data models and users had difficulty in learning their unique syntaxes, concepts, and limitations.

3.2 Query-by-Example Languages

To provide a high-level language that offers a more convenient way to query, the query-by-example languages were introduced. According to Ozsoyoglu and Wang [23], the early idea of query-by-example was a language that users specify an example output by directly making entries into relation skeletons instead of writing lengthy queries, making it simpler for the users to specify a query. The first was Zloof’s Query-by-Example [39], but it was refined by other developers [8, 20, 40, 16, 24, 30]. Time-by-Example [30] followed the concept of Query-by-Example and employed hierarchical arrangement of subsequences of Aggregates-by-Example [20] and Summary-Table-by-Example [24] to serve historical relational databases.

3.3 Query by Graphical User Interfaces (GUIs)

Query-by-Filters Approach: As the graphical user interfaces (GUIs) were becoming more common, many graphical query interfaces were developed for temporal data. Several GUIs used the query-by-filters approach, where users specify filtering rules usually by using filter controls on the user interface, such as sliders or drop-down lists. Karam [18] presented a visualization called MtQ, which allowed users to explore temporal data and do simple search for events. Hibino and Rundensteiner [12, 13] proposed a visual query language and user interface for exploring temporal relationships using slider filters with results displayed in a graph-like visualization. PatternFinder [11] allowed users to specify the attributes of events and time spans to produce pattern queries that are difficult to express with other formalisms. LifeLines2 [35] used an alignment, ranking and filtering (ARF) framework to query for temporal categorical records. ActiviTree [34] provided a tree-like user interface with suggestions about interesting patterns to query for sequence of events. QueryMarvel [17] utilized and extended the semantic elements and rules of comic strips to construct queries.

Query-by-Example Approach: The definition of a query-by-example in this paper is the user interface that allows the users to draw what they expect to see as a result. The results from a query is a list of records, sorted by similarity to the given example. Kato et al. [1] implemented QVE that accepted a sketch drawn by users to retrieve similar images or time series from the database. IFQ (In Frame Query) [22] was a visual user interface that supported direct manipulation [26] allowing users to combine semantic expressions, conceptual definitions, sketch, and image examples to pose queries. Spatial-Query-by-Sketch allowed users to formulate a spatial query by drawing on a touch screen and translated this sketch into a symbolic representation that can be processed against a geographic database. Bonhomme et al. [6, 5] discussed the limitations of previous query-by-sketch approaches and extended the Lvis language, which was developed for spatial data, to temporal data. The new language used visual metaphors, such as balloons and anchors, to express spatial and temporal criteria. QuerySketch [36] allowed users to sketch a graph freehand, then view stocks whose price histories match the sketch. It used the Euclidean distance between sequences of monthly percentage price changes as the similarity score.

Hybrid Approach: Timesearcher [14] introduced timeboxes, which are rectangular widgets that can be used to specify query constraints. Users draw timeboxes on the timeline to query for all time series that pass through those timeboxes. Timesearcher is a hybrid approach because the users can draw an example (timeboxes) to specify the query, but the timeboxes are converted into filtering rules when processing the query.

3.4 Multi Criteria Decision Making

The extended M&M measure in this paper allows the users to specify the definition of similarity based on four decision criteria. Multi Attribute Decision Making (MADM), which is a subarea of Multi Criteria Decision Making (MCDM) [32], is about finding the best decision from finite set of alternatives for the given problem. There are many proposed alternatives to combine scores from multiple decision criteria according to the relative importance of each criterion.
The weighted sum model (WSM) was the earliest and probably most widely used method. The weighted product model (WPM) had been proposed in order to overcome some of its weaknesses. The analytic hierarchy process (AHP) was later proposed by Saaty [25] and became increasingly popular. However, AHP suffered from the rank reversal [3] problem. Some other widely used methods are the ELECTRE [4] and the TOPSIS [15] methods.

4 Description of the Interfaces

Because our evaluation compares features of query-by-filters interfaces (Lifelines2) and query-by-example (Similan), we describe both interfaces here. Our two designs have evolved in parallel with Similan inheriting several features from Lifelines2 (e.g. layout of the records, or alignment by event type) while we first describe Lifelines2, then the new Similan.

