Vision, Modeling, and Visualization (2018)
F. Beck, C. Dachsbacher, and F. Sadlo (Eds.)

Clustering for Stacked Edge Splatting

M. Abdelaal[1], M. Hlawatsch[1], M. Burch[2], and D. Weiskopf[1]

[1]Visualization Research Center (VISUS), University of Stuttgart, Germany
[2]TU Eindhoven, Netherlands

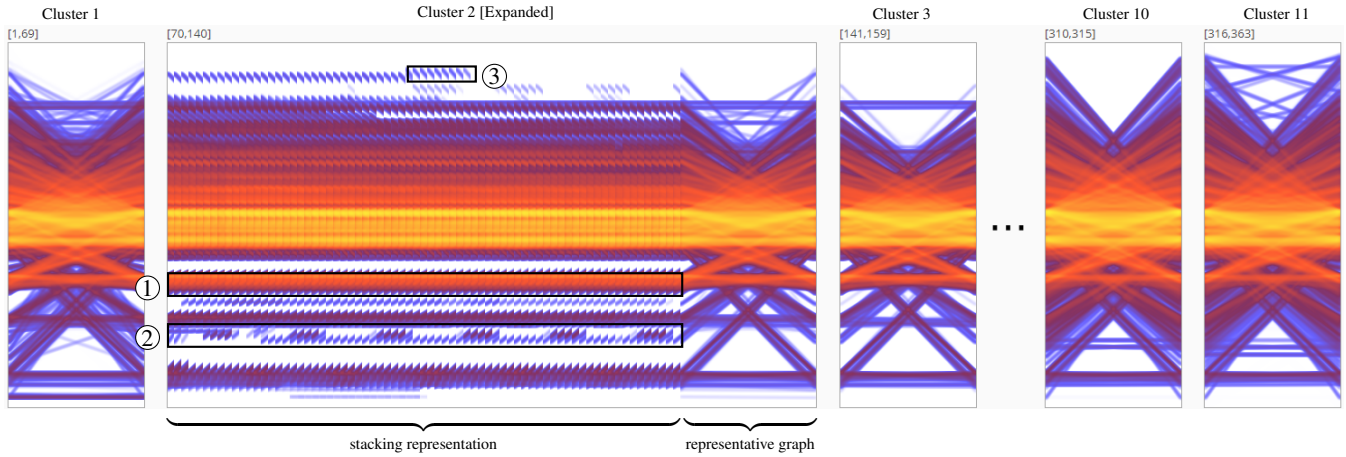


Figure 1: *A dynamic graph visualization depicting the US domestic flight dataset from October 1st, 1987, to December 31st, 2017. The data is aggregated on a per-month basis. It contains 402 vertices, which are the airports, 1,300,340 weighted edges, which are the flight connections and their frequencies, and 363 timepoints, which are given by the graphs per month. The dataset is sequentially clustered at a threshold of 1.1, resulting in 11 clusters. As annotated, the stacking representation allows us to identify several temporal patterns: (1) Stability, (2) Periodicity, and (3) Anomaly, whereas the clustering allows us to identify different temporal phases of the graph.*

Abstract

We present a time-scalable approach for visualizing dynamic graphs. By adopting bipartite graph layouts known from parallel edge splatting, individual graphs are horizontally stacked by drawing partial edges, leading to stacked edge splatting. This allows us to uncover the temporal patterns together with achieving time-scalability. To preserve the graph structural information, we introduce the representative graph where edges are aggregated and drawn at full length. The representative graph is then placed on the top of the last graph in the (sub)sequence. This allows us to obtain detailed information about the partial edges by tracing them back to the representative graph. We apply sequential temporal clustering to obtain an overview of different temporal phases of the graph sequence together with the corresponding structure for each phase. We demonstrate the effectiveness of our approach by using real-world datasets.

CCS Concepts

• ***Human-centered computing*** → ***Information visualization; Visual analytics;***

1. Introduction

Dynamic graphs build an important data structure for many application domains. For example, in software engineering, the dynamics of call relations [Die07] might be worth investigating in order to understand the internal processes of a software system and to possibly detect design flaws, prevent bugs, or maintain the system. Similarly, co-author networks formed within scientific communities can change over time; analyzing those networks is essential to understand the information flows and relationships between researchers and how they evolve over time. There are many of those examples and from a data perspective, all of them are commonly based on vertices, edges, and timepoints. In many cases, additional attributes are attached, typically leading to some kind of multivariate graph structure [KPW14].

DOI: 10.2312/vmv.20181262

Designing a visualization approach that is scalable in the three data dimensions is difficult but important for data analytics. To reach this goal, an overview of the evolving graph structure is required. Such an overview should uncover temporal patterns (i.e., stability, periodicity, trends, or anomalies), and temporal phases (i.e., periods of time where graphs share similar features) that exist in the data and, therefore, provide a starting point for further data exploration tasks.

The concept of parallel edge splatting [BVB*11] depicts dynamic data as a sequence of static bipartite layouts. Therefore, it follows the principle of dynamic stability [DG02] and mental map preservation [MELS95], making it a good candidate for a time-scalable dynamic graph visualization. Moreover, pushing the individual graphs together by an interleaving them [BHW17] makes the visualization even more time-scalable. However, it comes at the cost of merging edges and edge attributes, making it sometimes hard to identify temporal patterns over longer time spans.

