StageMap: Extracting and Summarizing Progression Stages in Event Sequences

Yuanzhe Chen[1], Abishek Puri[1], Linping Yuan[2], and Huamin Qu[1]

[1]Hong Kong University of Science and Technology
[2]Xi’an Jiaotong University

***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 of varying 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, a frequently occurring subsequence in the dataset. We introduce StageMap, 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 present a visual analytics system with four linked views, which are overview, tree view, stage view and sequences view. We also present case studies and discuss advantages and limitations of applying StageMap to real-world scenarios.**

***Keywords*-time series data; data transformation and representation; visual knowledge representation; visual analytics**

I. Introduction

Temporal event sequences 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.

The first visualization techniques specific to event sequences aimed to visualize the entire set of sequences [1]–[4]. However, such techniques are unable to scale with the number of sequences and the length of sequences. To remedy this issue, other works have focused on visualizing a summarized form of the data, thereby reducing the visual complexity of their systems. Recent work in this area involved mining the event sequences for frequent patterns and milestone events, then displaying them in a visual system. Such systems use established pattern mining models for this task. While such techniques reduce the visual complexity of the system, they have some key issues. First, such pattern mining models are often optimized using an objective function which does not take into account the visual form of the result. For example, the mining model might generate redundant patterns, thereby increasing the visual complexity of the system and potentially confusing the users. Second, only events that match the generated patterns are shown in the system, which means all unmatched events are discarded. Discarding events can mislead users about their data composition, thereby causing erroneous conclusions to be drawn.

Other work incorporated stage-based analysis to summarize the data. Stage-based analysis involves finding stages and their progression patterns in the dataset. A stage is a grouping of events, and the progression pattern of a stage is a sequence of stages that begins with the aforementioned stage. For example, in online learning click streams analysis, when students try to finish an online assignment, they may first browse the assignment, then review the course material and then ask questions on the course forum. If many students follow the same steps to finish the assignment, we can create a stage of these three events. In this way, all learning behaviors can be modeled as stages of the individual activities that form that behavior. We then can see whether some behaviors follow each other, i.e., whether stages are commonly seen after one another. Such an observation would form a progression pattern.

Visualizing sequences with stages and their progression patterns has its benefits. Since a stage is also a subsequence of events, the stage-based summary is, by definition, more concise than the event-based summary. Such stages can also contain high-level semantic information without increasing the cognitive load on the user. However, providing a scalable and meaningful visual summary for stage-based analysis is still challenging due to the following reasons: First, real world event sequence data often has thousands of distinct sequences and hundreds of distinct events. This level of complexity makes it difficult to extract an optimized set of patterns. Second, in large scale data analysis, it is difficult to directly drill down to individual sequences from the overview. Therefore, it requires a technique which has the flexibility to summarize progression patterns with different levels of detail. In general, the main issues in this field are related to the trade-off between limiting the loss of information and decreasing the visual complexity of the system.

In this paper, we propose a system that uses a novel implementation of stage-based analysis to create a visually concise, yet lossless system. To the best of our knowledge, this is the first work that attempts to tackle this problem in this way. We propose StageMap, an event sequence summarization method which presents sequences as a set of visually optimized stage progression patterns. Our method first extracts a set of frequently occurring stages, which we use to transform the original sequences into a set of progression patterns. We then model each progression pattern into a tree structure, where each node in the tree is a stage. The tree structure can aggregate similar stages and show how each stage branches into different subsequent stages. We then design a visual analytics system with four linked views to support visual analysis of the summary. We further test our method on 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.
- An algorithm which computes the visually optimized progression trees.
- An interactive visual analytics system which allows users to explore the summary of the data.