4.1 Query-by-Filters Interface: LifeLines2

In Lifelines2, each record is vertically stacked on an alternating background color and identified by its ID on the left. Events appear as triangle icons on the timeline, colored by their type (e.g. Admission or Exit). Placing the cursor over an event pops-up a tooltip providing more details. The control panel on the right side list filters and other controls. The visibility and color of each event category can be set in the control panel.

Users can select an event category to align all the records. For example, Figure 1 shows records aligned by the “Admit” event. When the alignment is performed, time is recomputed to be relative to the alignment event.

Users can apply the sequence filter to query records that contain a particular sequence, e.g. finding patients who were admitted, then transferred to a special room and exited. The first step is to select sequence filter from the “filter” by “drop-down list, then several drop-down lists that contain categories will appear. Users then set the values of the 1st, 2nd and 3rd drop-down lists to “Admit”, “Special” and “Exit”, respectively. The records that pass this filter will be selected and highlighted in yellow. A click on “Keep selected” removes the other records.

To query for records that have events occurring at particular intervals, users have to first display the distribution of selected events (with the distribution control) then select intervals on the distribution display. For example, to find patients who were admitted, then transferred to the ICU room on the first day of their stay and transferred to the intermediate ICU room on the fourth day, users have to align the records by “Admit”. The show the distribution of “ICU” using the “Show Distribution of” control, then select the 1st day on the distribution of “ICU” events at the bottom of the screen and click on “Keep selected” then show the distribution of “Intermediate” and draw a box from the 1st to the fourth day and “Keep selected”. (See Figure 1.) A similar process can be used for consecutive interval specification using different alignments and filtering.

LifeLines2 [35] (Figure 1) is a Java application, utilizing the Piccolo 2D graphics framework [2].

4.2 Query-by-Example Interface: Similan

The basic display of the records is similar to Lifelines2: each record is stacked on the main panel, events are colored triangle icons, and users can customize the visibility and colors of each event category. Users can also align all the records by a selected category (e.g. align by admission to the hospital in Figure 2).

To perform a query users first create or select a target record. For example, to find patients who were admitted, transferred to the ICU room on the first day and then to the intermediate room on the fourth day, users can start by aligning all records by “Admit”. Then users click on the edit button on the target panel to open a popup window, and drag and drop events on the empty timeline (i.e. They can select “Admit” from the list of categories shown in the popup and click on Add. The cursor will change into a magic wand and they can drop the event on the line). Figure 2 shows the patterns they created “Admit”, “ICU” and “Intermediate” at time 0, on the first day and fourth day, respectively. (See Figure 2.) Users can also select any existing record as a target by dragging that record from the main panel and dropping it into the target panel. This is useful for finding patients who exhibit a pattern of events similar to a particular known patient. A time range of interest can be drawn on the top of the timeline (See red line in Figure 2). In our example query, drawing a box from the time zero to the end of the fourth day will exclude all other events from the search. If no range is specified, the entire timeline will selected by default. The unit for time differences (e.g. hours or days) can be selected from a drop-down list. Event categories that should be excluded from the search can be unchecked in the control panel.

After clicking on Search, the records are sorted by their similarity score (with records with the highest scores on the top). Each record has a score indicator, a rectangle with four sections of different color (See Figure 2), inspired by ValueCharts [7], a visualization technique to support decision-makers in inspecting linear models. The length of a score indicator represents total score. It is divided into four colored parts which represent the four decision criteria. The length of each part corresponds to the weight * score. Placing a cursor over the score indicator brings up an explanation tooltip.

Users can see a detailed comparison of the target record and any other record by dragging that record into the comparison panel in the bottom. Lines are drawn between pairs of events matched by the M&M measure. Moving the cursor over a link displays a tooltip showing the event category, time of both events and distance.