In this paper, we overcome some of the formerly mentioned issues by providing a visualization approach capable of depicting and analyzing temporally long graph sequences. The main contributions of our work are: 1) introducing the *stacked edge splatting* representation to avoid the over-drawing problem caused by the interleaving method [BHW17] and, therefore, uncover the temporal patterns, 2) applying sequential temporal clustering to obtain an overview of different graph temporal phases together with the corresponding graph structure for each phase. To evaluate our approach, we present two application examples based on a US domestic flight traffic dataset [Uni18] and a software call graph dataset [JHo18].

2. Related Work

Visualizing and analyzing dynamic graphs is a challenging field of research. Many of the existing techniques are surveyed by Beck et al. [BBDW17]. The interesting aspects of this kind of data are the combination of several data dimensions that should be included in a corresponding dynamic graph visualization: edges, vertices, and timepoints [BETT99]. Additional attributes like weights or hierarchical organizations of the vertices can complicate the visual depiction of this kind of data. Moreover, a dynamic graph representation should provide a scalable variant for most of the data dimensions while still preserving the graph structure and topology when showing the temporal evolution of the graph.

Beck et al. [BBDW17] discuss two major concepts for displaying the time dimension in order to analyze the evolution of the graph structure over time. Time-to-time mappings make use of graph animation [FT04, DG02] and are typically based on node-link diagrams. Time-to-space mappings [BVB*11, BHW17] exploit the given display space to show as many graphs as possible next to each other with the goal to provide an overview of the time-varying behavior while still showing most aspects of the graph structure.

Although the animation concept can be used to visualize dynamic graphs, it typically suffers from mental map preservation problems [APP11, MELS95] if the graphs do not follow dynamic stability criteria. Moreover, comparing several graphs over time is rather difficult with animation [TMB02]. For this reason, much of the recent research in dynamic graph visualization focuses on static representations of the dynamic data, making it comparable over time due to perceptual benefits [HE12].

With these design issues in mind, the parallel edge splatting technique [BVB*11] focuses on a time-scalable variant for representing dynamic graphs by placing the graphs in a bipartite layout next to each other in a static diagram (apart from interaction techniques). The problem of visual clutter [RLMJ05] caused by many link crossings is mitigated by computing density fields from the graph edges similar to the work of van Liere and de Leeuw [vLdL03], who splatted the nodes instead. Color coding these density fields provides an overview of several graphs in a sequence, making them comparable while still showing the graph structure. Burch et al. [BHW17] extended this idea by interleaving, obtaining an even more time-scalable variant that scales up to more than a thousand timepoints.

The visual fusion resulting from interleaving emphasizes the graph structures. This method, however, suffers from over-drawing problems, as a result of the overlap between the individual graphs. This leads to merging of edges and edge attributes, making it hard to identify the length of periods in which graph edges persist, hence leading to data misinterpretations. Burch [Bur17] tried to tackle this problem by showing a splatted sequence of partially drawn [BEW95, BVKW11, BCG*12] bipartite layouts, showing each individual graph in a fixed stripe. However, in this case, the link information is hard to interpret and target node ambiguities mostly occur. Moreover, temporal clustering is not used as additional support for identifying temporal phases in dynamic data.

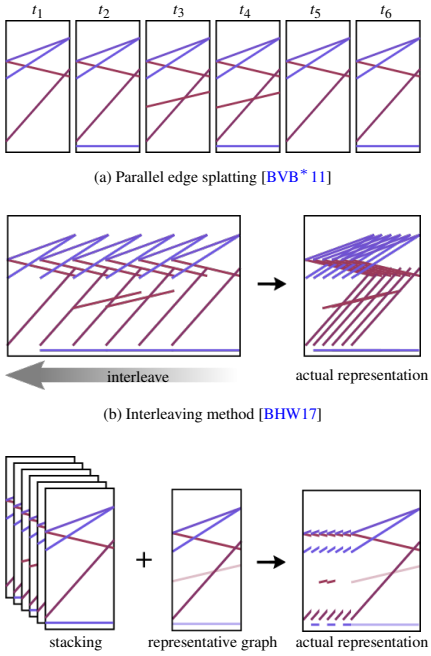
In our work, we focus on providing a *time-scalable overview* that reveals *temporal patterns* and *temporal phases* that exist in dynamic graph data without losing the *graph structural information*. Van den Elzen et al. [vdEHBvW16] tried to achieve time-scalability by reducing graphs to points, but the actual graph structures were not visible anymore. Also, massive sequence views [vdEHBvW13] are time-scalable representations that support different clustering and reordering techniques. However, there is no temporal clustering and the many parallel lines make a graph structure exploration difficult. Moreover, the matrix cube visualization by Bach et al. [BPF14] that is based on stacked adjacency matrices is hard to use [GFC05] to identify temporal patterns in long and dense dynamic graphs.

3. Visualization Technique

Our visualization approach builds on the concept of parallel edge splatting [BVB*11], which uses a sequence of bipartite graphs to visualize dynamic graphs. To design a visualization technique capable of depicting and analyzing temporally long graph sequences, we horizontally stack bipartite graphs by drawing partial edges, what we refer to as *stacked edge splatting*. By doing this, we avoid the over-drawing problem caused by the interleaving method [BHW17] and, therefore, uncover the temporal patterns. The graph structural information is preserved by the *representative graph*, where edges are aggregated and drawn at full length. To get the detailed information about the partial edges (i.e., direction or target node), we trace them back to the representative graph. We apply sequential temporal clustering to the entire graph sequence

to obtain a less-cluttered representative graph. On the one hand, it improves the edge tracing task. On the other hand, it provides an overview of different temporal phases of the graph together with the corresponding graph structure for each phase.