II. Related Work

A. Event Sequence Summarization

There are work that model event sequences as tree structures or graph structures. These approaches can be considered as one way to summarize event sequences. The complexity of the data is reduced by aggregating events of the same type 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 [5] proposes an algorithm to extract the tree structure with only high frequency events, which greatly reduces the size of the tree. However, 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 [6] uses treemap to show the hierarchical clustering results of sequences. Wang et al. [7] propose a technique to support unsupervised clustering and visualize the result with packed circles. Wei et al. [8] 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 work that generate a visual summary based on extracted frequent sequential patterns. TimeStitch [9] applies sequential pattern mining models to medical care data analysis and helps users to discover, construct and compare cohorts. Both Frequence [10] and Peekquence [11] 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. [13] propose a two-part representation to visualize both the sequential patterns and the 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 work do not consider the sequential relations among stages which limits the flexibility of presenting the inherent data structure.

Recently, EventThread [14] 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. However, their use of stages is static, as they create stages based on a constant time interval. Our method, on the other hand, dynamically creates stages, allowing for more control over the creation process of stage representations of sequences, thereby giving more control over the visual complexity of the data.

III. Summary of Stage Progression for Event Sequences

A. Problem Statement

Our idea is to visualize event sequences by extracting a set of stages and using this set to summarize the data into several stage progression patterns. A stage is a frequently occurring subsequence among the sequences and usually contains high-level structures. Fig. 1(a) shows an example with four event sequences (X_1 to X_4). Sequences can be represented as a series of stages. Two representations are shown in Fig. 1(b). To provide a concise and meaningful overview of the stages, we identify a set of stage progression patterns. In Fig. 1(c), the stage representation of sequences is converted into a tree, showing the progression patterns. In real-world applications, such as web clickstream analysis, these progression patterns can reveal certain human behavior which is critical for the analysis.

However, real world datasets are much more complex and noisy. To identify the user requirements, we collaborated with a group of experts in the field of online learning data analysis, including an instructor (E.1) and a teaching assistant (E.2) who wanted to improve their course design, as well as an education researcher (E.3) who wanted to understand the online learning behavior of the students. We collected the requirements and concerns about the current analytic tools from the experts. Based on the survey of existing works and exploration of real world datasets of different application domains, we further generalized the requirements of an effective visual summary of event sequences. We summarized three requirements as follows:

Minimizing visual complexity. Unlike existing pattern mining models, the proposed model should also consider the visual encoding of the summary and try to reduce visual complexity. The example in Fig. 1 illustrates why the non-visual based optimization can generate fragmented results. For example, when using coverage-based optimization, i.e., trying to only minimize the number of unmatched events, the

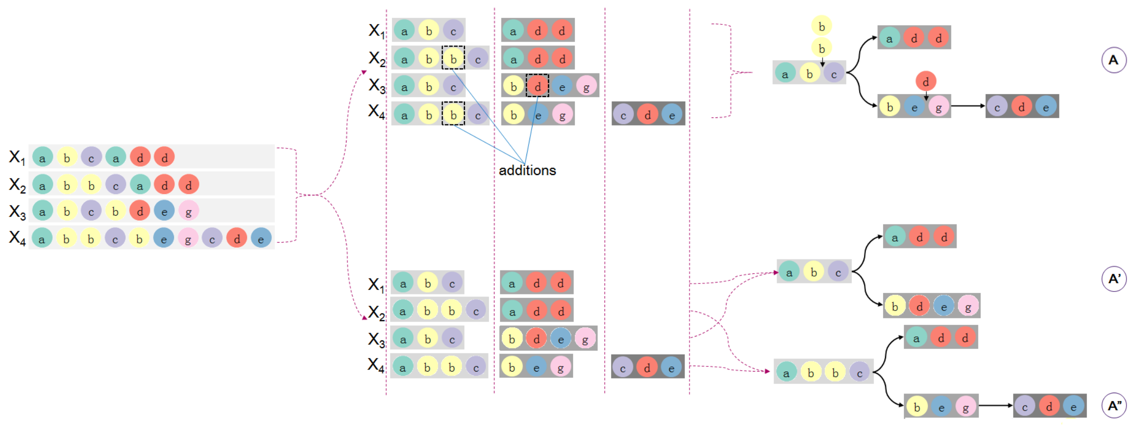

Figure 1. An example to illustrate the stage progression pattern in event sequences. A set of event sequences (a) is broken into stages (b) using a visually optimized covering (top) and a coverage optimized covering (bottom), resulting in two progression pattern representations for the original set of event sequences (c).

set of sequences in Fig. 1 ends up producing two separate trees A' and A'', while a visual-based optimization produces just one tree A. Without incorporating visual complexity into the cost function, when the dataset gets large, then the visual space will become cluttered.

