

Institute for Visualization and Interactive Systems

University of Stuttgart  
Universitätsstraße 38  
D-70569 Stuttgart

Masterarbeit

# Multi-Time Scale Dynamic Graph Visualization

Moataz Abdelaal

**Course of Study:** Computer Science

**Examiner:** Prof. Dr. Daniel Weiskopf

**Supervisor:** Dr. Michael Burch

**Commenced:** February 1st, 2017

**Completed:** August 1st, 2017

**CR-Classification:** I.7.2



## Abstract

Dynamic graphs build complex data structures composed of vertices, edges, and time steps. Visualizing these evolving structures is a challenging task when we are not only interested in the dynamics based on a fixed time granularity, but also in exploring the subsequences at multiple of those time granularities. In this thesis, we introduce a multi-timescale dynamic graph visualization. The dynamic graph is displayed with interleaved parallel edge splatting focusing on visual scalability to generate an overview of dynamic graph patterns first. Different time scales can then be displayed in a vertically stacked scale-to-space mapping showing finer time granularities in linked side-by-side views, which is in particular useful for comparison tasks. To obtain an uncluttered view of the evolving graph patterns, the data is first preprocessed by clustering and vertex ordering techniques. It is then plotted in a 1D bipartite layout, splatted, smoothed, and enhanced with contour lines for perceptual augmentation. Inner- and inter-scale comparisons are supported visually and algorithmically.



## Acknowledgments

*Micheal:* Thanks for the advising and assisting. This work would not have been possible without you. I am a lucky person to work with you. I learned a lot and still learning.

*Daniel:* I really appreciate your guidance and support. I hope I was up to the standards!

*Ayush:* Thanks for the company.

*Mom and Dad:* Thanks for making me the man who I am.

*My Beloved Aunt:* This is for you.



# Contents

|     |   |    |
|-----|---|----|
| 1   | INTRODUCTION  | 15 |
| 1.1 | Motivation . . . . .                                    | 15 |
| 1.2 | Objective . . . . .                                     | 16 |
| 1.3 | Contributions . . . . .                                 | 17 |
| 1.4 | Outline . . . . .                                       | 18 |
| 2   | RELATED WORK  | 21 |
| 2.1 | Introduction . . . . .                                  | 21 |
| 2.2 | Animation (Time-to-Time Mapping) . . . . .              | 21 |
| 2.3 | Timeline (Time-to-Space Mapping) . . . . .              | 22 |
| 2.4 | Hybrids . . . . .                                       | 28 |
| 2.5 | Our Approach . . . . .                                  | 28 |
| 3   | PROCESSING DYNAMIC GRAPH DATA                           | 31 |
| 3.1 | Introduction . . . . .                                  | 31 |
| 3.2 | Data Model . . . . .                                    | 31 |
| 3.3 | Vertex Clustering . . . . .                             | 32 |
| 3.4 | Vertex Reordering . . . . .                             | 34 |
| 3.5 | Quality Judgements for the Vertex Order . . . . .       | 35 |
| 4   | MULTI-TIMESCALE DYNAMIC GRAPH VISUALIZATION             | 39 |
| 4.1 | Introduction . . . . .                                  | 39 |
| 4.2 | Visualizing a Sequence of a Thousand Graphs . . . . .   | 40 |
| 4.3 | Visualizing on Multiple Time Scales . . . . .           | 45 |
| 4.4 | Algorithmic Dynamic Graph Sequence Comparison . . . . . | 47 |
| 4.5 | Parameter Adjustments . . . . .                         | 47 |
| 5   | INTERACTION   | 51 |
| 5.1 | Introduction . . . . .                                  | 51 |
| 5.2 | Exploratory Analytics . . . . .                         | 51 |
| 5.3 | Interactive Filtering . . . . .                         | 52 |
| 5.4 | Algorithmic Comparison of Subsequences . . . . .        | 54 |
| 5.5 | Selection . . . . .                                     | 55 |

|     |  |    |
|-----|--|----|
| 6   | APPLICATION EXAMPLE                            | 57 |
| 6.1 | Introduction . . . . .                         | 57 |
| 6.2 | Visualizing Dynamic Graph Data . . . . .       | 57 |
| 6.3 | Identifying Spatio-Temporal Patterns . . . . . | 60 |
| 6.4 | Multiscale View . . . . .                      | 62 |
| 6.5 | Algorithmic Dynamic Graph Comparison . . . . . | 64 |
| 7   | IMPLEMENTATION DETAILS                         | 67 |
| 7.1 | Introduction . . . . .                         | 67 |
| 7.2 | Data Format . . . . .                          | 67 |
| 7.3 | System Architecture . . . . .                  | 68 |
| 7.4 | Implementation . . . . .                       | 70 |
| 7.5 | Running Time . . . . .                         | 72 |
| 8   | CONCLUSIONS                                    | 75 |
| 8.1 | Summary . . . . .                              | 75 |
| 8.2 | Discussion and Limitations . . . . .           | 76 |
| 8.3 | Conclusion and Future Work . . . . .           | 77 |
|     | Bibliography                                   | 79 |