3.1. Stacked Edge Splatting



(a) Parallel edge splatting [BVB*11]

(b) Interleaving method [BHW17]

(c) Our approach: stacked edge splatting

Figure 2: *Stacking bipartite graphs to obtain a time-scalable visualization. (a) Parallel edge splatting: bipartite graphs are drawn next to each other. (b) Interleaving method: bipartite graphs are interleaved. (c) Our approach: bipartite graphs are horizontally stacked by drawing partial edges.*

Figure 2 (a) shows traditional parallel edge splatting after transforming the individual directed graphs into corresponding bipartite layouts and visualizing them next to each other. The graph vertices are hierarchically clustered and then reordered to obtain a good visual representation. For hierarchical clustering, we use the Jaccard similarities to compute the number of shared links for all vertex pairs. We then apply a vertex reordering technique [iS98] to find a better arrangement of vertices while preserving the hierarchical relationships between vertex-clusters.

With the aim of obtaining a more time-scalable visualization, Burch et al. [BHW17] introduced the interleaving method where the individual graphs are pushed together by interleaving them. The visual fusion resulting from the interleaving emphasizes the internal graph structures. The method, however, suffers from over-drawing problems, as a result of the overlap between the individual graphs, making it hard to identify the length of periods in which graph edges persist and, therefore, hiding the temporal patterns. As shown in Figure 2 (a), the bottom-most edge exists only at three timepoints (t_2, t_4, and t_6). However, the resulting visualization in Figure 2 (b) leads to the false impression that the edge is persisting from timepoint t_2 to timepoint t_6.

To avoid the over-drawing problem caused by the interleaving method without losing the time-scalability advantage, we horizontally stack the bipartite graphs by drawing partial edges (see Figure 2 (c)). To recover the lost information about the graph structure caused by drawing partial edges, we added the representative graph, where edges are aggregated and drawn once at full length. The representative graph is then placed next to the last graph in the sequence (see Figure 2 (c)).

With this design, we use the representative graph to depict the graph structure, whereas the stacked edge splatting representation reveals the temporal patterns that exist in the data. By drawing the full-length edges only once in the representative graph and placing it at the end of the sequence, we prevent the graph structural information from interfering with the temporal patterns. To obtain the edge information (i.e., direction and target nodes) that corresponds to the identified temporal pattern, we follow the partial edges across the entire sequence until we reach the representative graph, what we refer to as *edge tracing*.

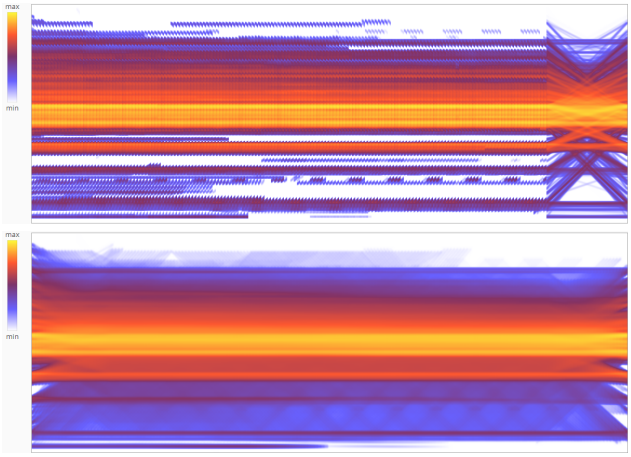


Figure 3: *First 159 timepoints of the flight dataset visualized using stacked edge splatting (top) and the interleaving method (bottom). Temporal patterns are more recognizable in the stacked edge splatting representation as a result of avoiding over-drawing problems caused by the interleaving method.*

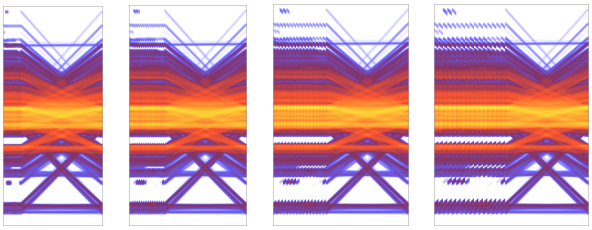
Figure 3 shows a visual comparison between our stacked edge splatting technique (at the top) and the interleaving method (at the bottom); we choose to visualize the first 159 timepoints of the flight dataset. As one might notice, both methods have analogous time-scalability. The representative graph in the top visualization allows us to obtain an overview of the graph structure. The same information can be obtained in the bottom visualization by observing the entire graph sequence. However, the temporal patterns are more recognizable in the top visualization as a result of avoiding the over-drawing problem.

As shown in Figure 3, we use a multi-hued color scale consisting of five colors (white, blue, purple, orange, and yellow). The multi-hued color scale is essential for visualizing the transformed pixels

scalar weights resulting from the edge splatting technique. We additionally apply a logarithmic transformation to the pixels weights to reduce the dynamic range of the final image. To de-emphasize the edges with lower weights (less frequent), we choose the color of the minimum value to be the same as the background color (white).

The edge tracing task is influenced by the occurrence frequency of the edge over the entire sequence. Edges that occur for a short period of time (i.e., outliers), are harder to trace and usually not visible in the representative graph. We address this problem in two ways. First, we apply temporal clustering to obtain a better representative graph (see Section 3.2). Second, we implemented a mouse-hover interaction technique to expand the individual graphs to the full width (see Figure 9). This technique visualizes the graph's changing structure as an animation when the users move the mouse cursor over the entire graph.

An additional factor that affects that edge tracing task is the length of the partial edges. Shorter partial edges result in a compressed representation that consumes less screen space, whereas longer partial edges provide more edge information. Figure 4 shows examples with different lengths of partial edges. We experimentally identified 5 pixels length as a good value because the space requirements are not too large and the edge information is still recognizable. Nevertheless, the users can adjust the length using an interactive slider.



