# StageMap: Extracting and Summarizing Progression Stages in Temporal Event Sequences

Fig. 1. Teaser

**Abstract**— Temporal event sequences are becoming increasingly important in many application domains such as website click streams, user interaction logs, electronic health records and car service records. A real-world dataset with a large number of event sequences and of varying sequence lengths is complex and difficult to analyze. To support visual exploration of the data, it is desirable yet challenging to provide a concise and meaningful overview of sequences. In this paper, we focus on the stage, that is, frequently occurring subsequences, which is common in event sequence datasets and carries high level semantics in the data. We present a novel visualization technique to summarize event sequence data into a set of stage progression patterns. The resulting overview is more concise compared with event-level summarization and supports level-of-detail exploration. We further introduce StageMap, a visual analytics system with three linked views to visualize the stage-level overview, the event-level patterns and the detailed individual sequences. We also present case studies where the system is used in two different domains and discuss advantages and limitations of applying StageMap to various application scenarios.

**Index Terms**—Time Series Data, Data Transformation and Representation, Visual Knowledge Representation, Visual Analytics

## 1 Introduction

Temporal event sequences, that is, ordered series of events which occur over time, appear in a wide range of domains such as website click streams, user interaction logs in software applications, electronic health records and car service records. Analyzing such data can help yield meaningful insights and therefore support decision making. For example, by analyzing user interaction logs, designers can identify users who contend with usability issues and then design interventions to improve user experience. Similarly, by analyzing the online learning click streams, instructors can better understand the learning behavior of students and improve the course design.

Real world event sequences are often complex. A typical dataset can contain thousands or more distinct sequences with hundreds of distinct events. The length of each sequence can vary from a few to hundreds of events. Therefore, many existing visualization techniques are inadequate in directly presenting the data. Instead of visualizing the raw data, recent works have tried to visualize the summary of the data with the help of pattern mining models [5, 9–11, 16, 18, 21, 22]. These methods have shown promising results since the data summary has a lower visual complexity. To further reduce the complexity and provide meaningful semantics, stage-based analysis can be used instead of event-based analysis. Stages and their progression patterns can be found in many event sequence datasets. For example, in online learning click streams analysis, when a student tries to finish an online assignment, he/she may first browse the assignment and then review the course material or ask questions on the course forum. Since many students follow the same steps to finish the assignment, this sequence of learning activities can be defined as a stage, while each activity is recorded as an event. The whole learning behavior of a student can be modeled as a progression of various stages. The benefits of presenting sequences with stages are two-fold: First, since the stage itself can be considered as a summary of events, the stage-based summary is in general more concise compared with the event-based summary. Therefore, it can handle more complex datasets, especially when the lengths of sequences vary a lot. Second, in many applications, stages contain high-level semantics, so users can easily understand the progression pattern without digging into detailed events.

*Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx*

Extracting and analyzing stage progression patterns has been studied for medical data analysis []. However, providing a scalable and meaningful visual summary for stage analysis is still challenging due to the following reasons: First, traditional pattern mining models only preserve or prioritize statistically significant events and discard those insignificant events. Missing these insignificant events has the risk of misleading users on domain specific tasks. For example, in software application log analysis, a rarely occurred but fetal error may not be identified by the model but is important to the end users. Therefore, the summary should keep the detailed information and make sure users are aware of the individual variances within the summary. Second, level-of-detail analysis is useful in visual analysis, especially for large scale datasets. Besides extracting progression patterns, the visual summary should also characterize the hierarchical structure of the data so that users can drill down to local patterns interactively.

In this paper, we propose StageMap, an event sequence summarization method which tries to present sequences with a set of stage progression patterns. Our method first extracts a set of frequently occurring stages. We support soft pattern matches when extracting stages so that later the individual variance is allowed and preserved in the stage progression patterns. We then develop an algorithm to transform the original sequences into a set of progression patterns. Each pattern represents a group of similar sequences and how the corresponding stages evolve over time. As in the online learning click streams analysis, a pattern can show a group of students who share similar learning behaviors and show how they gradually learn different topics and finish various course activities. Depends on the application scenario, different structure can be applied to present a progression pattern. Once the structure is fixed, the algorithm is designed to identify frequent progression patterns while also to minimize the visual complexity. The proposed algorithm can also preserve the hierarchical structure of the sequences so that each pattern can split into several detailed patterns. In this paper, to highlight the branching patterns between stages, we model each pattern into a tree structure while each node in the tree is a stage. We then design a visual analytics system to support visual analysis of the summary. The system has three linked views: the stage map view to show the summary of identified progression patterns, the tree view to represent the detailed events for a highlighted pattern and the sequence view to visualize each individual sequence. We further test our method on two real world datasets.