By default, the search uses default weights, which means that all criteria are equally important. However, users may have different meanings for similarity in mind. Similan allows users to adjust the weight of all criteria in the “Weight” tab in the control panel. (See Figure 3.) The weight for each decision criterion can be adjusted with the slider controls, as well as the weight of each event category for each decision criteria. A click on “Apply Weight” refreshes the similarity measures and the order of the records on the display.
For example, if the task does not care about the value of time intervals (e.g., finding patients who were admitted, transferred to the special room and exited) the user can set a low weight for “Avoid Time Difference” to reduce its importance. Because the definition of weights can be complex, Similan includes sets of preset weight combinations for users to choose from. For instance, one preset is called “Sequence”, which uses a low weight for “Avoid Time Difference” and a high weight for “Avoid Missing Events”.

The new version of Similan described here is an Adobe Air Application using the Adobe Flex 3 Framework.

5 The extended M&M measure

Capturing the definitions of similarity is a challenging problem. This section explains the extended M&M measure that calculates the similarity score and how it can be customized.

5.1 Matching and Difference

We still use the same matching concept used in the original M&M measure [38]. This approach allows one-to-one matching and within the same category only. (See Figure 4.) However, this time we use a technique based on dynamic programming instead of the Hungarian algorithm [21] to match the events between the two records together. We split each record in to several lists, one list for each category. The problem of matching events between two records is reduced to matching events between two lists that contain events in the same category multiple times. We use dynamic programming to compute the matching between each pair of lists that produces the minimum time difference. Let \( n_A \) denote number of A’s in record\#1 and \( n_B \) denote number of A’s in record\#2. The time complexity of matching events between record\#1 and record\#2 is reduced from \( O((\max(n_A, m_A) + \max(n_B, m_B) + \ldots)^3) \) to \( O(n_A * m_A + n_B * m_B + \ldots) \). Furthermore, the original M&M measure considers only two types of difference: time difference and mismatches (missing or extra events). The extended M&M measure now considers four types of difference:

1. A match event is an event that occurs in both the target and the compared record. For example, in Figure 5, both the target and record\#2 have events A, B and C. The time difference (TD) is a sum of time differences within each pair of matched events. The time difference is kept separately for each event category. Users can specify what time unit they want to use for the time difference. For example, if the target’s B and record\#2’s B differ by 7 days but A and C occur on the same day, the time difference for B (TD\(_B\)) is 7 days while TD\(_A\) and TD\(_C\) are both 0 day.

Since the time difference may be not equally important for all categories, the total time difference (\( \sum TD \)) is a weighted sum of time difference from each category. Users can adjust what is important by setting these weights (\( \sum w^{TD} = 1 \)).

\[
\sum TD = w^A_T \times TD_A + w^B_T \times TD_B + \ldots
\]

(1)

2. A missing event is an event that occurs in a target record but does not occur in a compared record. The number of missing events (NM) is counted for each event category. For example, in Figure 5, the target has one B but record\#3 does not have any B, so the number of missing B (NM\(_B\)) is 1 while NM\(_A\) and NM\(_C\) are both 0.

For the same reason, the total number of missing events (\( \sum NM \)) is a weighted sum of number of missing events in each category (\( \sum w^{NM} = 1 \)).

3. An extra event is an event that does not occur in a target record but occurs in a compared record. The number of extra events (NE) is counted for each event category. For example, in Figure 5, the target has one C but record\#4 has 2 Cs, so the number of extra C (NE\(_C\)) is 1 while NE\(_A\) and NE\(_B\) are both 0. Like the previous two types of difference, the total number of extra events (\( \sum NE \)) is a weighted sum of number of extra events in each category (\( \sum w^{NE} = 1 \)).

4. A swapping event occurs when the order of the events is reversed. The number of swapping events (NS) is counted for each pair of event categories. For example, in Figure 5, the target has a C followed by B then C but record\#5 has A followed by B then C then B. If you draw a line from target’s C to record\#5’s C and do the same for B, it will create one crossing. So, the number of swapings between B and C (NS\(_B,C\)) is 1 while NS\(_A,B\) and NS\(_A,C\) are both 0. Again, the total number of swapping events (\( \sum NS \)) is a weighted sum of number of swapping events in each pair of event categories (\( \sum w^{NS} = 1 \)).