(a) 1 pixel (b) 3 pixels (c) 5 pixels (d) 7 pixels

Figure 4: *Stacked edge splatting with different lengths of partial edges. Shorter partial edges result in a more compressed representation, whereas longer ones provide more edge information.*

Since the bipartite graph restricts the layout of vertices to one dimension, it increases the visual clutter caused by edge crossings. Therefore, it is hard to obtain the precise edge information only by looking at the representative graph, especially, when graphs are very dense. For this reason, we support mouse interaction to highlight the graph edges corresponding to the hovered vertex (see Figure 7).

3.2. Temporal Clustering

Using a single representative graph to depict the graph structure for the entire graph sequence results in a cluttered representation caused by many edge crossings, especially when the dynamic graph data is changing frequently from one timepoint to another. Obtaining an overview of the graph structure from such representation is a challenging task. To address this problem, we apply sequential temporal clustering, allowing timepoints that share the same graph structure to be grouped together in one cluster. In this way, each cluster is a subsequence of the original graph sequence, reflecting a certain graph structure. The time interval covered by each cluster is the interval between the first and the last timepoints within the cluster, what we refer to as *temporal phase*.

By having multiple representative graphs, one per each cluster, we obtain an overview of different temporal phases in the graph sequence together with the corresponding graph structure for each phase. It also improves the edge tracing task, since the edges of individual graphs are better depicted in the cluster-representative graph.

Similar to the work of Bach et al. [BHRD*15], we employ a distance-based clustering method that achieves good results at low computational cost. Clustering is done sequentially by computing the Euclidian distance d between the two adjacency matrices $m(t_i)$ and $m(c_j)$ for the current timepoint t_i and the aggregated timepoint of the current cluster c_j. Given a user-defined threshold p, and if $d < p$, t_i is added to the cluster c_j. If $d >= p$, a new cluster is created for t_i. We implemented a semi-automatic clustering where the users can experimentally try different threshold values in an interactive way. High thresholds result in fewer clusters, whereas lower thresholds result in more and smaller clusters but with a higher similarity within each cluster. The users can optimize the results by splitting/merging clusters manually.

By default, each cluster is depicted by the cluster representative graph. We refer to this view as the *collapsed view*, where individual graphs within the cluster are not shown. By doing this, we save screen space and provide the users with an overview of different graph-topological states without going into detail. In contrast, in the *expanded view*, users can explore the stacking representation within each cluster (see Cluster 2 in Figure 1) and exploit the mouse-hover interaction to expand individual graphs to the full width. The users can switch between both views by clicking on the cluster representative graph.

When graphs are very dense, it is hard to detect visual differences between different clusters by comparing the representative graphs. For this reason, we provide an algorithmic comparison to detect the differences between different temporal phases in terms of added and deleted edges (see Figure 5). The set of edges $E^{add}_{G_i}$ that are introduced in the graph G_i is expressed by:

$$E^{add}_{G_i} = E_{G_i} \setminus E_{G_{i-1}}$$

In contrast, the set of edges $E^{del}_{G_i}$ that are deleted in the graph G_i is expressed by:

$$E^{del}_{G_i} = E_{G_{i-1}} \setminus E_{G_i}$$

Here, $i \in \mathbb{N}$ denotes the i-th timepoint, i.e., the i-th graph in the graph sequence, E_{G_i} denotes the edge set of graph G_i.

4. Application Examples

In this section, we demonstrate the applicability and usefulness of our approach. We present two different application examples based on two different dynamic graph datasets: a US domestic flight traffic dataset [Uni18] and a software call graph dataset [JHo18]. The US domestic flight dataset demonstrates the scalability of the technique with respect to the number of edges, whereas the software

