Frequence: Interactive Mining and Visualization of Temporal Frequent Event Sequences

Adam Perer
IBM T.J. Watson Research Center
Yorktown Heights, New York, USA
adam.perer@us.ibm.com

Fei Wang
IBM T.J. Watson Research Center
Yorktown Heights, New York, USA
fwang@us.ibm.com

ABSTRACT

Extracting insights from temporal event sequences is an important challenge. In particular, mining frequent patterns from event sequences is a desired capability for many domains. However, most techniques for mining frequent patterns are ineffective for real-world data that may be low-resolution, concurrent, or feature many types of events, or the algorithms may produce results too complex to interpret. To address these challenges, we propose Frequence, an intelligent user interface that integrates data mining and visualization in an interactive hierarchical information exploration system for finding frequent patterns from longitudinal event sequences. Frequence features a novel frequent sequence mining algorithm to handle multiple levels-of-detail, temporal context, concurrency, and outcome analysis. Frequence also features a visual interface designed to support insights, and support exploration of patterns of the level-of-detail relevant to users. Frequence's effectiveness is demonstrated with two use cases: medical research mining event sequences from clinical records to understand the progression of a disease, and social network research using frequent sequences from Foursquare to understand the mobility of people in an urban environment.

Author Keywords

Visual Analytics, Frequent Sequence Mining, Temporal Visualization

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION

One of the challenges of the Big Data era is to leverage the voluminous data that is being captured to drive decision making and insights. Common to such data are temporal events, data points with both a timestamp and event type, so understanding patterns of temporal event sequences is an important problem to many. For example, medical researchers wish to leverage the data captured by electronic health records to determine if certain sequences of medical events correlate with positive outcomes. Similarly, city government officials wish to leverage the temporal data collected from their transportation systems, call centers, and law enforcement agencies to improve their cities' services.

IUI'14, February 24–27, 2014, Haifa, Israel.

ACM 978-1-4503-2184-6/14/02..$15.00.
http://dx.doi.org/10.1145/2557500.2557508

However, despite the availability of temporal data and the common desire to extract knowledge, mining patterns from temporal event sequences is still a fundamental challenge in data mining [7]. Frequent Sequence Mining (FSM) techniques have emerged in the data mining community to find sets of frequently occurring subsequences. However, these algorithms often have constraints that limit its applicability to real-world data:

- **Level of Detail.** Temporal events are often recorded at a specific level-of-detail to record maximum information about an event's type. FSM techniques applied to data with a large dictionary of event types will often suffer from computational complexity. Perhaps even more of a fundamental problem is that patterns extracted from a specific level-of-detail may impair an interpretable overview of patterns for users. In order for mining techniques to be technically feasible and usable, we propose multiple levels-of-detail should be available to users.
- **Temporal Context.** Many FSM techniques ignore the temporal context associated with data, and instead focus on the pure sequentiality of events. However, for certain real-world scenarios, if a certain amount of time elasped between events, the events should not be considered as part of the same sequence, even if events are technically sequential in the event log. We propose mining techniques need to take into consideration the temporal context of users.
- **Concurrency.** Many FSM algorithms suffer from pattern explosion when there are many concurrent events. This is particularly troubling for real-world data, as many systems may record data in low-resolution precision, such as a day, and many events may occur on the same day. For other datasets, even when there is extreme temporal precision to data (e.g. millisecond precision), the exact order of events may be irrelevant, so if they occur within a domain-relevant time window, they should be treated as concurrent. We propose mining techniques need to handle concurrency.
- **Outcome.** Many FSM algorithms are agnostic to the types of patterns mined. However, in real-world data, analysts may not just need a list of patterns but instead how each of the patterns correlate to an outcome measure. We propose mining techniques need to support outcome analysis.

In this paper, we enhance a fast and popular FSM algorithm to handle these real-world needs to serve actionable insights.

But to support each need properly, the parameters should be flexible to derive from user hypotheses or evolve during an iterative data exploration process. In order to support flexible decision making, we present *Frequence*, a intelligent user interface designed to reach actionable insights by integrating visualization with interactive mining algorithms.

We demonstrate Frequence's utility in two real-world use cases: a medical informatics researcher using frequent sequences in electronic medical records to understand the progression of a disease among patients, and a social network researcher using frequent sequences from a location-based social network to understand the relationship between online popularity and offline mobility throughout urban environments.

Concretely, Frequence's contributions include both **a novel algorithm** for Frequent Sequence Mining whose design handles real-world constraints of level-of-detail, temporal context, concurrency, and outcome, and **a novel intelligent user interface** that integrates mining and visualization to support interactive parameterization and exploration to reach insights.

This paper begins with related work, providing an overview of sequence mining and visualization techniques. Next, an overview of the design of Frequence is described, as well as the iterative workflow it supports. Then, Frequence's sequence mining algorithms are described as well as its novel enhancements. Next, Frequence's visual interface is described in terms of its visual encoding and interactive UI to parameterize the analytics. Then, use cases are described to demonstrate Frequence's utility on real-world data. Finally, this paper concludes with a discussion of future work.

RELATED WORK

There has been little work that integrates techniques for mining and visualizing frequent event sequences, so we present prior work on mining and visualization separately.

Mining Frequent Event Sequences

Frequent Sequence Mining (FSM) is a popular technique for finding sets of frequently occurring subsequences from a larger set of temporal event sequences. It is challenging since one may need to examine a combinatorially explosive number of possible frequent subsequences.

FSM has been researched for a long time and many approaches have been proposed. Initially, most of the approaches (e.g., Generalized Sequential Pattern miner (GSP) [1]) are *A priori*-like [1], which utilizes the fact that any super-pattern of a non-frequent pattern cannot be frequent. These approaches start by constructing a frequent pattern set (the appearance frequencies of the patterns included are above a certain percentage threshold) containing only one event, then they grow those patterns pass by pass, with each pass appending one single event to the detected patterns from last pass, until no frequent patterns can be found. This type of approach can be computationally expensive due to the huge candidate sequence set and multiple database scans. Many strategies have been proposed to make FSM more efficient and practical. For example, *PrefixSpan* (Prefix-projected Sequential PAtterN mining) [8] explores prefix projection in FSM to reduce the efforts of candidate subsequence generation. *SPADE* (Sequential PAttern Discovery using Equivalence classes) [17] utilizes combinatorial properties to decompose the original problem into smaller sub-problems, that can be independently solved in main-memory using efficient lattice search techniques, which greatly reduces the number of database scans. *SPAM* (Sequential Pattern Mining) [3] designs a smart bitmap based representation for those event sequences, so that all event sequences become 0-1 strings, and all operations needed for FSM will become bitwise AND/OR operations. This makes the FSM procedure much more efficient. However, despite these improvements, there are still some difficulties for applying such algorithms directly to real world data, especially when there are concurrent events or temporal constraints. Frequence's FSM algorithm enhances SPAM to support these user needs.

