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 [5,9,11,12,17,19,22,23]. 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, 18, 32]. Lifelines2 [25] 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 [15, 28], FP-Viz [22], TrailExplorer [20, 21] and CoreFlow [11] 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 [27], CareFlow [16] and DecisionFlow [4] to show the directed graph extracted from event sequences. MatrixWave [33] 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 [13].

Interaction is also a key component for visual exploration of event sequences. Typical interaction techniques include event alignment [24, 25], sequence query [4, 7, 31] and filtering [2, 3, 29], and so on. Lifelines2 [24, 25] is an early work to support comprehensive event alignment of event sequences. (S|qu)eries [31] 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 [11] 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 [14] uses treemap to show the hierarchical clustering results of sequences. Wang et al. [23] propose a technique to support unsupervised clustering and visualize the result with packed circles. Wei et al. [26] 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 [19] applies sequential pattern mining models to medical care data analysis and helps users to discover, construct and compare cohorts. Both Frequence [17] 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 [12] 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. [34] use fused lasso formulation for stage detection. Jackson et al. [6] and Wang et al. [13] both detect the underlying stages based on the Hidden Markov Model. Yang et al. [30] 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

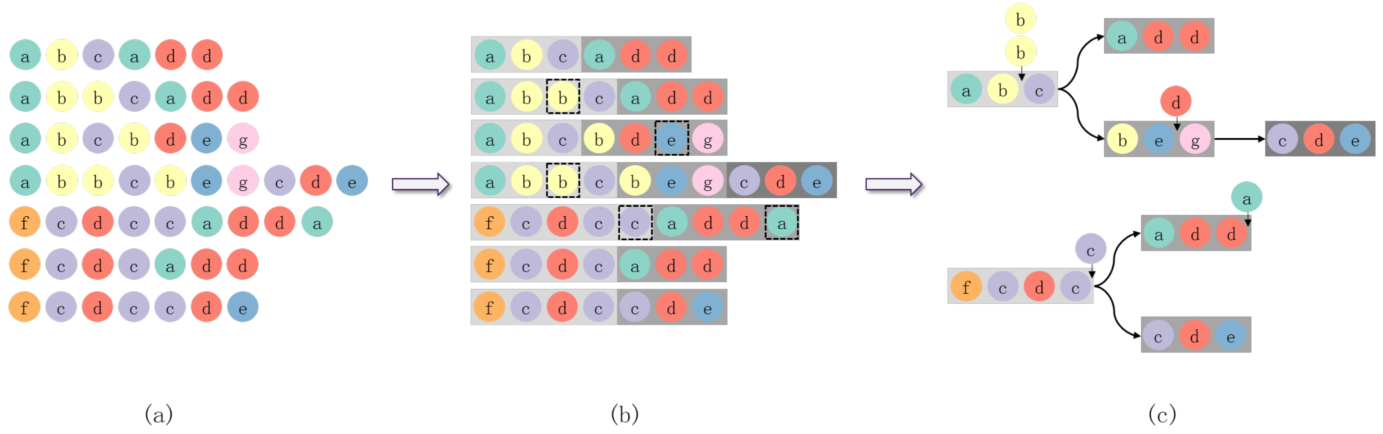

Fig. 2. stage progression summary

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 Summary of Stage Progression for Event Sequences

In this section, we introduce the proposed data model to extract the stage progression summary. We first give the problem statement and then describe the detailed algorithm to generate the summary which can be directly visualized by the system.

3.1 Problem Statement

Our idea of visualizing event sequences is to extract a set of stages and summarize the data into several stage progression patterns. Stage is a frequently occurring subsequence among the sequences and usually contain high-level structures. Fig. 2(a) shows seven event sequences. We can represent each sequence as a series of stages, as shown in Fig. 2(b). There are five different stages, and each of them occurs at least twice in the sequences. To provide a concise and meaningful overview of stages, we can then identify a set of stage progression patterns. In Fig. 2(c), seven sequences are splitted into two groups. The high-level progression pattern of each group can be described with a unified structure. In real-world application, such as web clickstream analysis, these patterns can reveal certain human behavior which is critical for the analysis.

In actual practices, the dataset is much more complex and noisy. To identify the user requirements, we have surveyed existing works, explored several real-world dataset and collaborated with analysts in online learning clickstream analysis. We observe that an effective visual summary of event sequences should have the following charactoristics:

- **Minimizing visual complexity.** Different from existing pattern mining models, the proposed model should also consider the visual encoding of the summary and try to reduce visual complexity.
- **Dealing with individual variance.** As shown in Fig. 2, when matching each sequence with stages, there are events which cannot fit into any stages. We define these events as addition events. Such addition events occur due to system error when recording the data or because of same behaviors still contain individual variance. Therefore, on one hand the model need to be robust to addition events to group similar sequences, on the other hand addition events should be preserved and presented to the user in the visual summary.
- **Supporting interactive analysis.** When analyzing large scale datasets, level-of-detail analysis is usually required to allow users drill down to details step by step. Therefore, the model should be computational efficient and update the summary interactively.