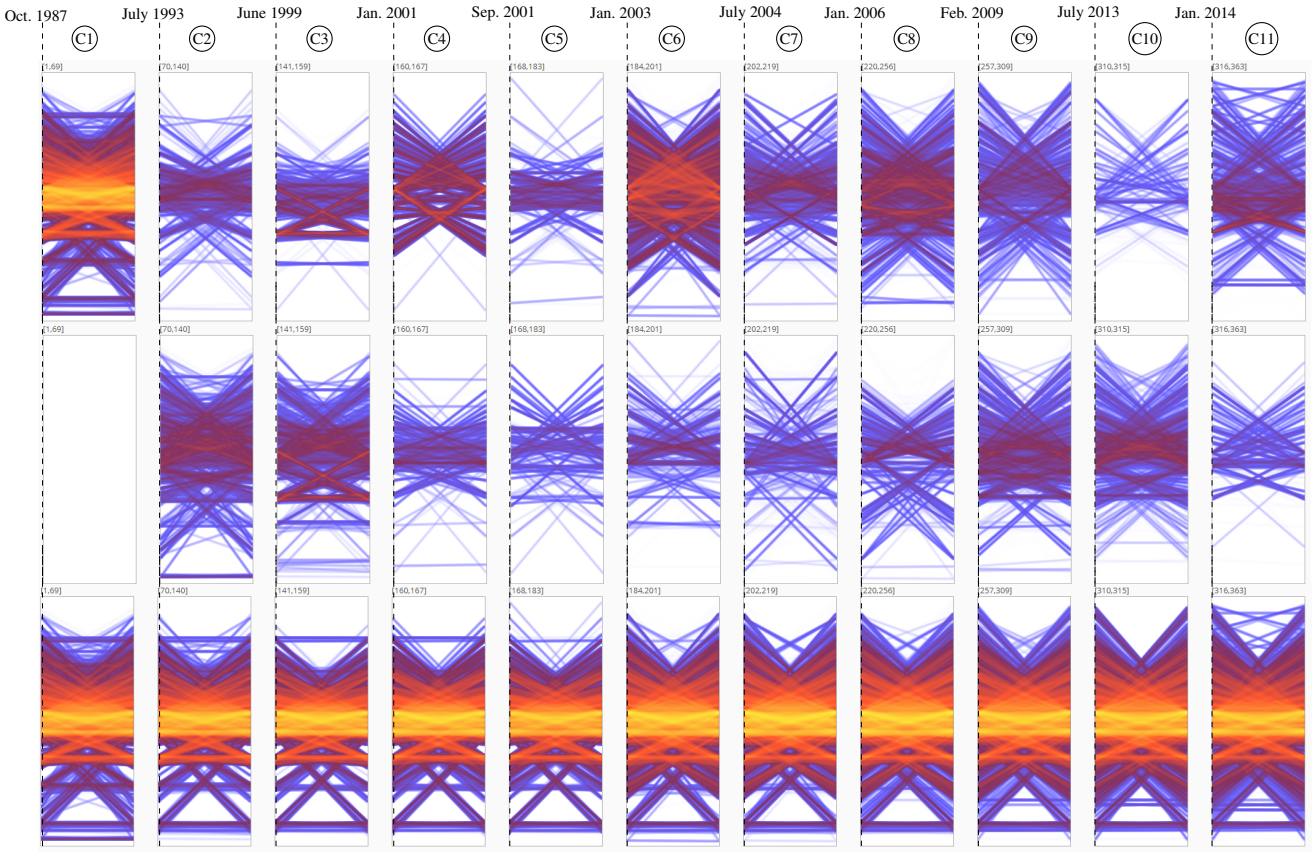


Figure 5: *Sequentially clustering the flight dataset at a threshold of 1.1, resulting in 11 clusters: (bottom) the clusters' representative graphs, (middle) links deleted by each cluster, (top) links added by each cluster. Each of the representative graphs at the bottom is a result of adding the links at the top to the previous representative graph, followed by subtracting links at the middle.*

call graph dataset shows the scalability of our visualization technique with respect to the number of timepoints. For each example, we show how the visualization allows us to identify different graph structures and temporal patterns.

4.1. US Domestic Flight Dataset

We extracted the flight data for 30 years starting from October 1st, 1987, to December 31st, 2017. The data is aggregated on a per-month basis, it contains 402 vertices, which are the airports, 1,300,340 weighted edges, which are the flight connections and their frequencies, and 363 timepoints, which are given by the graphs per month.

We apply temporal clustering to identify different temporal phases in the flight data. We experimentally identified 1.1 as a good threshold value, resulting in 11 different clusters as shown in Figure 5 (bottom). Each cluster is depicted by the cluster representative graph and marked by the time interval at the top. By looking at the representative graphs in Figure 5 (bottom), we notice that the middle region is rather similar across all clusters. However, we can observe some visual differences as we approach the top and the bottom regions.

For example, in the top region, we notice horizontal connections that existed between the years 1987 and 2001 (clusters C1 to C4) and started to fade-out in the years 2002 and 2003 (clusters C5 and C6), until they disappeared between 2004 and 2017 (clusters C7 to C11). Also, we see crossing links in clusters C6, C7, C8, C9, and C11. Some of these links existed in cluster C1. Then, they nearly disappeared between clusters C2 and C5 until they returned with high frequency starting from cluster C6. In the bottom region, we can detect similar behavior with the connections that come from the bottom of the graph toward the middle. Additionally, we notice these horizontal connections at the bottom that are only introduced in clusters C9, C10, and C11. These horizontal connections are not a result of self-connections, but they are connections between two adjacent airports on the vertical axis.

To investigate these changes in more detail, we use the difference feature to calculate the added and deleted connections introduced by each temporal phase. Figure 5 (top) shows the added links introduced by each cluster, whereas Figure 5 (middle) shows the deleted ones. In this arrangement, each of the representative graphs at the bottom is a result of adding the links at the top to the previous representative graph, followed by subtracting links at the middle. In this

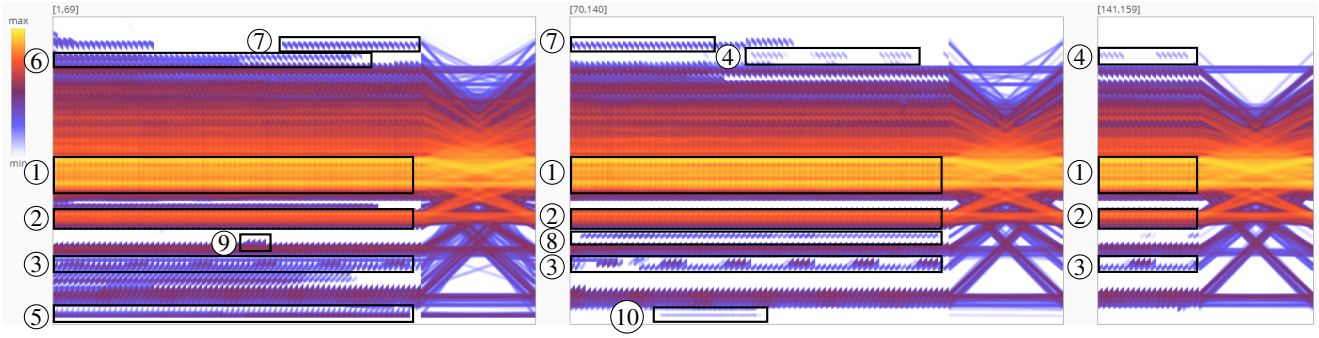


Figure 6: *The expanded view of the first three clusters of the flight dataset. The stacking representation allows us to identify several temporal patterns.*

way, we trace the evolution of the graph by going through Figure 5 column-by-column and reading top-down within each column.