To summarize, the main contributions of this work include:

- A summarized representation of event sequences with a set of stage progression trees as well as an algorithm to transform raw sequences into the summary.
- An interactive visual analytics system which allows users to explore the summary of the data.
- Case studies with real world datasets to demonstrate the effectiveness of the approach.

## 2 Related Work

This section provides an overview of previous work related to event sequence visualization, including the visual forms for presenting event sequences, the techniques for summarizing event sequences and the methods for stage analysis.

### 2.1 Event Sequence Visualization

The most straightforward way to represent event sequences is placing events along a time axis by their order [8, 17, 31]. Lifelines2 [24] further provides different temporal granularities of the axis to highlight important trends. It is now the most common visual form to represent individual sequences in a visualization system for event sequene analysis.

Other visual forms have also been explored. For example, EventFlow [14, 27], FP-Viz [21], TrailExplorer [19, 20] and CoreFlow [10] model event sequences as a tree structure and visualize it with tree visualization techniques such as Sankey diagram or Sunburst visualization. Besides visualizing the tree-like structure, the Sankey diagram has also been used by Outflow [26], CareFlow [15] and DecisionFlow [4] to show the directed graph extracted from event sequences. MatrixWave [32] is a recent work to visualize event sequences with matrix. Matrix visualization can avoid edge crossing in Sankey diagrams and show the relation between each timestamp. Besides, various visual forms can be applied to show additional attributes associated with events, such as using stacked area charts to display the sentiment trends of events [12].

Interaction is also a key component for visual exploration of event sequences. Typical interaction techniques include event alignment [23, 24], sequence query [4, 7, 30] and filtering [2, 3, 28], and so on. Lifelines2 [23, 24] is an early work to support comprehensive event alignment of event sequences. (S|qu)eries [30] utilizes regular expression to support flexible sequence query. Du et al. [2] propose an approach to help identify a group of sequences which are similar to a user selected sequence.

However, these visualization techniques are limited when handling large scale datasets. For more complex datasets, level-of-detail exploration is always required. Most of these techniques are suitable for detailed analysis while we still need an overview to support high-level analysis and to guide detailed analysis. Our work also adopts some of the existing techniques but focuses on providing a concise and meaningful overview which is unique compared with other techniques.

### 2.2 Event Sequence Summarization

In Section 2.1, we mentioned there are works model event sequences as tree structure or graph structure. These approaches can be considered as one way to summarize event sequences. The complexity of data is reduced by aggregating same events that occur at the same timestamp. However, these methods are sensitive to individual variances among sequences. Similar sequences may not be aggregated due to small variances, which limits the usage of these methods. Recently, CoreFlow [10] proposes an algorithm to extract the tree structure with only high frequency events which can greatly reduce the size of the tree. The low frequency events are discarded in the resulting visual summary which may lead to missing insights in the data.

Sequence clustering can also aggregate similar sequences and provide an overview of the data. LogView [13] uses treemap to show the hierarchical clustering results of sequences. Wang et al. [22] propose a technique to support unsupervised clustering and visualize the result with packed circles. Wei et al. [25] use a self-organizing map to cluster and visualize clickstream data. Many sequence summarization methods can be transformed to a clustering problem, but directly displaying the clusters is not suitable for visual exploration since it is difficult to interpret the meaning of each cluster.

There are also works that generate visual summary based on extracted frequent sequential patterns. TimeStitch [18] applies sequential pattern mining models to medical care data analysis and helps users to discover, construct and compare cohorts. Both Frequence [16] and Peekquence [9] use the frequent pattern mining algorithm and directly visualize mined patterns to help users analyze the data. Furthermore, a three-stage analytic pipeline [11] has been proposed to identify and prune mined sequential patterns. Recently, Chen et al. [1] propose a two-part representation to visualize both the sequential patterns and individual variances with the help of Minimum Description Length principle. In this paper, the concept of the progression stage is similar to the sequential pattern. However, existing works do not consider the sequential relations among stages which limits the flexibility of presenting the inherit data structure.