# List of Figures

|     |   |    |
|-----|---|----|
| 1.1 | Node-link representation. . . . .   | 16 |
| 2.1 | Force-directed based approach for visualizing clustered graphs [FT04]. .                          | 22 |
| 2.2 | <i>Cubix</i> an easy-to-understand approach to visualize dynamic networks [BPF14]. . . . .        | 23 |
| 2.3 | <i>MultiPiles</i> is based on the physical analogy of piling adjacency matrices [BRD+15]. . . . . | 24 |
| 2.4 | Adjacency Lists [HBW14]. . . . .  | 24 |
| 2.5 | Dynamic graph visualization based on radial layout. . . . .                                       | 26 |
| 2.6 | Visual analytics approach for the exploration and analysis of dynamic graphs [EHBW16]. . . . .    | 26 |
| 2.7 | Parallel Edge Splatting technique [BHW17]. . . . .  | 27 |
| 2.8 | Interleaving the vertical stripes together with only one pixel offset [BHW17].                    | 27 |
| 3.1 | Applying clustering and reordering techniques to dynamic graph data. .                            | 37 |
| 3.2 | Clusters hierarchies visualized as dendrograms. . . . .   | 38 |
| 4.1 | A multi-timescale view on a dynamic graph dataset. . . . .  | 39 |
| 4.2 | A visual mapping pipeline to visualize dynamic graphs. . . . .                                    | 40 |
| 4.3 | Transforming a node-link diagram in 2D into a bipartite layout. . . . .                           | 41 |
| 4.4 | Interleaving two graphs in a bipartite layout as node-link diagrams. . . .                        | 42 |
| 4.5 | Edge splatting illustrated for three crossing edges with different weights.                       | 44 |
| 4.6 | Smoothing the density field visualization. . . . .  | 45 |
| 4.7 | Augmenting contour lines in the density field of our visualization. . . . .                       | 46 |
| 4.8 | Discrete color bar with five color bins (white, gray, red, yellow, and green).                    | 46 |
| 5.1 | The vertices side menu. . . . .   | 53 |
| 5.2 | The edge weight slider. . . . .   | 54 |
| 5.3 | Modes of selection interaction technique. . . . .   | 56 |
| 5.4 | Multiple selections on the same time period. . . . .  | 56 |
| 6.1 | The dynamic graph visualization can show several views based on multiple timescales. . . . .      | 58 |
| 6.2 | Clusters visualized on the map of the United States. . . . .                                      | 59 |

|     |  |    |
|-----|--|----|
| 6.3 | Filtering the graph at different thresholds values. . . . .  | 60 |
| 6.4 | Interesting spatial and temporal patterns. . . . .   | 61 |
| 6.5 | Comparing the vertical axis of January 2000 and January 2001. . . . .  | 62 |
| 6.6 | Filtered flight data for January in 2000. . . . .  | 63 |
| 6.7 | A visual comparison of September 2000 and September 2001. . . . .  | 64 |
| 6.8 | Algorithmic comparison between September in both 2000 and 2001<br>respectively. . . . .  | 65 |
| 6.9 | Algorithmic comparison of three selected Tuesdays (11 <sup>th</sup> , 18 <sup>th</sup> and 25 <sup>th</sup> )<br>of January 2000 respectively. . . . . | 66 |
| 7.1 | High level architecture of our visualization system. . . . .   | 68 |
| 7.2 | The database design of our visualization system. . . . .   | 69 |
| 7.3 | tbl_img_t0_vertices maps between the vertices and outgoing edges. . .  | 70 |
| 7.4 | The main screen of our system. . . . .   | 71 |
| 7.5 | The running times of initialization tasks. . . . .   | 72 |
| 7.6 | Code snippet of getDynamicGraph controller. . . . .  | 73 |

# List of Tables

|     |  |    |
|-----|--|----|
| 7.1 | tbl_airport database table description. . . . .            | 69 |
| 7.2 | tbl_img_t0_vertices database table description. . . . .    | 69 |
| 7.3 | Technologies used during the implementation phase. . . . . | 71 |