As one might notice, there are many changes that occur in the middle region and that are hardly observed by only looking at the representative graphs at the bottom. For example, the added connections in cluster C3 reflect new flights between *Austin – Bergstrom International* airport and 14 other airports distributed over the East and the West of USA. We notice the similarity between the added and deleted connections in cluster C3, this due to the periodic behavior of some of these connections within the same cluster. The airports of *Dallas/Fort Worth International*, *Chicago O'Hare International*, *John F. Kennedy International*, and *LaGuardia* are the center of the change in cluster C4. Each airport runs new flights to relatively small airports located in the same or an adjacent state.

The big change at cluster C6 reflects many new connections between East Coast airports, most notably, the airports of *Hartsfield-Jackson Atlanta International*, *Chicago Midway International*, *Cincinnati/Northern Kentucky International*, and *Washington Dulles International*. It also reflects new flights that connect *George Bush Intercontinental/Houston* airport with other neighboring airports. Additionally, it involves new connections between *Denver International/Colorado* airport and the main airports located in the northern states of the USA. The new links in cluster C11 involve new flight connections between the Eastern airports (Chicago, Washington, and Florida) and the Western airports (San Francisco, San Diego, Portland (Oregon), and Seattle) through *Dallas Love Field* airport at the center of USA. We notice the similarity between the added connections of cluster C11 and the deleted connections in cluster C9.

Expanding the clusters allow us to see the stacking representation where we can identify the temporal patterns and trace them back to the representative graphs. Figure 6 shows the expanded view of the first three clusters in Figure 5. As we explore the graphs, we identify several temporal patterns. We explain some of these patterns as follows:
Stable Patterns: Patterns 1 and 2 reflect a high number of flight connections that run over the entire graph period. The busiest airports in the US typically belong to that group.
Periodic Patterns: Pattern 3 shows flight connections that occur every year between December and April. In contrast, pattern 4 shows flight connections that occur between May and October.
Shift Patterns: Patterns 5 and 6 reflect frequent flight connections that stopped by the year 1993. In contrast, pattern 7 shows one flight connection that frequently occurred between the years 1992 and 1996, whereas pattern 8 shows another flight connection occurred between the years 1994 and 1999.
Anomalies: Patterns 9 and 10 show unusual flight connections that occurred only for a short period of time.

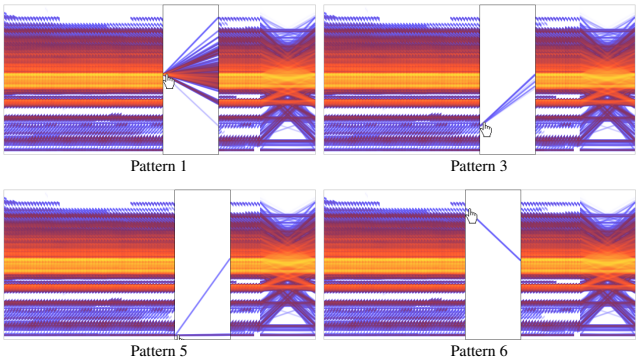


Figure 7: *The edge-highlighting interaction technique is used to view the source and destination vertices of the previously identified temporal patterns.*

To identify the source and destination vertices of these patterns, we trace the partial edges corresponding to each pattern back to the representative graph (i.e., patterns 1, 2, and 5). To obtain a less cluttered view, we utilize the edge-highlighting interaction technique to show only the edges corresponding to the hovered vertex as shown in Figure 7. We refer to the supplementary materials to view the figures in the paper. The figures are best viewed on a monitor.

4.2. Software Call Graph Dataset

The software call graph dataset contains 787 vertices, which are the software functions, 25,906 weighted edges representing the call relations during the program execution, and 1,077 timepoints. The call graph models the call relations between program functions dur-

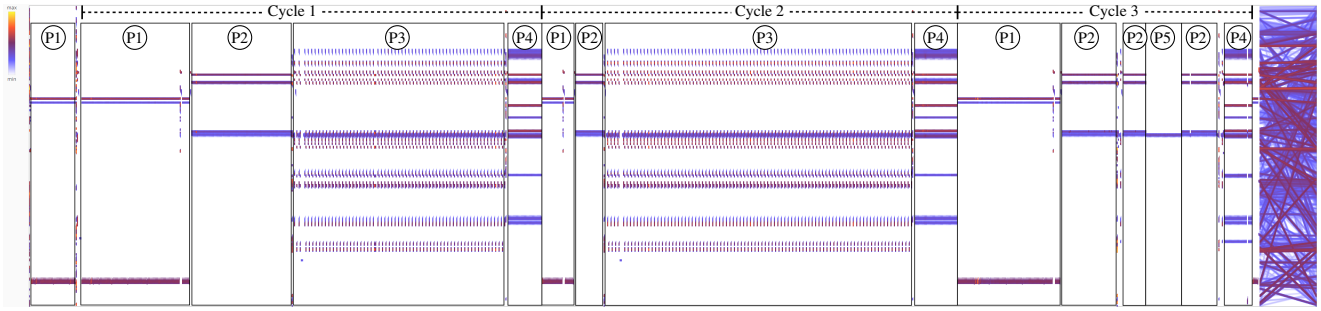


Figure 8: *Visualization of the software call graph dataset using our approach: Five different patterns can be identified along with the cyclic behavior starting from timepoint 47.*