### 2.3 Sequence Stage Analysis

A variety of data mining methods have been proposed to identify stages and their progression while few visualization techniques have been studied. Many of the data mining methods focus on the application domains such as medical data analysis. For example, Zhou et al. [33] use fused lasso formulation for stage detection. Jackson et al. [6] and Wang et al. [12] both detect the underlying stages based on the Hidden Markov Model. Yang et al. [29] use the EM algorithm to associate stages with sequences. Recently, EventThread [5] visualizes the stage progression patterns with storyline visualization. To the best of our knowledge, it is the first work that focuses on stage progression

visualization. The main difference between EventThread and StageMap is that EventThread mainly focuses on visual representation and simply extracts stages by uniformly segmenting sequences, while StageMap tries to extract the optimized stages and visually represent the data consistently.

## 3 MDL for Event Sequences

A high-level overview often plays a critical role in explorative data analysis, as emphasized in the well-known *visual information seeking mantra* "overview first, zoom and filter, details on demand" [**?**] and manifested in the design of numerous visualization systems. For event sequence data, an overview can serve as a starting point for descriptive analysis [**?**] and help users identify interesting patterns or subsets of data that are worth further exploration.

### 3.1 A Generic Method to Summarize Event Sequences

Our approach to visually summarize multiple event sequences is based on a two-part representation of the original data. The two-part representation consists of a set of sequential patterns and a set of corrections. Each event sequence is mapped to a pattern and the corrections specify the edits needed to transform the pattern to the original sequence. The edits may include insertion or deletion of events from the pattern. Fig. **??** gives an example. It shows six event sequences $\{S_1, S_2, ..., S_6\}$ together with the corresponding patterns and corrections. There are two sequential patterns $P_1$ and $P_2$. The original sequences can be reconstructed from either $P_1$ or $P_2$ by removing (-) or adding (+) events. For instance, $S_2$ can be recovered from $P_1$ by adding event A while $S_3$ can be recovered from $P_1$ by adding event E. $S_4$ can be recovered from pattern $P_2$ by removing event E and inserting event C. $S_1$ is exactly the same as pattern $P_1$, therefore no correction is needed.

The intuition of this two-part representation is to exploit the similarity of event sequences and identify a set of sequential patterns that can give a concise visual summary of the data. In the example in Fig. **??**, $P_1$ and $P_2$ can roughly characterize the six sequences by representing $\{S_1, S_2, S_3\}$ and $\{S_4, S_5, S_6\}$ respectively. This is common in many application scenarios. For example, a series of interdependent faults can happen in the same sequential manner in multiple vehicles. Visitors of a commerce website may follow a similar sequence of pages to complete their orders.

The corrections part, on the other hand, specifies the information loss if the original data is visually represented by the sequential patterns. To reduce information loss, more elaborated patterns can be introduced. For example, in Fig. **??**, the information loss can be reduced by mapping $S_4$ to a new pattern [A, B, C] instead of $P_2$. However, such changes can increase the visual complexity of the overview with more patterns to be displayed. The extreme case is when each individual sequence is treated as a pattern: there is no information loss, however severe visual clutter could occur when plotting all the sequences in a single visualization, making it much more challenging to identify high-level patterns. Essentially, we need to consider a trade-off between the readability of the visualization and the completeness of the information communicated through it.

### 3.2 The MDL Principle

We introduce the MDL principle [**?**, **?**] to identify a set of sequential patterns for an overview of the data while balancing the information loss in it. MDL is a well known information criterion for statistical model selection. It has been adopted for constructing optimized layout of hierarchical visualizations [**?**]. The MDL principle basically states that the best model for a dataset results in minimized description length of it. Like most authors, we apply the more 'practical' or crude version of MDL instead of the ideal MDL. The ideal MDL tries to find the shortest program in a general-purpose computer language that can print the data. On the other hand, in the crude version, the description length of a dataset is composed of two parts: (a) the encoding of the model $L(\mathcal{M})$ and (b) the encoding of the data with the help of the model $L(\mathcal{D}|\mathcal{M})$. The best model $\hat{\mathcal{M}}$ should minimize the total description length, which is $L(\mathcal{M}) + L(\mathcal{D}|\mathcal{M})$.

For event sequences, we consider the sequential patterns as the model. The original sequences are coded by specifying the edits to the corresponding patterns. The total description length is the sum of the pattern lengths and the corrections length. The best set of sequential patterns that represents the original data should minimize the sum.

The MDL principle applied in this scenario is directly connected to the goal we intend to achieve in the overview. Simplifying the overview which shows the sequential patterns corresponds to minimizing $L(\mathcal{M})$, whereas $L(\mathcal{D}|\mathcal{M})$, the corrections length, is added to the objective function to penalize the information loss in it.