3.2 Algorithm

Algorithm 1: SPTree

```
   Input: sequences $\mathcal{X} = \{X_1, X_2, ..., X_n\}$
   Output: stage progression trees $\mathcal{T} = \{T_1, T_2, ..., T_k\}$
1  initialize $\mathcal{T} = \{\mathcal{X}\}$, stage queue $\mathcal{Q} = \{\}$, covering $\mathcal{C} = \mathcal{X}$;
2  stage candidates $\mathcal{S} = Gap-BIDE(\mathcal{X})$;
3  $\mathcal{Q} = \{sortS(\mathcal{S})\}$;
4  while $\mathcal{Q} \neq \emptyset$ do
5      pop $S_i$ from Q;
6      for each $C_j \in \mathcal{C}$ do
7          $C_j^* = Cover(C_j, S_i)$;
8          replace $C_j$ with $C_j^*$;
9      end
10     $\mathcal{T}^* = BuildTrees(\mathcal{C})$;
11     if $Cost(\mathcal{T}^*) < Cost(\mathcal{T})$ then
12         $\mathcal{T} = \mathcal{T}^*$;
13     end
14 end
15 return $\mathcal{T}$
```

The algorithm for summarizing progression patterns is called *SPTree*. Algorithm 1 describes the whole algorithm pipeline. Taken the input sequences $\mathcal{X}$, *SPTree* first generates a set of stage candidates $\mathcal{S}$ which occur frequently in the dataset. To this end, we employ *Gap-BIDE* [10], a technique used for efficiently mining closed subsequences. We employ this technique mainly because it introduces wildcards when matching potential subsequences, which is consistent with our idea of allowing individual variance in each sequence. To be more specific, for each stage candidates, we can set up a constraint Th_{gap} in *Gap-BIDE* to limit the maximum number of wildcards when matching sequences. Then, *Gap-BIDE* will identify all the subsequences which have the frequency higher than a predefined threshold Th_{sup}. The candidates will then be used to construct the summary.

The algorithm then iteratively updates the summary by adding more stages from the candidates. We sort the candidates so that more frequently occurred stages can be considered first. This is the core subroutine of *SPTree*. Each iteration can be separated into two phases, that is, updating covering and constructing summary. In the first stage, a covering with the newly added stage is generated. Here a covering refers a presentation of a sequence with stages and additions. The covering determines where a stage matches to the original sequence.

Algorithm 2 describes how we compute the covering. For each stage candidate S_i, the subroutine *Cover* looks for all occurrences of S_i and replaces the corresponding subsequences. To accelerate the algorithm, the matchings between stages and sequences can be stored as a lookup table when employing *Gap-BIDE*, so *Cover* only need to check cells in the table.

In the second stage, *SPTree* builds a summary given the updated covering. In this paper, we try to build a visual summary which highlights the branching patterns between stages. Therefore, we construct each pattern as a tree structure with each stage as a node. When constructing other structures for summarizing progression patterns, we only need to change the algorithm in the second phase (i.e., line 11 in Algorithm 1).

Algorithm 2: Cover

```
Input: a covering $C$, stage candidates $\mathcal{S}$
Output: an updated covering $C^*$
1  for each $S_i \in \mathcal{S}$ do
2      for all $o \in$ occurrences of $S_i$ do
3          if $C \cap o = \emptyset$ then
4              $C = C \cup o$;
5          end
6      end
7      if no additions in C then
8          Break;
9      end
10 end
11 return $C$
```

Gap-BIDE can generate a rich set of candidates, however, some of the candidates are not valid for an optimized summarization. To validate each candidate, inspired by [1], we further introduce a cost function as described in Eqn. 1:

$$Cost(\mathcal{T},\mathcal{C}) = \sum_{S\in\mathcal{S},T\in\mathcal{T}} \{1|S \in T\} + \lambda ||addition|| \qquad (1)$$

The two terms in Eqn. 1 correspond to the main visual elements in the summary, that is, the number of nodes in trees and the number of addition events. The parameter λ is introduced to balance the importance of the two terms. At each iteration, the summary $\mathcal{T}$ is updated only when the candidate can reduce the cost.

Computational complexity. In *SPTree*, the computational complexities for *Gap-BIDE*, *Cover* and *BuildTrees* are [TODO], $O(m)$, $\sum_{S\in\mathcal{S},T\in\mathcal{T}}\{1|S \in T\}$ respectively, where m is the number of stage candidates. The most time consuming step is generating stage candidates. However, this step can be computed offline without interfering interactive analysis. For example, when users select a subset of sequences, e.g., sequences belong to a specific progression pattern, the alogorithm can update the candidates by directly recalculate the threshold of stages with a time complexity of $O(mn)$, where n is the number of sequences. We report the running time of the proposed algorithm in Section 5.

Parameter settings. There are three parameters Th_{sup}, Th_{gap}, λ need to be predefined. Th_{sup} ranges from 0 to n. It should be set small enough so that no important stages are eliminated at the first step. Based on experiments, we set Th_{sup} to be 0.05 times n since the result becomes steady when further decreases the thresould. Th_{gap} ranges from 0 to the maximum length of sequences. Cross validation can be used to select the optimized value. λ is used to balance the importance of patterns and additions. [TODO]

4 Visualization

5 Evaluation

We demonstrated

6 Example Usage Scenarios

We present example usage scenarios with real-world datasets from two application domains to showcase the utility of our approach.

7 Limitations and Future Work

8 Conclusion

In this paper,

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] C. Li and J. Wang. Efficiently mining closed subsequences with gap constraints. In *proceedings of the 2008 SIAM International Conference on Data Mining*, pp. 313–322. SIAM, 2008.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.