Temporal Event Sequence Visualizations

There has been a great body of work in the visualization community designing techniques for temporal event sequences. A common approach is to place records on a series of horizontal timelines to show multiple records in parallel [2, 9] and support interactive search [6, 13, 16], filtering [13], and clustering [4]. Recently, LifeFlow [15] introduced a way to aggregate multiple event sequences into a tree, and Outflow [14] later designed a way to aggregate events into a graph, as well as integrating statistics.

The visual design of Frequence's visualization is similar to Sankey [10] and alluvial [11] diagrams. However, these diagrams focus on the flow of resources, as Sankey diagrams ignore sequential ordering whereas alluvial diagrams were designed to show how network structures change over time. Outflow also looks visually similar to our approach, however, Outflow aggregates subsequences and outcomes [14]. In Frequence, each subsequence is represented as an individual edge to provide an overview of all sequences and their individual outcomes and support, as we support user tasks to find meaningful individual sequences. Furthermore, only Frequence supports navigation by level-of-detail which is a novel contribution to prior approaches.

SYSTEM DESIGN AND WORKFLOW

Frequence's intelligent user interface is designed to put users at the center of both analytics and visualization. In line with many advanced visual interfaces, Frequence supports the common browsing and searching strategy of the Visual Information Seeking Mantra, which suggests to support *Overview first, Zoom and Filter, then Details-on-Demand* [12]. When users begin their exploration, as illustrated in Figure 6, they are first shown frequent patterns at a coarse level-of-detail to give them an overview. If they discover an pattern of interest, they can select the pattern and then zoom and filter to see the underlying patterns at a finer level-of-detail. Whenever users want more information about a particular pattern, at any level-of-detail, they can select it to get details-on-demand, which include statistics about the pattern's frequency, outcome, and the events composed within.

However, parallel to browsing and searching, Frequence also supports interaction with the analytics to support a refinement strategy, where users can adapt the mining algorithm based upon their data-driven hypotheses to generate more meaningful patterns to explore. By interactively refining parameters of the mining algorithm, such as temporal context, outcome measures, and other temporal constraints, users can be empowered to make sure they are reaching meaningful results.

In the next section, we explain the novel frequent pattern mining algorithms necessary to support this iterative workflow.

MINING FREQUENT PATTERNS

Frequence's frequent pattern mining algorithm is based on the SPAM algorithm [3], with several enhancements. In order to understand the enhancements, we first provide an overview of the SPAM algorithm.

The goal of Frequent Sequence Mining (FSM) is to mine *frequent* subsequences from a set of event sequences. Formally, event sequences and subsequences are defined as follows:

Definition 1. An event sequence $\boldsymbol{\theta} = \langle \theta_1, \theta_2, \cdots, \theta_m \rangle (\theta_i \in \mathcal{D})$ is an ordered list of events, where $\mathcal{D}$ is the event dictionary and θ_i happened no later than θ_{i+1}.

Definition 2. A sequence $\boldsymbol{\tau} = \langle \tau_1, \tau_2, \cdots, \tau_{m_\tau} \rangle$ is said to be a subsequence of another sequence $\boldsymbol{\theta} = \langle \theta_1, \theta_2, \cdots, \theta_{m_\theta} \rangle$, denoted by $\boldsymbol{\tau} \subseteq \boldsymbol{\theta}$ if $\exists\, i_1, i_2, \cdots, i_m$ such that $1 \leqslant i_1 \leqslant i_2 \leqslant \cdots \leqslant i_m \leqslant m_\theta$ and $\tau_1 = \theta_{i_1}, \tau_2 = \theta_{i_2}, \cdots, \tau_{m_\tau} = \theta_{i_m}$.

Support is the measure that determines whether a subsequence is frequent or not:

Definition 3. Given a set of event sequences $\mathcal{S} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_n\}$, the support of a sequence $\boldsymbol{\tau}$ is the percentage of the sequences in $\mathcal{S}$ which have $\boldsymbol{\tau}$ as a subsequence.

With the above definition, we say a subsequence (or temporal pattern) is *frequent* for sequence set $\mathcal{S}$ if its appearance percentage in $\mathcal{S}$ is above a certain pre-specified support value.

As described in our overview of related work, most FSM algorithms adopt a *pattern growing* strategy, which is illustrated in Figure 1. Initially, the algorithms start with an empty frequent pattern set $\mathcal{F} = \{\}$, then each single event in the event dictionary $\mathcal{D}$ is checked, and the events that are frequent are added to $\mathcal{F}$. These patterns are referred to as frequent patterns of *length* 1, as the length of a pattern is defined as the number of events it contains. The pattern can be grown with two types of extensions: an *S-extension* and an *I-extension* [3]:

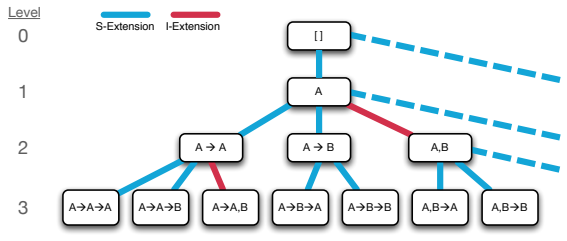

Figure 1. A graphical illustration of the pattern growing procedure.

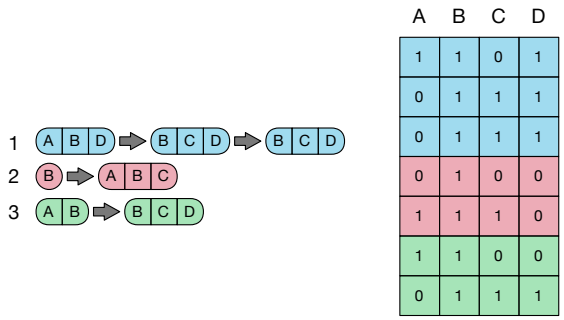

Figure 2. An example of the bitmap representation of the sequences, where there are four different events and three sequences.