Dealing with unmatched events. Most of the previous summarization methods discard the remaining events after stage extraction. In domain specific tasks, discarding such events risks misleading users. For example, in software application log analysis, a rare but fatal error may be discarded by the model but is important to the end users. Therefore, in the visual summary, it is important to make the users aware that there are addition events, and provide a way for the users to access such events.

Supporting interactive analysis. When analyzing large scale datasets, level-of-detail analysis is usually required to allow users to drill down to details step by step. Therefore, the model should be computationally efficient enough to update the summary interactively.

B. Algorithm

The proposed algorithm for summarizing progression patterns is called *SPTree*. Algorithm 1 describes the whole algorithm pipeline, which consists of three subroutines, that is, extracting stages, computing covers and building trees.

Taking 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 *VMSP* [15], a technique used for efficiently mining subsequences. We employ this technique mainly because it introduces wildcards when matching potential subsequences, which is consistent with our idea of allowing addition events in each stage. Furthermore, it runs faster than other approaches. We can set up a constraint Th_{gap} in *VMSP* to limit the maximum number of wildcards when generating stages. Then, *VMSP* will identify all the subsequences which have a frequency higher than a predefined threshold Th_{sup}. The resulting set of stages will be the set of stage candidates.

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 $\mathcal{T} = \{\mathcal{X}\}$, covering $\mathcal{C} = \{\mathcal{X}\}$, stage set $\mathcal{Q} = \{\}$;
2  stage candidates $\mathcal{S} = VMSP(\mathcal{X})$;
3  $sortS(\mathcal{S})$;
4  while $\mathcal{S} \neq \emptyset$ do
5      pop $S_i$ from $\mathcal{S}$;
6      for each $X_j \in \mathcal{X}$ do
7          $C_j = Cover(X_j, \mathcal{Q} \cup S_i)$;
8      end
9      $\mathcal{T}^* = BuildTrees(\mathcal{C})$;
10     if $Cost(\mathcal{T}^*) < Cost(\mathcal{T})$ then
11         $\mathcal{T} = \mathcal{T}^*$;
12         $\mathcal{Q} = \mathcal{Q} \cup S_i$
13     end
14 end
15 return $\mathcal{T}$
```

Given this set of stage candidates, the algorithm will search for a subset to construct the summary with. To do this, we need a cost function to evaluate the visual complexity of the visual summary, which is a set of trees. The proposed cost function, which is inspired by a recent work [13], is described in Eqn. 1:

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

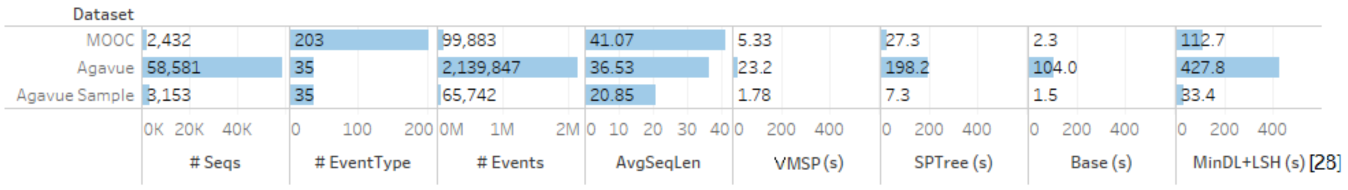

Figure 2. Statistics on two real-world datasets. The running time of *StageMap* is reported and compared with two sequence summarization techniques.

The two terms in Eqn. 1 correspond to the main visual elements in the summary, that is, the number of nodes in the trees and the number of addition events, respectively. The parameter λ is introduced to balance the importance of the two terms.

Given the cost function, a straightforward way to find the optimal set of stages is to search through all the combinations of stage candidates, and use the combination with the smallest cost to generate the result. However, this method has an exponential time complexity with respect to the stage number, i.e., $O(2^{|S|})$, making the computation too slow to be practically useful. To avoid this issue, *SPTree* uses a heuristic-based approach. Our heuristics involve favoring stages that are more frequent as well as of a longer length. We do this because, intuitively, the more frequent or longer a stage is, the more likely it is to reduce the visual complexity of the data. Therefore, the algorithm first sorts (subroutine *sortS*) the candidates by the multiplication of their frequency and length, then iteratively updates the summary by adding more stages from the sorted candidates. In this way, the algorithm becomes linear with respect to the stage number,i.e., $O(|S|)$. This is the core subroutine of *SPTree*.