5.2 Four Decision Criteria

The 4 types of differences are normalized into a value ranging from 0.01 to 0.99 and called penalties. The total time difference (\( \sum TD \)), total number of missing events (\( \sum NM \)), number of missing events (\( \sum NM \)) and total number of swappings (\( \sum NS \)) are normalized into TD penalty, NM penalty, NE penalty and NS penalty, respectively. The 4 penalties are converted into these 4 decision criteria:

1. Avoid Time Difference (AT) = 1 – TD penalty
2. Avoid Missing Events (AM) = 1 – NM penalty
3. Avoid Extra Events (AE) = 1 – NE penalty
4. Avoid Swapping Events (AS) = 1 – NS penalty

5.3 Total Score

The total score is a weighted sum of the four decision criteria. The users can adjust the weights (\( w_{AT}, w_{AM}, w_{AE}, w_{AS} \)) to set the significance of each decision criteria (\( \sum w = 1 \)).

\[
T = w_{AT} \times AT + w_{AM} \times AM + w_{AE} \times AE + w_{AS} \times AS
\]

(2)

The total score (\( T \)) is from 0.01 to 0.99. The higher score represents higher similarity. We chose the weighted sum model to combine the score because of its simplicity and ease of presentation to users.
6 Controlled Experiment

We conducted a controlled experiment comparing 2 interfaces: LifeLines2, a query-by-filters interface, and Similan, a query-by-example interface. Our goal was not to prove a tool to be superior (as they are clearly at different stages of refinement and represent different design concepts), but to understand which query method was best suited for different tasks. Another goal was to observe the difficulties that users encountered while using the tools to perform given tasks. Both systems were simplified by hiding certain controls to focus on the query features we wanted to compare.

- **Independent Variable**: Interface type: query-by-filters and query-by-example
- **Dependent Variables**: The time to complete each task, errors, and subjective ratings on a 7-points Likert-type scale. We use paired t-test to analyze the results.
- **Controlled Variables**: Computer, mouse and window size. We used equivalent datasets for each interface.

6.1 Experimental Procedure

We used a 2x5 (2 interfaces by 5 tasks) repeated measure within subject design. Eighteen participants each received $20 for their 90-minute participation. To provide the motivation to perform the tasks quickly and accurately, an additional $5 was promised to the fastest user with the least errors of each interface. Because our target users are researchers and data analysts we chose computer science students who are assumed to have high level of comfort with computers but no knowledge of this particular interface.

Participants were given 10 minutes of training with each interface. Next the participants were asked to perform 5 tasks using that interface. After that, the experimenter followed the same procedure for the second interface. To control learning effects, the presentation of the LifeLines2 and Similan interfaces was counterbalanced. At the end of the experiment, the participants were asked to complete a 7-point Likert-scale questionnaire with 7 as the best.

We used a modified version of the deidentified patient transfer data provided by our partners. The data contained information about when patients were admitted (Admit), transferred to Intensive Care Unit (ICU), transferred to Intermediate Care Unit (Intermediate), transferred to a normal room (Floor), and exited (Exit).

6.2 Tasks

The tasks were based on real scenarios provided by physicians. Participants were requested to find patients in the database, who satisfy the given description. To avoid the effect of alignment choice, all tasks were provided with an obvious sentinel event (e.g. Admit). We considered these factors when designing the tasks:

1) **Target type**: Either a sequence description was provided or a existing record was to be used as a target.
2) **Time constraint**: Present or not
3) **Uncertainty**: Yes or No, e.g. the number of events may be precise or not, the time constraint may be flexible or not.

**Task type 1** Description without time constraint, no uncertainty

1.1: “Find at least one patient who was admitted, transferred to Floor then to ICU.”
1.2: “Count all patients who fit task 1.1 description”

Task 1.1 was designed to observe how quickly the participants can use the interface to specify the query while task 1.2 focused on result interpretation and counting.

**Task type 2** Description with time constraints, no uncertainty

2.1: “Find at least one patient who was admitted and transferred to Intermediate on the second day then to ICU on the third day.”
2.2: “Count all patients who passed task 2.1 description.”