Definition 4. An S-extension of a sequence $\boldsymbol{\tau} = \langle \tau_1, \tau_2, \cdots, \tau_{m_\tau} \rangle$ with event θ, denoted by $\boldsymbol{\tau} \rightarrow \theta$, is to append θ at the end of $\boldsymbol{\tau}$ so that θ happens after τ_{m_τ}.

Definition 5. An I-extension of a sequence $\boldsymbol{\tau} = \langle \tau_1, \tau_2, \cdots, \tau_{m_\tau} \rangle$ with event θ, denoted by $\boldsymbol{\tau}, \theta$, is to append θ at the end of $\boldsymbol{\tau}$ so that θ and τ_{m_τ} happen concurrently.

After a pattern growing stage, all candidate patterns of length 2 are obtained and the frequent ones are added to $\mathcal{F}$. Then the algorithm grows each pattern of length 2 via an S-extension and an I-extension to get the frequent patterns of length 3, and the pattern growing procedure repeats until no more frequent patterns are found. If one considers the complete pattern growing process as a tree (see Figure 1), then the procedure can be seen as a *breadth-first search* on that tree, which checks the tree level-by-level. A *depth-first search* strategy has also been proposed [3], which checks the tree branch by branch, and can be a more efficient recursion procedure for long sequences.

SPAM is able to be computationally efficient by using a *bitmap representation* for event sequences [3], which is illustrated in Figure 2. In this representation, a sequence $\boldsymbol{\theta}$ is represented as a $|\boldsymbol{\theta}| \times |\mathcal{D}|$ bitmap $\mathcal{B}(\boldsymbol{\theta})$, where $|\boldsymbol{\theta}|$ is the length of $\boldsymbol{\theta}$ and $|\mathcal{D}|$ is the size of $\mathcal{D}$, such that $\mathcal{B}(\boldsymbol{\theta})(i, j) = 1$ if the j-th event in $\mathcal{D}$ happened at the i-th timestamp of $\boldsymbol{\theta}$, otherwise $\mathcal{B}(\boldsymbol{\theta})(i, j) = 0$. With this representation, both S-extensions and I-extensions can be performed with a bitwise AND operation. Figure 3 shows how to extend event A with event B for the example in Figure 2. For an S-extension, the first appearance of A in each sequence is detected, so the corresponding bit value is changed to 0 and all the following bits to 1 (denoted as $T(A)$), and then an AND-operation is executed between the bitmap vectors of $T(A)$ and B. For an I-extension, an AND-operation is directly executed between the bitmap vectors of A and B. After the extension, the value of 1 will exist in each sequence if the extended pattern exists in those sequences. Therefore, the appearance frequency of the extended pattern is simply the number of sequences that have a value of 1.

Temporal Context

Although SPAM uses a smart bitmap representation to make the FSM procedure efficient, it does not take into consideration the temporal context of detected patterns. This is problematic for real-world applications, as patterns lasting years

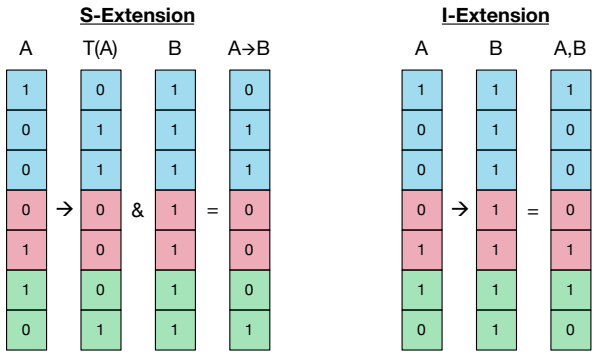

Figure 3. An example of S-extension and I-extension of event A with event B for the example sequences in Figure 2.

and other patterns lasting days may have completely different meanings in different contexts, but SPAM would treat them equivalently.

In order to make SPAM capable of detecting temporal patterns within a temporal context (that is, within the threshold of a user-defined duration), we propose the following enhancement: Suppose that there is an explicit timestamp associated with every event in every sequence, then when a candidate pattern appears in a specific sequence, the pattern must not only appear, but also its duration in the sequence should be below the threshold. Specifically, we define the pattern duration in a sequence as:

Definition 6. Let $\boldsymbol{\tau} = \langle \tau_1, \tau_2, \cdots, \tau_{m_\tau} \rangle$ be a candidate pattern of sequence $\boldsymbol{\theta} = \langle \theta_1, \theta_2, \cdots, \theta_{m_\theta} \rangle$, where each event θ_i is associated with timestamp t_i, and $(i_1, i_2, \cdots, i_m)$ be a set of indices of $\boldsymbol{\theta}$ satisfying $1 \leqslant i_1 \leqslant i_2 \leqslant \cdots \leqslant i_m \leqslant m_\theta$ and $\tau_1 = \theta_{i_1}, \tau_2 = \theta_{i_2}, \cdots, \tau_{m_\tau} = \theta_{i_m}$, then the duration of $\boldsymbol{\tau}$ in $\boldsymbol{\theta}$ is $min_{(i_1,i_2,\cdots,i_m)}\,(t_{i_m} - t_{i_1})$.

Algorithm 1 summarizes the our enhanced SPAM approach, which we call *Time-Aware SPAM*. Note that in step 3 and step 8 of the **Pattern_Grow** subroutine, when checking the appearance frequencies of $\boldsymbol{\tau}_S/\boldsymbol{\tau}_I$, there is a requirement that patterns should both appear in the counted sequence and their duration should be shorter than L.