### 3.3 Denotations and Formal Problem Definition

Now we start to introduce the denotations and formally define the description length of the two-part representation. An event sequence is an ordered list of events $S = [e_1, e_2, ..., e_n]$ where $e_i \in \Omega$, an event alphabet. Given a set of event sequences $\mathbb{S} = \{S_1, S_2, ..., S_m\}$, the goal is to identify a set of patterns $\mathcal{P} = \{P | P = [e_1, e_2, ..., e_l]\}$ and a mapping $f : \mathbb{S} \to \mathcal{P}$ from the event sequences to the patterns that can minimize the total description length:

$$L(\mathcal{P}, f) = \sum_{P \in \mathcal{P}} L(P) + \sum_{S \in \mathbb{S}} L(S|f(S)) \tag{1}$$

In this equation, $L(P)$ is the description length of pattern $P$ and $L(S|f(S))$ is the description length of $S$ given its pattern $f(S)$. Considering that 1) a pattern can be described by simply listing the events in it[1] and 2) an edit can be fully specified by the position and the event involved and its description length can be roughly treated as a constant, Eqn. **??** can be rewritten as:

$$L(\mathcal{P}, f) = \sum_{P \in \mathcal{P}} len(P) + \alpha \sum_{S \in \mathbb{S}} \|edits(S, f(S))\| + \lambda \|\mathcal{P}\| \tag{2}$$

where $len(P)$ is the number of events in the pattern and $edits(S, f(S))$ is a set of edits that can transform $f(S)$ to $S$. We further introduce the parameter $\alpha$ in Eqn. **??** to control the importance of minimizing information loss over reducing visual clutter in the overview, following the practice used by Veras and Collins [**?**]. The third term with the parameter $\lambda$ is added to directly control the total number of patterns. Increasing $\lambda$ will reduce the number of patterns in the optimized result. Therefore the scalability of the overview can be improved by setting $\lambda$ properly.

The mapping $f$ clusters the event sequences: sequences mapped to the same pattern can be considered as in the same cluster. We denote a cluster as a tuple $c = (P, G)$ where $G = \{S | S \in \mathbb{S} \wedge f(S) = P\}$ is the set of sequences mapped to pattern $P$. We denote the set of tuples for all the clusters as $\mathcal{C} = \{(P_1, G_1), (P_2, G_2), ..., (P_k, G_k)\}$, where $\{G_1, G_2, ..., G_k\}$ forms a partition of $\mathbb{S}$. Therefore finding $\hat{f}$ and $\hat{\mathcal{P}}$ that minimize $L(\mathcal{P}, f)$ is equivalent to finding $\hat{\mathcal{C}}$ that minimize $L(\mathcal{C})$:

$$L(\mathcal{C}) = \sum_{(P,G) \in \mathcal{C}} len(P) + \alpha \sum_{(P,G) \in \mathcal{C}} \sum_{S \in G} \|edits(S, P)\| + \lambda \|\mathcal{C}\| \tag{3}$$

To summarize, our goal is to identify a partition/grouping of the sequences and a representative pattern for each group that can minimize the total description length. Conceptually it seems to be similar to sequence clustering algorithms. However the other methods do not follow an information-theoretic approach. Furthermore, the patterns can give an interpretable coarse-level summary of the original data, which is not possible with the existing sequence clustering techniques.

## 4 Computing MDL Representation

We introduce the algorithm to identify a grouping of the sequences and a representative pattern for each group that minimize $L(\mathcal{C})$ in Eqn. **??**.

[1] In this work, minimizing the description length is not an end goal, but a means to extract meaningful sequential patterns for a succinct visual display. Therefore we do not use compression schemes such as Huffman coding [**?**] to shorten the code lengths for the events and we consider the events are described with a constant length code. Same for encoding the edits.

### 4.1 Basic Algorithm

We now present our first algorithm called *MinDL*. Since the optimization problem itself entails a rather large search space, we adopt a heuristic approach using a bottom up strategy. Initially, each sequence starts in its own cluster and is treated as a sequential pattern by itself. Starting from that, we iteratively merge pairs of clusters and compute the representative sequential patterns for the new clusters. The merges are determined in a greedy manner: the algorithm always choose to combine the pair that leads to the maximum description length reduction in each iteration. The algorithm stops when it can no longer find a pair to further reduce $L$.