**Task type 3** Description with uncertainty, without time constraint

3.1: “Find a patient who best matches the following conditions: Admitted and then transferred to special room approximately 2 times and transferred to ICU room after that. If you cannot find any patient with exactly 2 transfers to the special room, 1-3 transfers are acceptable.”
3.2: “Count all patients who passed task 3.1 description.”

**Task type 4** Description with uncertainty and time constraint

“Find a patient who best matches the following conditions: Admitted, transferred to Floor on the first day, ICU approximately at the end of the third day. The best answer is the patient who was transferred to ICU closest to the given time as possible.”

**Task type 5** Existing record provided as target:

“Find a patient who was transferred with the most similar pattern with patient no. xx during the first 72 hours after being admitted. Having everything the same is the best but extra events are acceptable.”

6.3 Hypotheses and Results

The experimental results are illustrated in Table 1. In this section we described the hypotheses and results in more details. We used “F” to denote query-by-filters (LifeLines2) and “E” to denote query-by-example (Similan).

In Task 1.1, the results supported our hypothesis that the users could perform faster with query-by-filters because the query could be specified directly with drop-down menus, while they would spend more time to plot the events on the timeline and possibly adjust the weight. The qualitative assessment confirmed that the participants preferred to use query-by-filters to specify the query with sequence only ($p < 0.001, F:6.9, E:5.5$ on 7-point Likert scale).

In Task 2.1, we thought participants would perform better with query-by-example because they may specify more filters than necessary (use both sequence and distribution filters) and may be overwhelmed by the many steps required by the query-by-filters. Even though we had emphasized in the training that they could use the distribution filter only to perform this kind of task, six participants still used both filters, suggesting that the richness of filters available may have some drawbacks. Nevertheless, the participants still performed faster with query-by-filters. However, in the questionnaire results, we found that the participants preferred to use query-by-example ($p < 0.001, F:4.9, E:6.0$). Some participants expressed that it felt more “natural” and did not require as many steps. They did not seem aware that they actually performed faster with query-by-filters although it required more steps.

In Task 3.1, we thought that participants would perform the tasks faster with the query-by-example because this task has uncertainty. They have to specify the query multiple times when using query-by-filters. However, no difference in average time was found but the participants thought that it was easier to specify the query with uncertainty with query-by-example ($p < 0.001, F:4.1, E:5.8$).

For the counting tasks (1.2, 2.2, 3.2), we hypothesized that the users could perform faster with query-by-filters because they receive only the records that passed the conditions. On the other hand, the participants may spend time adjusting the weights and interpreting the results when using query-by-example because it returns all records and there is no clear cut off point. The experimental results confirmed our hypothesis.

In Task 4, we thought that participants would perform better with query-by-example because it allows uncertainty and will bring the best match to the top. Specifying the query using query-by-filters is possible but require many more steps and some strategizing. The users also have to compare the time difference manually. As expected the participants performed faster with query-by-example.

In Task 5, the participants performed faster with query-by-example as we had expected. Two participants answered incorrectly with query-by-filters. The questionnaire results supported that the participants preferred query-by-example ($p < 0.001, F:3.9, E:6.8$).
When the participants were asked about confidence of the answers for finding at least one patient tasks, there was no significant difference. However, when asked about the confidence of their answers for finding all patients (i.e. counting tasks), query-by-filters was rated significantly better ($p < 0.001$, F:6.7, E:4.8).

6.4 Discussion

Our experiment showed that query-by-filters had advantages in showing exact results, which gave more confidence to the users, especially in tasks that involve counting. It allows users to find a simple sequence without time constraint or uncertainty faster than with query-by-example. However, users felt that it was more complex to use (probably because it required many steps). Users commented that query-by-filters needed a better way to keep track of what has been done to the data. A list of previous actions and an easy way to undo were features requested by the participants.