Algorithm 1 Time-Aware SPAM

Require: Sequence set $\mathcal{S} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_n\}$, Support value α, Pattern duration L

1. Construct bitmap representations for all sequences in $\mathcal{S}$
2. Set the output frequent pattern set $\mathcal{F} = \{\}$
3. Count all the single event frequencies, and add the ones whose frequencies above α to $\mathcal{F}$
4. **For** every event $\theta \in \mathcal{F}$
5. Do **Pattern_Grow**($\theta, \mathcal{F}, \mathcal{F}, L$)
6. Output $\mathcal{F}$

Pattern_Grow($\boldsymbol{\tau}, \mathcal{S}_n, \mathcal{I}_n, L$)

1. $\mathcal{S}_t = \{\}, \mathcal{I}_t = \{\}$
2. **For** every $\theta \in \mathcal{S}_n$
3. **if** $\boldsymbol{\tau}_S$=S_extension($\boldsymbol{\tau}, \theta$) is frequent w.r.t. L
4. $\mathcal{S}_t = \mathcal{S}_t \cup \theta$, add $\boldsymbol{\tau}_S$ to output pattern set
5. **For** every $\theta \in \mathcal{S}_t$
6. **Pattern_Grow**($\boldsymbol{\tau}_S, \mathcal{S}_t, \mathcal{S}_t \succeq \theta$)
7. **For** every $\theta \in \mathcal{I}_n$
8. **if** $\boldsymbol{\tau}_I$=I_extension($\boldsymbol{\tau}, \theta$) is frequent w.r.t. L
9. $\mathcal{I}_t = \mathcal{I}_t \cup \theta$, , add $\boldsymbol{\tau}_I$ to output pattern set
10. **For** every $\theta \in \mathcal{I}_t$
11. **Pattern_Grow**($\boldsymbol{\tau}_I, \mathcal{S}_t, \mathcal{I}_t \succeq \theta$)

Concurrency

A bottleneck for Time-Aware SPAM occurs when events happen at the same time, as this greatly increases the search tree size in the pattern growing procedure. This is particularly problematic as concurrent events are often common in real-world applications. For example, the finest resolution of temporal data in many Electronic Health Records systems is a *day*, and during a day, multiple medical events may occur to a patient (e.g. a patient often gets multiple lab tests during a single lab visit). For other datasets, even when there is extreme temporal precision to data (e.g. millisecond precision), the analyst may not be concerned about the exact order of events, as long as they occurred within a domain-relevant time window.

To alleviate this problem, we pre-process event sequences prior to Time-Aware SPAM to contain pattern explosion. As there can be many *Concurrent Event Sets* (CESs) contained in event sequences, the first step is to detect the frequent *Event Packages* (EPs) that are frequent subsets of CESs. If we treat each CES in every event sequence as a *transaction*, then the problem of detecting EPs is equivalent to the problem of *frequent item set mining* [1], and each detected EP can be used as a *super event*. Then, a greedy approach is applied based on *Two-Way Sorting* to break down each CES as a combination of regular and super events, such that the number of events contained in each CES is greatly reduced.

To better explain the process of breaking down CESs, we provide the following example: Suppose there exists a CES `ABCDE` that needs to be broken down using the detected EPs (shown in the center of Figure 4). The algorithm then sorts the packages according to the two-way sorting strategy as shown in Figure 4 – that is, the EPs are first sorted according to their cardinalities. Then, for packages with the same cardinality, they are sorted with respect to their appearance frequency. To breakdown `ABCDE`, the algorithm first finds the *longest* event packages that are subsets. In this case, `ABC` and `ACE` are the longest packages which are subsets of `ABCDE`. Then, because `ABC` occurs more frequently than `ACE`, `ABC` is selected as a super event contained in `ABCDE`. Besides `ABC`, the rest of the events are `DE`. Then the procedure is applied again to break down `ABCDE` as `ABC`,`D`, `E`. Using this technique, there are only 3 super events in `ABCDE` after the break-down procedure. The full details of the algorithm are described in Algorithm 2.

Level of Detail

In real-world data sets, temporal events are often recorded at a very specific level-of-detail to retain maximum information about an event. However, applying many FSM techniques (e.g. [17, 3, 8]) or Time-Aware SPAM to data with a large

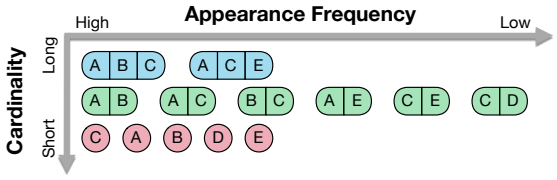

Figure 4. A graphical illustration of how the two-way sorting procedure works. We first sort the mined event packages according to their cardinalities, and then for the packages with the same cardinality, we sort them according to their frequencies.

Algorithm 2 Breaking Down a CES

Require: An CES $\mathcal{S}$ to be broken down into frequent Event Packages (EPs).

1: Sort the detected EPs into buckets according to their cardinalities (number of events contained), such that the packages within the same bucket have the same cardinality.
2: Sort the packages within the same bucket with their appearance frequencies in the patient traces.
3: $\mathcal{O} = \emptyset$
4: **for** Every bucket $\mathcal{B}$ **do**
5: **if** `length`($\mathcal{B}$) <`length`($\mathcal{S}$) **then**
6: **for** Every EP $\mathcal{E}$ in $\mathcal{B}$ **do**
7: **if** $\mathcal{E}$ is a subset of $\mathcal{S}$ **then**
8: Add $\mathcal{E}$ to $\mathcal{O}$, Set $\mathcal{S} = \mathcal{S} \backslash \mathcal{E}$
9: **if** $\mathcal{S} == \emptyset$ **then**
10: **Return** $\mathcal{O}$
11: **else**
12: Return to Step 4
13: **end if**
14: **end if**
15: **end for**
16: **end if**
17: **end for**

dictionary of event types will often suffer from computational complexity. Technical complexity aside, another fundamental problem is patterns extracted from a fine levels-of-detail may impair gaining an interpretable overview from the results. In order for mining to be technically feasible and usable, Frequence is able to mine data on multiple levels-of-detail.

For each level-of-detail available, Frequence runs its mining algorithm on event sequences using the event dictionary available for each level-of-detail. Typically, levels-of-detail are hierarchical and thus the coarse level will likely have a small event dictionary. In this case, special techniques for handling concurrency or collapsing event sets may not be required to remain performant. However, for finer levels-of-detail, large data sets may require such features to be computationally feasible. That said, in practice, if an overview of coarse patterns are shown first, a coarse pattern of interest may be selected to focus on, and then Frequence only needs to mine from the cohort that matched the coarse pattern, which drastically reduces the number of sequences and event types. Frequence's iterative workflow was designed to support this type of exploration.

Outcome

Once a set of frequent patterns has been identified, correlations between the mined patterns and the event sequences' outcome measure can be identified. For example, in a medical informatics scenario, analysts may be curious how certain patterns correlate with discrete outcome variables (e.g., deceased vs. alive) or continuous ones (e.g., blood pressure measurements).