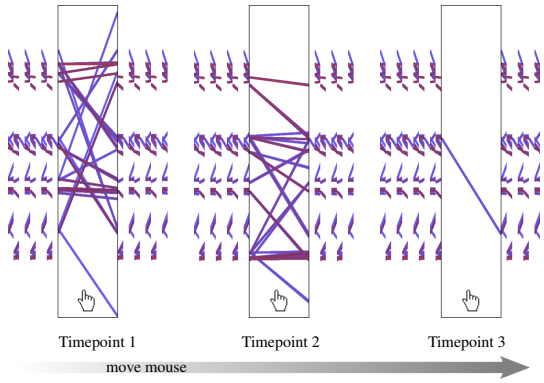


Figure 9: *Closeup segment of pattern P3 shows a dynamic behavior that keeps altering between three distinct timepoints. The mouse-hover interaction technique is used to expand the timepoints to the full width.*

ing runtime. By visualizing the call graphs we could examine and analyze how the software is performing.

We choose to visualize the entire dataset without applying temporal clustering. Figure 8 shows the visualization using our approach. Since the dataset contains 1,077 timepoints, we adjusted the partial edge length to two pixels, so that the screen space is enough to depict the entire graph sequence. Similar to the flight dataset, the periodic patterns are clearly visible by looking at the resulting graph. We notice that the software dataset is rather sparse compared to the flight dataset. Since we did not apply temporal clustering, the representative graph contains visual clutter caused by many edge crossings, making it hard to obtain an overview of the graph structure.

By exploring the graph in Figure 8, we can identify five different patterns: P1, P2, P3, P4, and P5. Also, we notice the cyclic behavior starting from timepoint 47. The patterns cycle exhibits the following order: P1 $\rightarrow$ P2 $\rightarrow$ P3 $\rightarrow$ P4. The complete cycle occurs twice between timepoints 47 and 812. However, at the third time, we can observe a different order of patterns along with a newly introduced pattern (P5). We also notice the existence of single timepoints when switching from one pattern to another. We refer to those single timepoints as *transition timepoints*. Moreover, by looking closely at pattern P3, we observe its dynamic behavior that keeps altering between three distinct timepoints. We use the mouse-hover interaction technique to expand the three timepoints in a full-width as shown in Figure 9.

To investigate the cyclic behavior in more detail, we choose to cluster the graph sequentially. Instead of using a threshold, we split the graph manually at the transition timepoints. Figure 10 shows the representative graphs of the resulting clusters. Each transition timepoint is a cluster that has a single timepoint. As we explore the first cycle, we notice three different transition timepoints (T1, T2, and T3). Transition T1 occurs between patterns P1 and P2. Transition T2 occurs between patterns P2 and P3. Transition T3 followed by transition T1 both occur between patterns P3 and P4. In contrast, we notice that pattern P1 directly follows pattern P4 with no transition timepoints in-between. The second cycle of the graph follows the same behavior as the first one, except for transition T1 being introduced after transition T2 between the patterns P2 and P3.

The third cycle, however, shows a different behavior, in comparison to the previous two cycles. By inspecting it, we notice the following. First, the transition T2 does not look exactly the same and is followed by a new transition timepoint (T4), which reoccurs before pattern P4. Second, pattern P3 is significantly shorter in length compared to the previous cycles, we can hardly see it in Figure 8. Additionally, it occurs again before pattern P4. Third, we can detect the newly introduced pattern (P5) along with the altering behavior between pattern P5 and pattern P2 before transition T3.

To interpret these insights, we looked-up the function names corresponding to the vertices in the graph. In the first two cycles, the user tried to draw an ellipse (transition T2), then he used the scribble tool (transition T3), whereas in the third cycle, the user draws a rectangle instead (transition T2), then again he used the scribble tool (transition T3). Furthermore, pattern P1 corresponds to function calls that release the used computing resources, whereas pattern P4 reflects the deactivation of the current drawing tool, which explains why pattern P1 always follows pattern P4.

5. Conclusion

In this paper, we introduced a time-scalable approach for visualizing dynamic graphs based on bipartite graph layout. To achieve time-scalability together with revealing temporal patterns, individ-

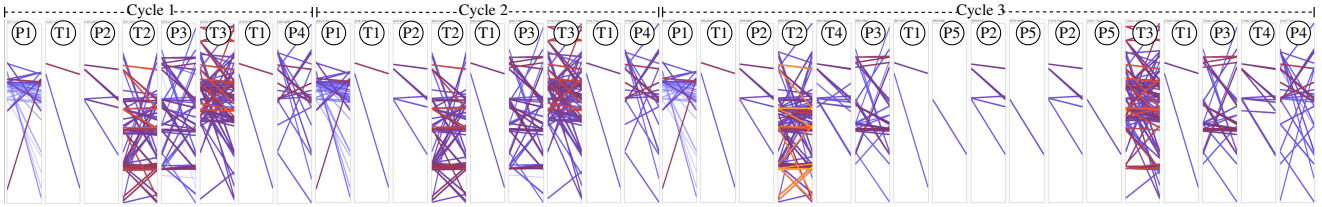


Figure 10: *The dynamic graph from the software system sequentially clustered so that timepoints that share the same temporal pattern belong to the same cluster. Each cluster is represented by the representative graph.*

ual graphs are horizontally stacked by drawing partial edges, leading to stacked edge splatting. By introducing the representative graph and placing it next to the last graph in the sequence, we preserve the information about the graph structure. Therefore, the detailed information about edge direction and target node can be obtained by tracing the partial edges back to the representative graph where edges are drawn once at full length. We apply temporal clustering to group similar graphs together in one cluster. On the one hand, it provides an overview of different temporal phases of the graph sequence together with the corresponding graph structure for each phase. On another hand, it improves the edge-tracing task. We demonstrated the effectiveness of our approach using two real-world datasets. For future work, we plan to explore other heuristics for temporal clustering (i.e., the model-based clustering methods).