# List of Algorithms

- 3.1 Hierarchical Clustering . . . . . 32
- 3.2 Vertex Reordering . . . . . 35



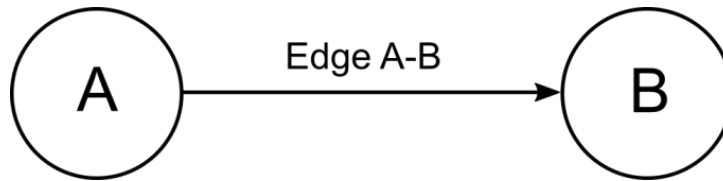
# 1 INTRODUCTION

## 1.1 Motivation

In our world, we can hardly find anything being static. From the day we born we are connected to something else. i.e., to the family. Being part of the family network allows us to share information and resources with other family members who are themselves connected to other networks like the neighbors, the work associates, or the school friends, who are connected to other networks which are connected to other networks and so on.

This theme of networking is ubiquitous to our world, where *objects* are connected to each other by one *relation* or another. The network could model physical phenomena, like in the family network. But it could also describe non-physical one; like the online social networks with friendships between people or the e-mail networks where e-mails are sent between email accounts. Some other examples are the networks of transportation where people and goods are transported between cities; the call graph of a computer program where different subroutines call each other; the neural network inside the human brain where electrical signals are transmitted between neurons; the protein interaction networks in a cell where interactions occur between protein molecules, *et cetera*.

Node-link diagrams are the most familiar representation to visualize networks (see Figure 1.1). Each object is depicted as a node and each relation is depicted as an edge connecting two nodes. This representation is good at showing the overall graph structure and works better than other representations (i.e., adjacency matrices) for graph exploration tasks. However, when the underlying network consists of several hundred vertices and several thousands of edges, which is often the case, the resulting graph becomes very dense, suffering from visual clutter. Hence, this makes the graph exploration a real challenge and non-trivial task.



**Figure 1.1:** Node-link representation, each object is depicted as a node and each relation is depicted as an edge connecting two nodes

### 1.1.1 Dynamic Graphs

In many scenarios, networks are not static but exhibit a dynamic behavior where the structure of the network changes over the time. Dynamic graphs are frequently used to model such temporal behavior. Getting an overview of long graph sequences is one of the biggest challenges in dynamic graph visualization.

To obtain insights into the evolution of the graph, the temporal patterns such as *trends*, *countertrends*, *temporal shifts*, *oscillations*, *outliers*, and *time-dependent anomalies* need to be identified. To reach this goal, a visual scalable approach that uncovers patterns in the dynamic graph data is required. Such an approach would provide a starting point for further data exploration tasks.

### 1.1.2 Multiple Time Scales

Visualizing dynamic graphs is a challenging task when we are not only interested in the dynamics based on a fixed time granularity, but also in exploring the graph subsequences at multiple of those granularities and comparing them without losing the context.

Showing the graph dynamics at multiple time granularities allows us to uncover the temporal patterns at the finer granularities. Such patterns are harder to detect at coarser levels as they are aggregated and compressed. Therefore, a multi-timescale visualization approach that shows the dynamic graph structures on multiple time scales directly in a combined fashion is needed.

## 1.2 Objective

In this work, we are trying to answer the following question:



“ *How to provide an overview of multiple time scales of the time-varying relational data, allowing an analyst to inspect the dynamics on these scales separately, but also in combination?* ”

To address this question we follow an experimental approach in which a prototype is implemented and continuously enhanced and improved to reach our proposed design goals. Also, we integrate extensive interaction techniques into our solution. Such techniques help users to filter data, maintain context information and to compare, either visually or algorithmically, between individually selected time periods.

Additionally, we illustrate the visual scalability and the usefulness of our solution by means of an application example based on a real world dataset which demands a scalable visualization.

## 1.3 Contributions

In this thesis, we introduce a multi-timescale dynamic graph visualization approach that is visually scalable in any of the three data dimensions (vertices, edges, and time). To reach this goal, we follow a concept recently developed by Burch et al. [BHW17] based on the interleaved parallel edge splatting technique to generate an overview of dynamic graph patterns first. We then extend our visualization by displaying different time scales in a vertically stacked scale-to-space mapping showing finer time granularities in linked side-by-side views, which is in particular useful for comparison tasks.