**Input**: sequences $\mathcal{S} = \{S_1, S_2, ..., S_n\}$
**Output**: pattern and cluster tuples $\mathcal{C} = \{(P_1, G_1), (P_2, G_2), ..., (P_k, G_k)\}$
`/* Initialization phase */`
1 $\mathcal{C} = \{(P, G) | P = S, G = \{S\}$ for all $S \in \mathcal{S}\}$;
2 PriorityQueue $Q = \emptyset$;
3 **for** *all pairs* $c_i, c_j \in \mathcal{C}$ *and* $i \neq j$ **do**
4 $\quad \Delta L, c^* = Merge(c_i, c_j)$;
5 $\quad$ **if** $\Delta L > 0$ **then**
6 $\quad\quad$ insert $(\Delta L, c^*, c_i, c_j)$ into $Q$;
7 $\quad$ **end**
8 **end**
`/* Iterative merging phase */`
9 **while** $Q \neq \emptyset$ **do**
10 $\quad$ retrieve $(\Delta L, c^*, c_i, c_j)$ from $Q$ with the largest $\Delta L$;
11 $\quad c_{new} = c^*$;
12 $\quad$ remove $c_i, c_j$ from $\mathcal{C}$, add $c_{new}$ to $\mathcal{C}$;
13 $\quad$ remove all pairs containing $c_i$ or $c_j$ from $Q$;
14 $\quad$ **for** $c \in \mathcal{C} - c_{new}$ **do**
15 $\quad\quad \Delta L, c^* = Merge(c, c_{new})$;
16 $\quad\quad$ **if** $\Delta L > 0$ **then**
17 $\quad\quad\quad$ insert $(\Delta L, c^*, c, c_{new})$ into $Q$;
18 $\quad\quad$ **end**
19 $\quad$ **end**
20 **end**
21 **return** $\mathcal{C}$

**Algorithm 1:** MinDL

*MinDL* is described formally in Algorithm **??**. The algorithm is subdivided into two phases - *Initialization* and *Iterative merging*. In the *Initialization* phase, $\mathcal{C}$ is set to contain all the sequences in $\mathcal{S}$ as individual clusters. The algorithm then computes the description length reduction $\Delta L$ for all the pairs in $\mathcal{C}$ and uses a standard priority queue $Q$ to store the pairs with a positive $\Delta L$. With that the algorithm can efficiently retrieve the pair with the maximum $\Delta L$ in constant time. The subroutine *Merge* in line 4 returns not only $\Delta L$ by merging $c_i$ and $c_j$, but also $c^* = (P^*, G_i \cup G_j)$ where $P^*$ is the optimal sequential pattern for the merged group which minimizes the description length for the sequences in $G_i \cup G_j$. $c^*$ is stored together with $\Delta L$ and $(c_i, c_j)$ in $Q$ to avoid recomputation.

In the *Iterative merging* phase, the algorithm picks the pair $(c_i, c_j)$ with the maximum $\Delta L$ from $Q$. It updates $\mathcal{C}$ by removing $c_i$, $c_j$ and inserting $c^*$. $Q$ is updated by adding new pairs of clusters containing $c^*$ with a positive $\Delta L$. This process is repeated until $Q$ is empty. The remaining tuples in $\mathcal{C}$ specify a grouping of the sequences along with the representative patterns which give an optimized description length of the original data.

The core subroutine in *MinDL* is *Merge*, which appears in line 4 and line 15. It is described in Algorithm. **??**. *Merge* calculates the cost reduction $\Delta L$ when combining a pair of clusters $c_i = (P_i, G_i)$ and $c_j = (P_j, G_j)$. It also returns the sequential pattern $P^*$ after merging. The algorithm initializes $P^*$ as the Longest Common Subsequence (LCS) of $P_i$ and $P_j$ and iteratively add the remaining events in $P_i$ and $P_j$ to it, starting from those that appear most frequently in $G_i$ and $G_j$. The iteration stops when $\Delta L$ no longer increases or is less than 0. The intuition behind this procedure is that the pattern $P^*$ should be a mixture of $P_i$ and $P_j$. Starting from the LCS of $P_i$ and $P_j$ can greatly reduce the efforts needed to build the sequential pattern $P^*$ from scratch.