References

[APP11] ARCHAMBAULT D., PURCHASE H. C., PINAUD B.: Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics 17*, 4 (2011), 539–552. 2

[BBDW17] BECK F., BURCH M., DIEHL S., WEISKOPF D.: A taxonomy and survey of dynamic graph visualization. *Computer Graphics Forum 36*, 1 (2017), 133–159. 2

[BCG*12] BRUCKDORFER T., CORNELSEN S., GUTWENGER C., KAUFMANN M., MONTECCHIANI F., NÖLLENBURG M., WOLFF A.: Progress on partial edge drawings. In *Proceedings of the Symposium on Graph Drawing* (2012), pp. 67–78. 2

[BETT99] BATTISTA G. D., EADES P., TAMASSIA R., TOLLIS I. G.: *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999. 2

[BEW95] BECKER R. A., EICK S. G., WILKS A. R.: Visualizing network data. *IEEE Transactions on Visualization and Computer Graphics 1*, 1 (1995), 16–28. 2

[BHRD*15] BACH B., HENRY-RICHE N., DWYER T., MADHYASTHA T., FEKETE J.-D., GRABOWSKI T.: Small MultiPiles: piling time to explore temporal patterns in dynamic networks. *Computer Graphics Forum 34*, 3 (2015), 31–40. 4

[BHW17] BURCH M., HLAWATSCH M., WEISKOPF D.: Visualizing a sequence of a thousand graphs (or even more). *Computer Graphics Forum 36*, 3 (2017), 261–271. 2, 3

[BPF14] BACH B., PIETRIGA E., FEKETE J.: Visualizing dynamic networks with matrix cubes. In *Proceedings of Conference on Human Factors in Computing Systems* (2014), pp. 877–886. 2

[Bur17] BURCH M.: Visual analytics of large dynamic digraphs. *Information Visualization 16*, 3 (2017), 167–178. 2

[BVB*11] BURCH M., VEHLOW C., BECK F., DIEHL S., WEISKOPF D.: Parallel edge splatting for scalable dynamic graph visualization. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2344–2353. 2, 3

[BVKW11] BURCH M., VEHLOW C., KONEVTSOVA N., WEISKOPF D.: Evaluating partially drawn links for directed graph edges. In *Proceedings of the Symposium on Graph Drawing* (2011), pp. 226–237. 2

[DG02] DIEHL S., GÖRG C.: Graphs, they are changing. In *Proceedings of the Symposium on Graph Drawing* (2002), pp. 23–30. 2

[Die07] DIEHL S.: *Software Visualization – Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, 2007. 1

[FT04] FRISHMAN Y., TAL A.: Dynamic drawing of clustered graphs. In *Proceedings of 10th IEEE Symposium on Information Visualization* (2004), pp. 191–198. 2

[GFC05] GHONIEM M., FEKETE J., CASTAGLIOLA P.: On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis. *Information Visualization 4*, 2 (2005), 114–135. 2

[HE12] HEALEY C. G., ENNS J. T.: Attention and visual memory in visualization and computer graphics. *IEEE Transactions on Visualization and Computer Graphics 18*, 7 (2012), 1170–1188. 2

[iS98] I SILVESTRE J. P.: Approximation heuristics and benchmarkings for the MinLA problem. In *Proceedings of Algorithms and Experiments* (1998), pp. 112–128. 3

[JHo18] JHotDraw Start Page. http://www.jhotdraw.org/, 2018. (Accessed on 03/22/2018). 2, 4

[KPW14] KERREN A., PURCHASE H. C., WARD M. O.: Introduction to multivariate network visualization. In *Multivariate Network Visualization*, Kerren A., Purchase H. C., Ward M. O., (Eds.). Springer, 2014, pp. 1–9. 1

[MELS95] MISUE K., EADES P., LAI W., SUGIYAMA K.: Layout adjustment and the mental map. *Journal of Visual Languages and Computing 6*, 2 (1995), 183–210. 2

[RLMJ05] ROSENHOLTZ R., LI Y., MANSFIELD J., JIN Z.: Feature congestion: a measure of display clutter. In *Proceedings of Conference on Human Factors in Computing Systems* (2005), pp. 761–770. 2

[TMB02] TVERSKY B., MORRISON J. B., BÉTRANCOURT M.: Animation: can it facilitate? *International Journal of Human-Computer Studies 57*, 4 (2002), 247–262. 2

[Uni18] UNITED STATES DEPARTMENT OF TRANSPORTATION: On-Time Performance. https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time, 2018. (Accessed on 03/22/2018). 2, 4

[vdEHBvW13] VAN DEN ELZEN S., HOLTEN D., BLAAS J., VAN WIJK J. J.: Reordering massive sequence views: Enabling temporal and structural analysis of dynamic networks. In *Proceedings of IEEE Pacific Visualization Symposium* (2013), pp. 33–40. 2

[vdEHBvW16] VAN DEN ELZEN S., HOLTEN D., BLAAS J., VAN WIJK J. J.: Reducing snapshots to points: A visual analytics approach to dynamic network exploration. *IEEE Transactions on Visualization and Computer Graphics 22*, 1 (2016), 1–10. 2

[vLdL03] VAN LIERE R., DE LEEUW W. C.: GraphSplatting: Visualizing graphs as continuous fields. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (2003), 206–212. 2