In each iteration, the algorithm takes two steps (*Cover* and *BuildTrees*) to construct a summary with the updated set of stages. In the first step, the algorithm needs to determine where each stage is used, so as to encode a sequence, and where the wildcards occur. For each stage candidate S_i, the subroutine *Cover* replaces all occurrences of this stage with the symbol S_i. Then, the covering of the dataset can be generated by running *Cover* over each sequence in the dataset. To accelerate the algorithm, the matchings between stages and sequences can be stored as a lookup table when employing *VMSP*, so *Cover* only needs to check cells in the table. In the result of *Cover*, there will be a sequence of stages (the Cover) but there will also be some events that were not matched in the earlier step. These events are the addition events. Events that are matched in the earlier step are called matched events.

In the second step, *SPTree* builds a summary given the updated covering. After all the iterations finish, the summary with the smallest cost will be returned as the final result. Since we try to build a visual summary which highlights the branching patterns between stages, we construct the visual summary as a set of trees with each stage as a node. However, our algorithm can be readily adapted to construct other structures, as such a change would only apply to our *BuildTrees* Subroutine. (i.e., line 9 in Algorithm 1).

Parameter settings. There are three parameters (Th_{sup}, Th_{gap} and λ) that need to be set for the algorithm. Th_{sup}, the frequency threshold for *VMSP*, ranges from 0 to the number of sequences n. Its value cannot be too large, as this would result in a large number of stages being discarded by *VMSP*. However, if the value is too low, then the computation time becomes significantly longer. Th_{gap} ranges from 0 to the maximum length of sequences. Increasing the value of Th_{gap} allows a more flexible stage extraction of *VMSP*, however, increases the computation time. λ is used to balance the importance of patterns and additions, and ranges from 1 to the maximum length of sequences. In our experiments, we run the algorithm with different parameter settings. Based on the results, we set Th_{sup} to $0.1*n$, Th_{gap} to 1 and λ to 0.1.

Running time. We further evaluate the running time of the proposed method. To effectively compare our results to a baseline implementation, we build a baseline algorithm (*Base*). This algorithm computes a covering of the dataset in a similar way to *SPTree*, except it does not prune the set of stage candidates given by *VMSP*, so it will choose the covering which minimizes noise, making it a coverage optimizing algorithm.

Fig. 2 reports the running time for both datasets with the fixed parameters, and compare with *Base* and a recent sequence summarization technique called *Sequence Synopsis* [13]. From the results, we can see that *SPTree* needs more time to compute the optimized summary as compared to *Base*. However, *SPTree* is faster than the *Sequence Synopsis*.

IV. Visualization

A. Visual Encoding

Fig. 3 shows the system overview of StageMap. The system consists of four linked views, which are overview, tree view, stage view and sequences view.