**Input**: $c_i = (P_i, G_i)$,$c_j = (P_j, G_j)$
**Output**: $\Delta L$ and $c^* = (P^*, G_i \cup G_j)$ by merging $c_i$ and $c_j$
`/* Initialization phase */`
1 init pattern $P^* = P = LCS(P_i, P_j)$;
2 candidate events $E_c = P_i - P \cup P_j - P$;
3 sort $E_c$ by frequency in desc order;
4 $\Delta L = -1$;
`/* Pattern buildup phase */`
5 **for** *e in* $E_c$ **do**
6 $\quad P = Add(P, e)$;
7 $\quad \Delta L' = len(P_i) + len(P_j) - len(P) + \alpha \sum_{S \in G_i} edits(S, P_i) + \alpha \sum_{S \in G_j} edits(S, P_j) - \alpha \sum_{S \in G_i \cup G_j} edits(S, P) + \lambda$;
8 $\quad$ **if** $\Delta L' < 0$ *or* $\Delta L' < \Delta L$ **then**
9 $\quad\quad$ break;
10 $\quad$ **else**
11 $\quad\quad \Delta L = \Delta L'$, $P^* = P$;
12 $\quad$ **end**
13 **end**
14 **return** $\Delta L, c^* = (P^*, G_i \cup G_j)$

**Algorithm 2:** Merge

The function *edits* in line 7 calculates the *minimum* number of edits that can transform a pattern $P$ to a sequence $S$. Different types of edits can be supported in the algorithm, given that the minimum number of edits are computed accordingly. For example, if we allow missing or additional events in the pattern, the minimum number of edits can be obtained by computing the LCS distance between $P$ and $S$. Adding event substitution into the available types of edits, we get Levenstein distance. The algorithm can support even more editing types such as swapping the positions of adjacent events. In this paper we mostly consider event insertion and deletion as the available editing operations. In Section 4.3 we use an example to illustrate how swapping adjacent events can be supported in the same framework.

### 4.2 Speedup with Locality Sensitive Hashing (LSH)

An analysis on the time complexity of *MinDL* shows that it can be quite time consuming even for a moderate amount of data. Given $n$ event sequences, the *MinDL* algorithm runs the subroutine *Merge* for $O(n^2)$ times to calculate $\Delta L$ for all pairs of clusters. The subroutine *Merge* itself has a time complexity of $O(kmd^2)$, where $m$ is the number of sequences in the combined cluster, $k$ is the number of iterations in the pattern buildup phase (line 5-13 in Algorithm. **??**) and $d$ is the average length of the sequences. We assume the minimum number of edits can be computed efficiently through dynamic programming, hence the time complexity of computing *edits* is $O(d^2)$.

To tackle this challenge, we propose a fast randomized approximation algorithm utilizing *Locality Sensitive Hashing (LSH)* [**?**]. The intuition of this approach is that pairs of sequences/patterns which share very few common events or no common event at all (regardless of the order) can be skipped when searching for candidate pairs to merge. If we can design a method that can quickly filter out such pairs, the times of calling function *Merge*, the most time consuming routine, can be significantly reduced.

Based on this observation, we integrate weighted LSH [**?**] into the *MinDL* algorithm. Weighted LSH takes a predefined threshold $th$ within the range $(0.0, 1.0)$ as a parameter. If two multisets have a weighted Jaccard similarity larger than $th$, they will have the same hash value with a sufficiently high probability. We use weighted LSH to quickly identify pairs of sequences/patterns with similar sets of events regardless of their exact order. This allows us to prioritize the clusters when searching for candidates to merge.

When applying LSH, a higher threshold $th$ can filter out more candidates and make the algorithm faster. However, the risk of missing potential candidates also becomes higher. We follow the strategy proposed by Koga et al. [**?**] and run *MinDL* for multiple iterations while

Fig. 2. Algorithm performance comparison on two real-world datasets, vehicle fault sequences (VFS) and Agavue [**?**]. We sample the Agavue dataset to create test data with different number of sequences. We run algorithms on a PC with 2.5GHz Intel dual-core i5 CPU with 4GB RAM. The algorithms are implemented in Python except that HAC in scikit-learn and weighted LSH in datasketch use external C libraries.

gradually decreasing $th$. In the first few runs, we use higher $th$ so each time fewer pairs need to be checked in *MinDL*. To ensure no possible merge is missed due to the usage of LSH, we gradually decrease the threshold in the later runs such that more candidate pairs could be considered. Since the number of clusters decreases quickly during the first few runs, this method still can significantly reduce the running time. In this work, the threshold setting is guided by the experimental result. The *MinDL+LSH* algorithm is described in detail in the Appendix.

Fig. 3. Change of the description length ($L(\mathcal{C})$) over time as *MinDL* and *MinDL+LSH* progress, tested on the Agavue dataset. $L(\mathcal{C})$ is normalized with its initial value when each sequence is treated as an individual pattern.

Fig. 4. An example sequence cluster where events may have different orders when compared to the pattern. By adding transposition operation in possible edits, the algorithm allows small perturbations in event order.