To enable outcome analysis, a *Bag-of-Pattern* (BoP) representation is constructed for each event sequence. Formally, given a set of n frequent patterns, the BoP representation is an n-dimensional vector, where the i-th element of that vector stores the frequency of the i-th pattern within the corresponding event sequence. If there are m event sequences, then we construct an $m \times n$ sequence-pattern matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]$ whose (j, i)-th element indicates the number of times the i-th pattern appeared in the j-th event sequence. Thus, the i-th column $\mathbf{x}_j$ summarizes the frequency of the i-th pattern in all m sequences. We can also construct an m-dimensional sequence outcome vector $\mathbf{y}$, such that y_j is the outcome of the j-th event sequence. For example, with a binary outcome, $y_i \in \{+1, -1\}$, +1 represents a positive outcome whereas -1 represents a negative outcome. Given this formulation, statistics are then computed to measure the correlation between each $\mathbf{x}_i$ and $\mathbf{y}$ to measure the informativeness of the i-th pattern in terms of predicting a sequence's outcome. Frequence has implemented a variety of statistical measures including Pearson correlation, P-value, information gain, and odds ratio.

FREQUENCE'S VISUAL USER INTERFACE

Frequence is a web-based interface that features a visualization panel in the center of the interface to navigate the visualization, as well as a panel on the left side of the interface to control the interactive analytics.

Interactive Visualization

Frequence mines sequential knowledge temporal from event sequences so analysts can gain insights. However, the quantity of patterns discovered is often too large for users to make sense of them. The goal of Frequence is not only to mine the patterns but also to present the data in a user-centric way so that the patterns mined can be utilized in real-world settings. Information visualization is an effective way of communicating complex data, and thus the key part of the Frequence UI is a flow visualization.

We describe the characteristics of our visualization using Figure 5 as an illustrative example. In this example, the patterns are sequences of places that users have traveled, and each user has an outcome measure of popularity. This type of data will be subsequentially described in more detail as a use-case.

Events in the frequent sequences are represented as nodes, and event nodes that belong to the same sequence are connected by edges. The nodes and edges are positioned using a modified Sankey diagram layout [10]. However, while Sankey layouts aggregate common subsequences into a single edge, such an aggregation in Frequence would make it impossible to reach insights about individual patterns.

Temporal Event Sequence	Outcome	Support
Arts & Entertainment→Food→Travel & Transport	Unpopular	0.25
Arts & Entertainment→Food→Nightlife Spot	Popular	0.50
Shop & Service→Food→Nightlife Spot	Popular	0.50
Food→Outdoors & Recreation	Neutral	0.25

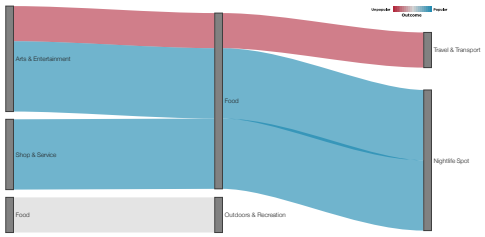

Figure 5. An illustrative example of Frequence's visual encoding for a set of frequent patterns. Patterns are represented by a sequence of nodes (events) connected by edges (event subsequences). Patterns are colored according to their correlation with users' outcomes.

Thus, in Frequence, subsequences are represented as individual edges. For instance, the simple pattern `Food` → `Outdoors&Recreation`, is visualized as a `Food` node connected to a `Outdoors&Recreation` node, as shown at the bottom of Figure 5. Patterns that share similar subsequences, such as `Arts&Entertainment` → `Food` → `NightlifeSpot` and `Arts&Entertainment` → `Food` → `Travel&Transport`, involve two edges from `Arts&Entertainment` to `Food` representing each subsequence. Thus, prominent subsequences also become visually prominent due to the thickness of the combined multiple edges.

Of course, not all event subsequences are equal as some correlate to a positive outcome, whereas others correlate to a negative outcome, as determined by Frequence's outcome analytics. The visualization uses color to encode each pattern's association with an associated outcome. For this scenario, the patterns that occur more often with popular users (those who have a lot of followers on Twitter) are more blue. The patterns that occur more often with unpopular users (those who have few followers) are more red. The neutral patterns that appear common to both popular and unpopular users are gray.

The frequency of each pattern also varies, and the visualization uses the thickness of edges to encode how often each pattern occurs using the pattern's support, as defined in Definition 3. For instance, in the example shown in Figure 5, the pattern `Arts&Entertainment` → `Food` → `Travel&Transport` has a support of 0.25, which is why its thickness is half the size of `Arts&Entertainment` → `Food` → `NightlifeSpot`, which has a support of 0.5.

Users can interact with the visualization by moving their mouseover an edge, which will display statistics, such as support and outcome, about the pattern the edge represents. If users click a node, all patterns that do not match the selected node have their opacity diminished so users can focus on patterns that match the user's specification. Users can specify a frequent pattern of interest by clicking on multiple nodes until their pattern is fully selected. Once a pattern is selected, users can use this selection to zoom and filter. Frequence supports two ways to zoom and filter. The first way is by *cohort*, which constrains the dataset to the population that has the selected pattern, and shows the most frequent patterns among that population. The second way is by *pattern*, which uses the whole population, but only mines events that feature the subtypes of the selected pattern. This latter option is useful for navigating hierarchical levels-of-detail.

Interactive Analytics

In addition to exploring patterns using the visualization, users can also adapt and modify the mining algorithm according to their needs by using the control panel located on the left-side of the interface.

By default, the *Statistics* section is selected to show a statistical overview of various aspects of the dataset and patterns. Users can navigate to the *Pattern Definition* section to refine the patterns being mined. Here, users can adjust the support value for patterns, to focus the analytics on either stronger or weaker patterns. By selecting the *Level of Detail* panel, users can change the level-of-detail of the displayed patterns, and choose if filtering is by cohort or by pattern. The *Outcome* panel allows users to select an outcome variable, as well as a threshold on that variable to segment the population. Here, users can also interactively filter the visualization using a range slider to focus on positive or negative patterns. The *Temporal Context* panel allows users to turn on temporal constraints, such as concurrency. This panel also features a slider to specify the time duration for events to be considered concurrent. The *Database* panel allows users choose from where to load their data.

USE CASE: LOCATION SHARING SOCIAL NETWORKS

Some social network researchers are curious about the relationship between people's online social media personas and their offline activities. While mobility data has been difficult to uncover in the past, this is changing due to the proliferation of Location Sharing Services (LSS), where people broadcast the places they visit. One such LSS is Foursquare, a service that allows users to "check-in" at different venues to save and share their locations to friends. According to Foursquare[1], there are over 40 million users and over 4.5 billion check-ins with Foursquare, as of September 2013. A corpus of a subset of Foursquare data was made publicly available[5], which contains over 200,000 users and over 22 million check-ins, and which was imported into Frequence.