1) Overview: As shown in Fig. 3(b), the overview directly takes the summary $\mathcal{T}$ as input and shows the overview. Since the summary $\mathcal{T}$ contains a set of trees, we present the summary with a space-filling representation and employ the treemap algorithm Nmap [16] to compute the layout. Inside each node of the treemap, we visualize these trees

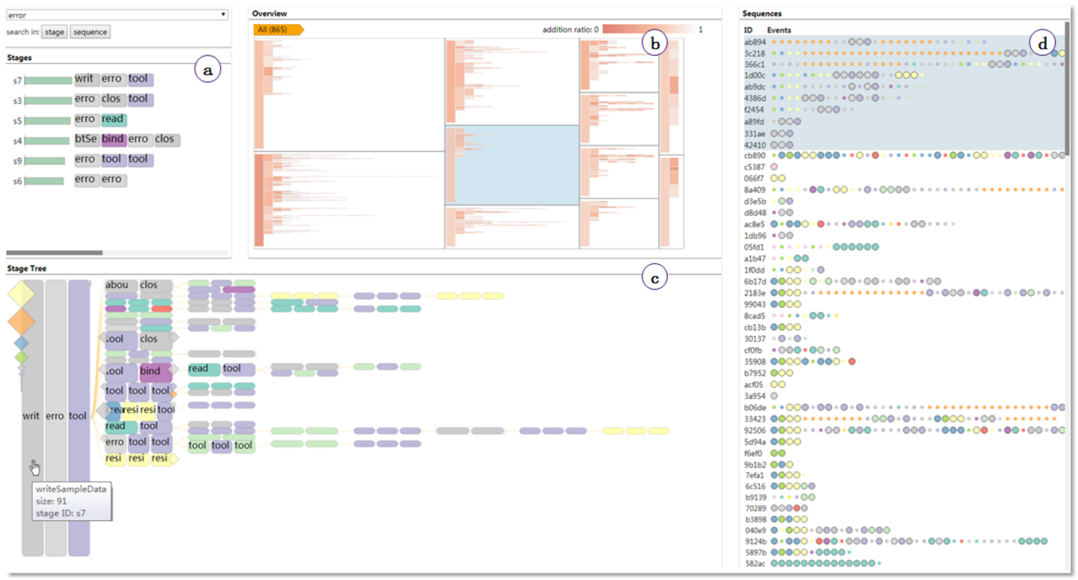

Figure 3. A screenshot of the StageMap visualization system applied to application log analysis. Our system includes four views: a stage view (a) displaying the extracted stages, an overview (b) showing the structure of each progression pattern, a tree view (c) showing the progression patterns of similar sequences with an icicle plot and a sequences view (d) listing each individual sequence. The usage case of this figure is described in Section V.

with icicle plots as long as the area of the node is large enough, otherwise we leave the node empty.

In the stage map, the area of each rectangle in the treemap encodes the number of sequences that match the progression pattern up to that rectangle. To fit the icicle plot inside the rectangle, we calculate an aspect ratio for each tree according to its depth and breadth. We then extend Nmap to generate a layout where each node has an aspect ratio close to the ratio of the tree in that node.

Inside an icicle plot, each rectangle represents a stage. In a traditional icicle plot, nodes in the same row have the same depth. However, in event sequences, the depth of a node encodes temporal information, which is normally visualized in a horizontal axis. Therefore, we rotate the icicle plot, so that nodes in the same column have the same depth. The height of each rectangle represents the frequency of that stage. Within the same column, we can rank the nodes by their frequency, timestamp of the first occurrence or the number of addition events. Here we choose not to encode the type of stages with different color. Instead, we save this visual channel to encode the type of events in other views.

Since we prefer to provide only high-level statistics of addition events to avoid information overload, we compute an addition ratio for each stage and visualize it with a sequential color scheme. The addition ratio is defined as:

$$R_{variant} = \frac{1}{1 + \frac{|addition|}{|S|}} \tag{2}$$

where $|addition|$ is the total number of addition events, and $|S|$ is the total number of events in the sequences.

2) Tree View: The tree view (Fig. 3(c)) displays a user selected node in the treemap and provides more details as compared to the overview. We again use the icicle plot to reduce the cognitive gap between the tree view and the overview. Inside each node of the plot, we fill the space with a stage glyph to show the composition of a stage and the addition events.

A stage glyph's constituent events are encoded as rectangles. We use a categorical color scheme to represent the event type. When the dataset consists of a large number of event types, the color scheme is used to encode the top 12 frequent occurring events, with the remaining events are encoded with a grayscale color. Users can also check the event type using the label in the rectangle or the tooltip. This color encoding is consistently used across the system. Besides, the users can also expand a subtree through interaction.

3) Sequences View: The sequences view is designed to aid the detailed analysis of each sequence. Within the view, each sequence is horizontally visualized as a chain of events, where each event is represented as a circle. The color and size of the circle represent the event type and whether the event is matched or not respectively. With respect to the size, there are two sizes; large and small. Large represents matched events while small represents addition events.