We conduct experiments to evaluate the effectiveness of the speedup strategy on two datasets, vehicle fault sequences (VFS) and Agavue [**?**]. Detailed information of the two datasets are described in Section. 6. Some basic statistics about the datasets and the experimental results are displayed in Fig. **??**. It can be observed that we can reduce 95% to 99% running time of *MinDL* with the help of LSH. We also test the running time of a standard clustering algorithm *Hierarchical Agglomerative Clustering (HAC)* and a SPM algorithm [**?**]. For fair comparison, we use editing distance as the metric in the HAC algorithm, and use the minimum cluster size in our *MinDL* algorithm as the support threshold in the SPM algorithm. From the result, we can observe that *MinDL+LSH* is much faster than both *HAC* and *SPM*, especially when number of sequences becomes larger.

Fig. **??** compares *MinDL* and *MinDL+LSH*. It shows how the total description length $L$ decreases over time as the two algorithms progress. Besides the dramatic difference in the decreasing speed, we also observe that the resulted description length is similar for the two algorithms. This means that the two algorithms can achieve similar optimization results. To further validate this conclusion, we also compare the clustering results before and after embedding LSH with the Adjusted Rand Index (ARI). ARI is a common metric to compare clustering results. It ranges from -1 to 1 where 0 means random clustering and 1 means identical results. An ARI larger than 0.5 means that the results are very similar [**?**]. Fig. **??** shows that all the results have an ARI larger than 0.5. Therefore, we can safely conclude that adding LSH does not have significant negative effect on the clustering results.

### 4.3 Soft Pattern Matching

In the algorithm, the individual sequences may deviate from the patterns with missing or additional events, given that they do not add too much to the corrections part. Therefore the method is quite robust to noises, common in real-world data. The algorithm is also generic and can support more editing operations such as swapping the positions of adjacent events, thus allowing small perturbations in event order. Fig. **??** shows an example when we include insertion, deletion and transposition between two successive events as the possible edits. In the algorithm, the minimum number of edits can be determined by following the method to calculate the Damerau-Levenshtein distance [**?**]. Fig. **??** shows that the events in the patterns appear in the individual sequences in different order.

## REFERENCES


[1] Y. Chen, P. Xu, and L. Ren. Sequence synopsis: Optimize visual summary of temporal event data. *IEEE transactions on visualization and computer graphics*, 24(1):45–55, 2018.

[2] F. Du, C. Plaisant, N. Spring, and B. Shneiderman. Eventaction: Visual analytics for temporal event sequence recommendation. *Proceedings of the IEEE Visual Analytics Science and Technology*, 2016.

[3] F. Du, B. Shneiderman, C. Plaisant, S. Malik, and A. Perer. Coping with volume and variety in temporal event sequences: Strategies for sharpening analytic focus. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–14, 2016.

[4] D. Gotz and H. Stavropoulos. Decisionflow: Visual analytics for high-dimensional temporal event sequence data. *IEEE transactions on visualization and computer graphics*, 20(12):1783–1792, 2014.

[5] S. Guo, K. Xu, R. Zhao, D. Gotz, H. Zha, and N. Cao. Eventthread: Visual summarization and stage analysis of event sequence data. *IEEE transactions on visualization and computer graphics*, 24(1):56–65, 2018.

[6] C. H. Jackson, L. D. Sharples, S. G. Thompson, S. W. Duffy, and E. Couto. Multistate markov models for disease progression with classification error. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 52(2):193–209, 2003.

[7] J. Krause, A. Perer, and H. Stavropoulos. Supporting iterative cohort construction with visual temporal queries. *IEEE transactions on visualization and computer graphics*, 22(1):91–100, 2016.

[8] M. Krstajic, E. Bertini, and D. Keim. Cloudlines: Compact display of event episodes in multiple time-series. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2432–2439, Dec 2011.

[9] B. C. Kwon, J. Verma, and A. Perer. Peekquence: Visual analytics for event sequence data. In *ACM SIGKDD 2016 Workshop on Interactive Data Exploration and Analytics*, 2016.

[10] Z. Liu, B. Kerr, M. Dontcheva, J. Grover, M. Hoffman, and A. Wilson. Coreflow: Extracting and visualizing branching patterns from event sequences. 2017.

[11] Z. Liu, Y. Wang, M. Dontcheva, M. Hoffman, S. Walker, and A. Wilson. Patterns and sequences: Interactive exploration of clickstreams to understand common visitor paths. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):321–330, 2017.