In this use case, the researcher was interested in understanding the behavior of users in New York City (NYC), so the analysis described here is limited to check-ins within the boroughs of Manhattan, Queens, and Brooklyn. This was done by filtering all check-ins to fall within a bounding box of latitude and longitude coordinates, surrounding the city. After the geographical filtering, the size of the NYC-constrained database consisted of 17,739 users with 419,023 check-ins in 17,182 unique venues over 11 months. Each unique venue was augmented with Foursquare's hierarchy of 9 top-level categories and 289 sub-level categories, retrieved using Foursquare's Venue Search API[2] to create three levels-of-data on which to mine.

[1] http://www.foursquare.com/about
[2] http://developer.foursquare.com/docs/venues/search

The goal of the analysis was to understand if certain patterns of mobility correlate with popularity to procure evidence for their hypothesis that people who are popular in social media go to different places in different sequences than people who were less popular. In order to gain a proxy for social popularity, users' Foursquare accounts are unified with their Twitter accounts, and the number of Twitter followers was used as a metric of popularity. All Foursquare users in the database had an active Twitter account, and used Twitter to broadcast their check-ins. The analyst was only interested in users who were active users of Twitter, so users that had less than 1000 tweets in their lifetime were filtered out[3]. The median number of Twitter followers for a user the dataset was 265, so the analyst used the Outcome panel to consider patterns by users greater than 265 to be popular, and less than 265 to be unpopular. The analyst also used the Temporal Context panel to constrain event sequences within a 6-hour time window so that only events that occured within that same window were considered a part of the same event sequence. Each of these constraints were determined by rapidly testing multiple hypotheses, and the constraints evolved from original hypotheses to different sensible parameters based upon data exploration.

With the parameters tuned from interactive exploration, an overview of patterns at the coarsest level-of-detail is shown at the top of Figure 6. Interestingly, many of the patterns associated with popular social media users (the blue patterns) contain a `Shop&Service` event. The analyst was also interested in the unpopular patterns, and selected one of the red patterns for further inspection. After selecting the red `Professional&OtherPlaces` $\rightarrow$ `Food` pattern, the researcher used it as a cohort filter to show patterns only from people who have this pattern. Again, the analyst was interested in understanding patterns among less popular users, and to focus on these patterns, the analyst filtered to only the red patterns, shown in the middle of Figure 6. The analyst determines that `Office` $\rightarrow$ `Hotel` correlates with the least popular social media users, and uses this pattern as a cohort filter to mine only users who have this pattern. The bottom of Figure 6 shows the patterns at the finest level of detail, where the names of the actual venues are now visible to the researcher. No specific offices are frequent enough to appear in the patterns, due to the countless companies spread throughout the city, but certain hotels (e.g. the W Hotel in Times Square) do appear and correlate to unpopular users, whereas most popular patterns are those tied with Shopping (e.g. Macy's, H&M). The researcher remarked the potential applications of these findings to marketing professionals, who wish to understand who frequents their venues, and their ability to communicate to their individual social media audiences.

While the researcher acknowledges these insights only apply to a specific demographic that uses Foursquare and broadcasts their check-ins on Twitter, this case study demonstrates the utility of Frequence for exploring temporal patterns of human mobility and reaching insights.

[3]On average, there were 2144 tweets per user in the NYC dataset, before filtering.

USE CASE: MEDICAL INFORMATICS

Due to a large number of medical institutions adopting Electronic Health Records (EHRs), the opportunities to analyze and derive insights from medical data has never been greater. In particular, clinicians and clinical researchers are very interested in understanding patterns of medical events that often lead to positive or negative outcomes, so that they can understand or improve their clinical practices.

This use case involves a team of clinicians and clinical researchers interested in determining if there are particular patterns that lead to patients with lung disease developing sepsis, a potentially deadly medical condition. The institution used a set of 2,336 patients diagnosed with lung disease, each with longitudinal events of diagnostic codes (which encode symptoms, causes, and signs of diseases using ICD-9 standards [4]). ICD-9 codes are organized according to a meaningful hierarchy and this hierarchy was utilized by Frequence as multiple levels of detail.

Of the patients with lung disease, 483 developed sepsis within six months of their diagnosis of lung disease, whereas 1,853 managed to not contract the condition. Prior to using Frequence, the clinical researchers had difficulty drawing any conclusions between these two cohorts, as both types of patients tend to share many of the same diagnosis codes. However, the researchers were interested in mining for frequent patterns to see if the order of these diagnoses has any effect on their patients.

At the top of Figure 7, the coarsest patterns for all of the lung disease patients are shown. The clinician was particularly interested in cardiovascular complications, and noticed that the pattern `CardiacDisorders` $\rightarrow$ `SymptomDisorders` was common yet neutral (that is, this pattern was common to patients who did and did not end up contracting sepsis). After selecting this pattern and filtering by cohort to see the matching patients, the finer level of detail (Level 1) allowed the clinician to see more detailed cardiac conditions, such as cardiac dysrhythmia and heart failure. Other complications, such as acute renal failure (which medical literature suggests is linked to developing sepsis), also appear. However, the clinician is interested in the patterns that led to patients not developing sepsis, and filtered to the positive patterns in the middle of Figure 7. Surprised to see the pattern `HeartFailure` $\rightarrow$ `LungDiseases`, the clinician filtered to the cohort that matched this pattern and pivoted to Level 2, as shown in the bottom of Figure 7. The clinician immediately noticed that patterns that featured both Atrial Fibrillation and Acute Respiratory Failure are red, which is sensible, as medical literature suggests both are risk factors for sepsis. However, the clinician found it interesting that patterns beginning with Acute Respiratory Failure alone were not predictive of sepsis, but rather what happened next in the sequence was more predictive. From the Acute Respiratory Failure node in the first column of the visualization, the patterns diverge into red and blue, making it clear that what happens immediately after such Acute Respiratory Failure will likely determine if the patient will get sepsis or not.

[4]http://www.who.int/classifications/icd/

CONCLUSION AND FUTURE WORK

This paper presents Frequence, a novel visual interface designed to mine and visualize frequent event sequences from temporal events. Frequence includes a novel FSM algorithm to handle real-world data requirements to serve actionable insights. Frequence also supports iterative and flexible exploration with a visual interface where users can explore results and refine parameters of the mining algorithm.