4) Stage View: The stage view vertically displays all the stages in the system sorted by frequency, with the highest frequency stage situated at the top of this view. In each row of this view, we first display the frequency with a bar chart, after which we display the name of the stage followed by the component events of the stage. This view provides users a high level summary of the stages in the data and aids the

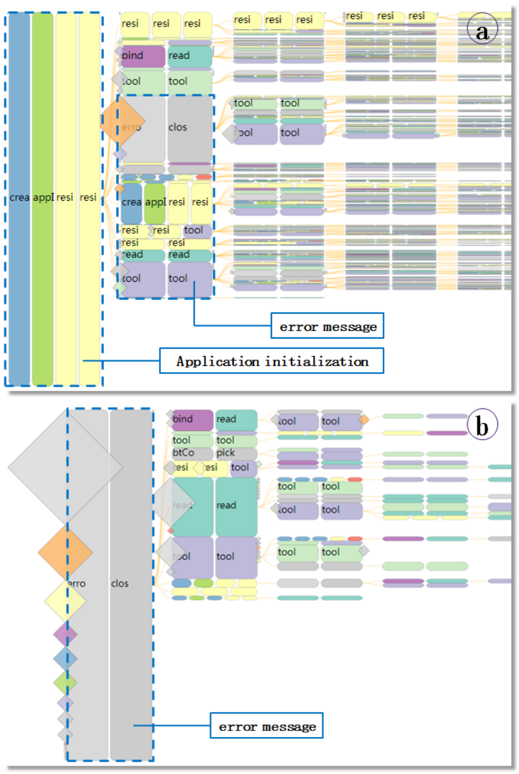

Figure 4. Two progression patterns in the Agavue Sample dataset. (a) shows the largest tree which starts with initialization of the application, and (b) shows the tree which starts with an error message of the application.

users in narrowing down their sequences of interest.

B. Interaction

To allow users to explore the data from various perspectives and enable level-of-detail analysis, StageMap is augmented with interactive techniques. Common interactions in event sequence visualization such as linked-highlighting, ranking, querying and tooltips are supported in our system. To allow the users to drill down to the summary of a subset of sequences, we designed two details-on-demand interactions to explore the visual summary. First, users can expand a node in the treemap. Second, users can query an event either in stages or in sequences. We believe these interaction techniques can help to demonstrate the effectiveness of the proposed visual summary. However, other interaction techniques can be easily embedded as needed.

V. Usage Scenario

In this section, we present results from a usage scenario with real world dataset. The statistics of the dataset are shown in Fig. 2. We implement the system as a web-based application and conduct interviews with domain experts.

In our usage scenario, we applied StageMap to Agavue Sample dataset. The dataset records user interactions in a data visualization application, where users try to generate visualizations such as bar charts and treemaps. The analyst aimed to understand users behavior with the main goal of being to improve usability. The insights gained from the analysis can help the analyst to improve the UI design to ensure a smooth experience.

The analyst first loaded the data, after which he started the analysis by looking at the overview (Fig. 3(b)). In total there were 11 nodes of varying size in the overview. To first get a sense of what most users do, the analyst clicked on the largest node, i.e., the one with the highest number of represented sequences. This interaction updated the tree view, which now displayed an icicle plot showing all the progression patterns for the root stage of this tree (Fig. 4(a)), which turned out to be the stage [*create*, *appInit*, *resize*, *resize*]. The analyst, as a domain expert, knew that this corresponds to the starting of the application, so it was expected that this would be a common starting stage.

The analyst now looked at the progression patterns of this stage. One progression path had the stage [*error*, *errorCloseBox*], now referred as error stage, right after the root stage. The analyst knew that this meant that the system had thrown an error message. Using the tooltip, he saw that 18.5% of all sequences in this subtree were in this progression pattern. This was a key observation that the analyst noted, as he mentioned that this indicated there were some issues with how they explained the creation of an app.

Having noticed this error-causing issue, the analyst now wanted to further investigate the occurrence of the error stage. In particular, he wanted to know when users experienced errors at the beginning of their sessions. To do this, he went to the stage view and used our querying feature to find all stages that contained errors. After this query was processed, only the error stage was shown. Using the linked-highlighting feature, he identified the corresponding node in the overview and clicked it, updating the tree view (Fig. 4(b)).