On the other hand, query-by-example advantages were in the flexibility and intuitiveness of specifying the query, especially in tasks with time constraint or uncertainty. Users commented that they could see the big picture of their query. However, query-by-example suffered from tasks that involve counting. The participants requested a better way to support counting tasks. They did not seem confused by the concept of the weights but spent a fair amount of time learning how to adjust them properly. Some mentioned that once they got used to it, they would perform faster. They found the weight presets to be useful and they asked to have more presets available. However, two participants noticed that query-by-example responded slightly slower than query-by-filters, which is correct. This is due to the time complexity of the similarity measure. Also, sometimes the participants did not set the time range when they should have, which reduced the accuracy of sort order.

7 Next Steps: Guidelines for a Hybrid Interface

The query-by-filters and query-by-example approaches each have their advantages. How can we combine the best features from these two approaches without adding unneeded complexity? Based on the results of the experiment and our observations during the longitudinal study with our partners we propose the following guidelines for future event sequence query interfaces:

1. Draw an example. Specifying the query by dropping events on a blank timeline, seems closer to what the users are thinking about and the example also helps them compare results with their query. The visual representation of the query also seems to help users notice and correct errors.

2. Sort results by similarity to the query but do not return all records and allow users to see more if necessary instead. Showing all records even those that do not fit the query at all confuses users and reduces confidence in the results. However, users may want to see more at certain times. One possible strategy is to show only exact results first (i.e. like query-by-filters) and have “more” button to show the rest or the next $n$ records. This may increase confidence and still be useful for exploratory search.

3. Allow users to specify what is flexible and what is not. Sometimes there are certain rules that users want to apply, e.g. must be admitted to the ICU (i.e. do not even bother showing me records with no ICU event). Consider setting the scope of flexibility for each event. This may be specified in form of annotations or error bars on the drawn example.

4. Weight presets are useful. Although participants did not have problem with the concept of similarity measure, they still spent considerable time when have to adjust the weight. We can provide presets of common definitions of similarity.

5. Do not provide too many alternative ways to perform the same task. This can lead to confusion. In the experiment, we found many users used more filters than necessary.

8 Future Work

The extended M&M measure uses an approach similar to the analytic hierarchy process to combine the four decision criteria. This approach has a problem called the rank reversal problem, which occurs when a new record $r$ is added such that $r$ is the same as an existing record in the initial data set. Then the evaluations on the new data set are not consistent with evaluations on the initial data set. A more sophisticated model can be applied but may make it harder to visualize the scores. The speed of the similarity measure may be improved to increase responsiveness.

The user interface for adjusting the weights can be improved. Asahi et al. [1] used treemaps to assist weight adjustment for the analytic hierarchy process. Moreover, the analytical hierarchy process can calculate the proper weight for the users by asking for pairwise comparison between each pair of decision criteria. A wizard that asks for pairwise comparison between each pair of decision criteria may be used.

9 Conclusion

Event sequence records are continuously being gathered by various organizations. Querying for event sequences to answer questions or look for patterns is a very common and important activity. Most existing temporal query GUIs use a query-by-filters approach, which return only records that match the query. However, in exploratory search, the users are usually uncertain about what they are looking for. Too narrow queries may ignore the results which are on the borderline of the filtering rules. On the other hand, the query-by-example approach allows users to sketch an example of what they are seeking and find similar results, which provides more flexibility.

This paper makes these contributions: First, we presented Similan, an interface to specify an event sequence example and search
for records that are similar to the example, following the query-by-example approach. Second, we improved and extended the M&K measure to be customizable by four decision criteria, increasing its performance and flexibility. Third, we conducted a controlled experiment that assessed the benefits of query-by-filters and query-by-example interfaces for different tasks, leading to guidelines for improved sequence query interfaces that combine the benefits of both approaches.

While the paper focused on medical examples, the application of this work is not limited to the medical domain. The design principles are based on event sequence data, therefore make this work applicable to a wide variety of event sequences: transportation incidents logs, student records, web logs, criminal investigations, financial histories, and many more topics.

ACKNOWLEDGEMENTS

We appreciate collaboration from our physician partners at the Washington Hospital Center, especially Dr. Phuong Ho, Dr. Mark Smith and David Roseman, and would like to thank Dr. Vibha Saza-wal, Dr. Jen Golbeck, Taowei David Wang and Sureyya Tarkan for their thoughtful comments, and all participants in the studies for their participations.

REFERENCES