We demonstrate Frequence's utility in two real-world use cases: a medical researcher using frequent sequences in Electronic Medical Records to understand the relationship between the progression of a disease and patient outcomes, and a social network researcher using frequent sequences from a Location-based Sharing Service to understand the relationship between online popularity and offline mobility throughout urban environments.

While these use cases show that Frequence can lead to insights, many topics remain for future work. While Frequence supports interactive parameterization of the mining algorithm, choosing the right parameters might be a challenge for some users. A next goal of Frequence is to augment the user interface with visual feedback to act as a preview of how the parameters will affect mining. An additional item for future work reflects the scalability of our frequent sequence mining algorithm. Although our bitmap-based approach is computationally efficient, as data sets get larger and larger, new approaches will be necessary for Frequence to remain scalable. We plan to investigate applying our algorithm to a cloud-based distributed architecture, as the hierarchical aspects of our mining algorithm are well-suited for such an environment. Finally, a more comprehensive user evaluation is critical to understanding the relationship between analytics and visualization in Frequence. We plan to deploy Frequence to domain partners to conduct longitudinal case studies, where we capture how the capabilities of Frequence affect their workflow and help them reach insights.

REFERENCES

1. Agrawal, R., and Srikant, R. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases* (1994), 487–499.
2. Aigner, W., Miksch, S., Thurnher, B., and Biffl, S. Planninglines: novel glyphs for representing temporal uncertainties and their evaluation. In *Information Visualisation, 2005. Proceedings. Ninth International Conference on* (2005), 457–463.
3. Ayres, J., Gehrke, J. E., Yiu, T., and Flannick, J. Sequential pattern mining using bitmaps. 429–435.
4. Burch, M., Beck, F., and Diehl, S. Timeline trees: visualizing sequences of transactions in information hierarchies. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '08, ACM (New York, NY, USA, 2008), 75–82.
5. Cheng, Z., Caverlee, J., Lee, K., and Sui, D. Z. Exploring millions of footprints in location sharing services. *AAAI ICWSM* (2011).
6. Fails, J., Karlson, A., Shahamat, L., and Shneiderman, B. A visual interface for multivariate temporal data: Finding patterns of events across multiple histories. In *Visual Analytics Science And Technology, 2006 IEEE Symposium On* (2006), 167–174.
7. Mitsa, T. *Temporal Data Mining*, 1st ed. Chapman & Hall/CRC, 2010.
8. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., and Hsu, M. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering 16*, 11 (2004), 1424–1440.
9. Plaisant, C., Milash, B., Rose, A., Widoff, S., and Shneiderman, B. Lifelines: visualizing personal histories. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '96, ACM (New York, NY, USA, 1996), 221–227.
10. Riehmann, P., Hanfler, M., and Froehlich, B. Interactive sankey diagrams. In *IEEE InfoVis* (2005), 233–240.
11. Rosvall, M., and Bergstrom, C. T. Mapping change in large networks. *PLoS ONE 5*, 1 (01 2010), e8694.
12. Shneiderman, B. The eyes have it: a task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on* (1996), 336–343.
13. Wang, T. D., Plaisant, C., Quinn, A. J., Stanchak, R., Murphy, S., and Shneiderman, B. Aligning temporal data by sentinel events: discovering patterns in electronic health records. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, ACM (New York, NY, USA, 2008), 457–466.
14. Wongsuphasawat, K., and Gotz, D. Exploring flow, factors, and outcomes of temporal event sequences with the outflow visualization. In *IEEE InfoVis* (2012).
15. Wongsuphasawat, K., Guerra Gómez, J. A., Plaisant, C., Wang, T. D., Taieb-Maimon, M., and Shneiderman, B. Lifeflow: visualizing an overview of event sequences. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, ACM (New York, NY, USA, 2011), 1747–1756.
16. Wongsuphasawat, K., and Shneiderman, B. Finding comparable temporal categorical records: A similarity measure with an interactive visualization. In *Visual Analytics Science and Technology, 2009. VAST 2009. IEEE Symposium on* (2009), 27–34.
17. Zaki, M. J. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning 42*, 1/2 (2001), 31–60.