Upon looking at the icicle plot, he first noticed that the two largest non-error addition events before the root stage were *slider* (orange color) and *resize* (creme color). As these were common events, the analyst wanted to see in detail what happened in those sessions. To do this, he looked at the sequences view, which had automatically updated to only show the sequences represented in the tree view. He noticed that sessions with a long chain of slider events (orange circles) ended up having an error. The analyst mentioned it as further improvements of the design.

Overall, the analyst was able to use the system to do a high level analysis of the sequences, after which he could focus on a specific point of interest. The benefit of our system is the ability to seamlessly switch from a high-level summary of all sequences via the overview to a focused analysis of a particular stage or event. More importantly, the fact our system keeps all events without discarding allows the analyst to see the buildup to and aftermath of a stage, increasing the amount of information that can be garnered without a large increase in complexity. In other systems, such a task involves going back to the raw data to investigate what happens around a pattern, which is a significantly more complex task.

VI. Conclusion

In this paper, we presented a novel approach for analysts to explore event sequence data by visualizing the data with a set of stage progression patterns. First, we proposed a novel algorithm to compute the optimized summary of the data. We then proposed a comprehensive visual analytics system which takes the computed summary as input. Our system supports levels-of-detail data exploration. We also conducted case studies to demonstrate the effectiveness of our approach. The limitations of our work were finally summarized, with a discussion of future research directions. In the future, we also plan to extend the tree structure and *VMSP* to further improve the scalability and time efficiency of our approach.

Acknowledgment

We would like to thank all the reviewers for their valuable comments. This project is partially supported by RGC GRF 16213317.

References

[1] 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.

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

[3] 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*, vol. 19, no. 12, pp. 2080–2089, 2013.

[4] 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*, vol. 15, no. 6, 2009.

[5] Z. Liu, B. Kerr, M. Dontcheva, J. Grover, M. Hoffman, and A. Wilson, "Coreflow: Extracting and visualizing branching patterns from event sequences," in *Computer Graphics Forum*, vol. 36, pp. 527–538, Wiley Online Library, 2017.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[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*, vol. 23, no. 1, pp. 321–330, 2017.

[13] Y. Chen, P. Xu, and L. Ren, "Sequence synopsis: Optimize visual summary of temporal event data," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 1, pp. 45–55, 2018.

[14] 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*, vol. 24, no. 1, pp. 56–65, 2018.

[15] P. Fournier-Viger, C.-W. Wu, A. Gomariz, and V. S. Tseng, "VMSP: Efficient vertical mining of maximal sequential patterns," in *Canadian Conference on Artificial Intelligence*, pp. 83–94, Springer, 2014.

[16] F. S. Duarte, F. Sikansi, F. M. Fatore, S. G. Fadel, and F. V. Paulovich, "Nmap: A novel neighborhood preservation space-filling algorithm," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 2063–2071, 2014.