[12] Y. Lu, M. Steptoe, S. Burke, H. Wang, J.-Y. Tsai, H. Davulcu, D. Montgomery, S. R. Corman, and R. Maciejewski. Exploring evolving media discourse through event cueing. *IEEE transactions on visualization and computer graphics*, 22(1):220–229, 2016.

[13] A. Makanju, S. Brooks, A. N. Zincir-Heywood, and E. E. Milios. Logview: Visualizing event log clusters. In *Privacy, Security and Trust, 2008. PST'08. Sixth Annual Conference on*, pp. 99–108. IEEE, 2008.

[14] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman. Temporal event sequence simplification. *IEEE transactions on visualization and computer graphics*, 19(12):2227–2236, 2013.

[15] A. Perer and D. Gotz. Data-driven exploration of care plans for patients. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pp. 439–444. ACM, 2013.

[16] A. Perer and F. Wang. Frequence: interactive mining and visualization of temporal frequent event sequences. In *Proceedings of the 19th international conference on Intelligent User Interfaces*, pp. 153–162. ACM, 2014.

[17] C. Plaisant, B. Milash, A. Rose, S. Widoff, and B. Shneiderman. Lifelines: visualizing personal histories. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 221–227. ACM, 1996.

[18] P. J. Polack, S.-T. Chen, M. Kahng, M. Sharmin, and D. H. Chau. Timestitch: Interactive multi-focus cohort discovery and comparison. In *Visual Analytics Science and Technology (VAST), 2015 IEEE Conference on*, pp. 209–210. IEEE, 2015.

[19] Z. Shen and N. Sundaresan. Trail explorer: Understanding user experience in webpage flows. *IEEE VisWeek Discovery Exhibition*, pp. 7–8, 2010.

[20] Z. Shen, J. Wei, N. Sundaresan, and K.-L. Ma. Visual analysis of massive web session data. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pp. 65–72. IEEE, 2012.

[21] J. Stasko and E. Zhang. Focus+ context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, pp. 57–65. IEEE, 2000.

[22] G. Wang, X. Zhang, S. Tang, H. Zheng, and B. Y. Zhao. Unsupervised clickstream clustering for user behavior analysis. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pp. 225–236. ACM, 2016.

[23] T. D. Wang, C. Plaisant, A. J. Quinn, R. Stanchak, S. Murphy, and B. Shneiderman. Aligning temporal data by sentinel events: discovering patterns in electronic health records. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 457–466. ACM, 2008.

[24] T. D. Wang, C. Plaisant, B. Shneiderman, N. Spring, D. Roseman, G. Marchand, V. Mukherjee, and M. Smith. Temporal summaries: Supporting temporal categorical searching, aggregation and comparison. *IEEE transactions on visualization and computer graphics*, 15(6), 2009.

[25] J. Wei, Z. Shen, N. Sundaresan, and K.-L. Ma. Visual cluster exploration of web clickstream data. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pp. 3–12. IEEE, 2012.

[26] K. Wongsuphasawat and D. Gotz. Exploring flow, factors, and outcomes of temporal event sequences with the outflow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2659–2668, Dec 2012.

[27] K. Wongsuphasawat, J. A. Guerra Gómez, C. Plaisant, T. D. Wang, M. Taieb-Maimon, and B. Shneiderman. Lifeflow: visualizing an overview of event sequences. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 1747–1756. ACM, 2011.

[28] K. Wongsuphasawat and B. Shneiderman. Finding comparable temporal categorical records: A similarity measure with an interactive visualization. In *Visual Analytics Science and Technology, 2009. VAST 2009. IEEE Symposium on*, pp. 27–34. IEEE, 2009.

[29] J. Yang, J. McAuley, J. Leskovec, P. LePendu, and N. Shah. Finding progression stages in time-evolving event sequences. In *Proceedings of the 23rd international conference on World wide web*, pp. 783–794. ACM, 2014.

[30] E. Zgraggen, S. M. Drucker, D. Fisher, and R. Deline. (s|qu)eries: Visual regular expressions for querying and exploring event sequences. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*, pp. 2683–2692, 2015.

[31] J. Zhao, C. Collins, F. Chevalier, and R. Balakrishnan. Interactive exploration of implicit and explicit relations in faceted datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2080–2089, 2013.

[32] J. Zhao, Z. Liu, M. Dontcheva, A. Hertzmann, and A. Wilson. Matrixwave: Visual comparison of event sequence data. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 259–268. ACM, 2015.

[33] J. Zhou, J. Liu, V. A. Narayan, J. Ye, A. D. N. Initiative, et al. Modeling disease progression via multi-task learning. *NeuroImage*, 78:233–248, 2013.