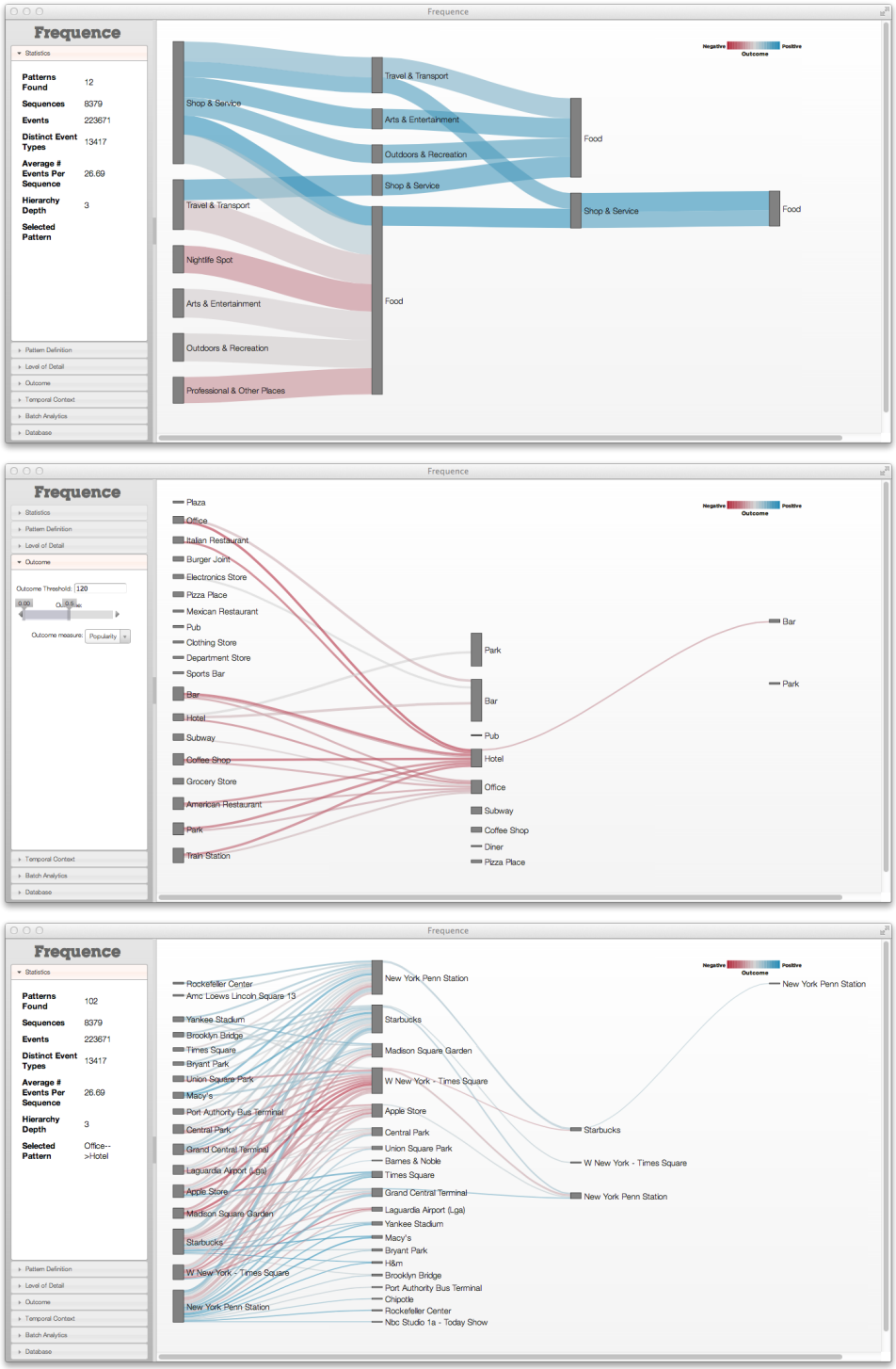

Figure 6. The exploration process of the Foursquare dataset with Frequence. The top figure shows an overview of the coarsest patterns. The middle figure shows a filtered view of unpopular patterns at a finer level-of-detail for the cohort who matched the `Professional&OtherPlaces` → `Food` sequence. The bottom figure shows the patterns at the finest level of detail, where the names of the actual venues become visible, after filtering to the cohort who have the `Office` → `Hotel` pattern.

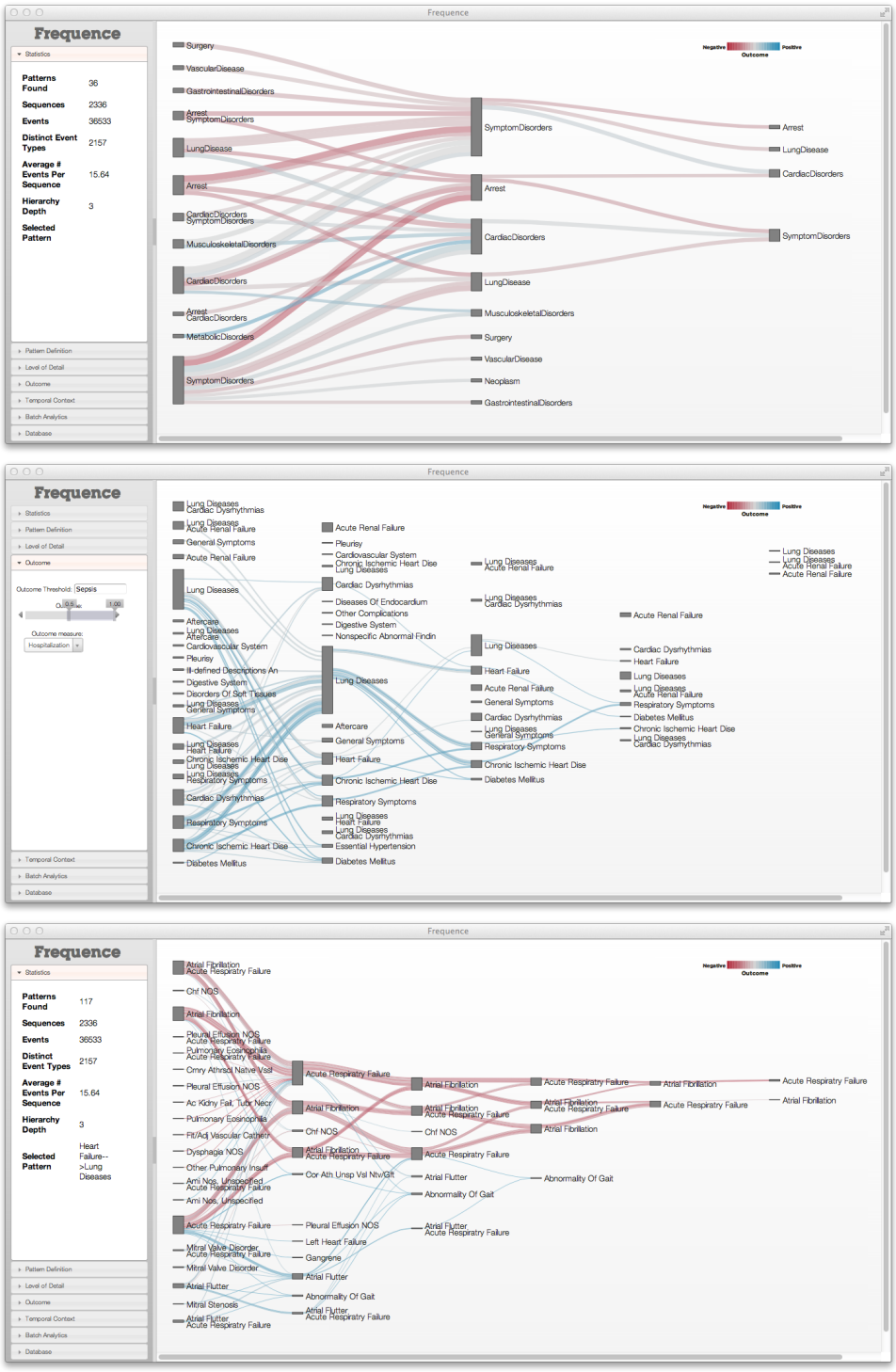

Figure 7. The exploration process of the lung disease patient dataset with Frequence. The top figure shows an overview of the coarsest patterns. The middle figure shows the positive patterns at a finer level-of-detail for the cohort who matched the `CardiacDisorders` → `SymptomDisorders` sequence. The bottom figure shows the patterns at the finest level of detail, after selecting `HeartFailure` → `LungDiseases`.