The key contributions of this thesis are:

1. Introducing a visually scalable dynamic graph visualization approach based on the interleaved parallel edge splatting technique. It maps time points to a time axis in which each time point has a certain extent allowing a meaningful application of the interleaving concept. This is particularly useful when the graph structure has to be identified while being visually scalable in the time dimension.
2. Using the compact layout of our visualization technique, we show the dynamic graph structures on multiple time scales directly in a combined fashion. The multiple timescales are stacked on top of each other, while several time periods can be selected on the same temporal granularity placed next to each other. By following this design, we are able to compare the dynamic graphs side-by-side and to set them in context to the coarser granular visualizations placed on top of them while guiding lines support the comparisons tasks.

3. To obtain an uncluttered view of the evolving graph structures and temporal patterns, clustering and ordering techniques are implemented to improve the aesthetics of the diagrams and to reveal certain graph structures over time.
4. To uncover important graph structures, we apply several interactive filtering techniques in any of the three data dimensions (vertices, edges, and time). It helps reducing the amount of visual clutter, overdrawing, and occlusions.
5. To illustrate the usefulness and applicability of our technique. We introduce an application example based on a real world dataset, which demands a scalable visualization in all three data dimensions.

Our approach guarantees that several dynamic graph visualizations are visualized in a single static view. Such a horizontal time-to-space mapping for each time scale and vertical scale-to-space mapping is in favor to a time-to-time mapping (graph animation) since it supports comparison tasks over longer time periods [TMB02] while it visually scales well in the time dimension reflecting dynamic patterns.

### 1.4 Outline

The remaining of this thesis is organized as follows:

Chapter 2 discusses the related work of dynamic graph visualization. We review different visualization techniques based on the three major concepts for representing the time aspect in graphs, time-to-time mappings, time-to-space mappings and hybrids techniques. For each method, we identify the limitations and the open problems.

Chapters 3 to 6 contain the main contributions of this thesis. In Chapter 3, we explain the preprocessing phase of dynamic graph data. First, we introduce the data model of vertices and edges in the graph layout. Then, we discuss the clustering and reordering techniques adopted by our solution.

In Chapter 4 we present the novel multi-timescale dynamic graph visualization approach. First, we review the interleaved parallel edge splatting technique developed by Burch et al. [BHW17]. Next, we introduce the visualization on multiple time scales. Finally, we discuss the effect of the user-defined parameters on the visual appearance of the final visualization.

Chapter 5 focuses on the interaction techniques implemented in our solution. We present the clustering and ordering techniques followed by different filtering options. Finally, we investigate the usability of our technique in conducting comparison tasks between dynamic graph sequences, both visually and algorithmically.

In Chapter 6, we present an application example to illustrate the usefulness and applicability of our approach based on a real dynamic graph dataset, while in Chapter 7 we provide the necessary implementation details to reproduce the claimed results of this thesis.

Finally, Chapter 8 concludes the thesis by discussing the scalability issues of our proposed solution followed by the conclusions and directions for future work.



## 2 RELATED WORK

### 2.1 Introduction

In the previous chapter, we have introduced the motivation and the main contributions of this thesis. In this chapter, we explore the state-of-the-art of dynamic graph visualization. Visualizing the dynamics of a graph has become an important research topic in visualization and visual analytics [BBDW17; LKS+11]. There are three major concepts for the representation of the time aspect in graphs that can be classified into time-to-time mappings, i.e., animated diagrams (Section 2.2), time-to-space mappings, i.e., static diagrams (Section 2.3), or hybrid approaches that combine both (Section 2.4). In the following sections, we briefly present selected works for each mapping concept with more focus towards time-to-space mapping.

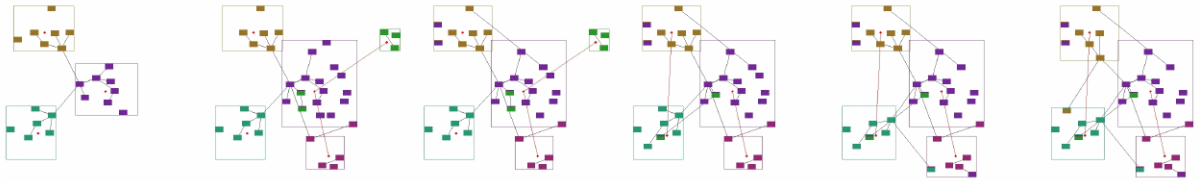
### 2.2 Animation (Time-to-Time Mapping)

In this section, we shortly review two methods where the time dimension of the graph is encoded as an animation. i.e., animated node-link diagrams.

Based on Foresighted Layout [DGK01], Diehl et al. [DG02] introduced a generic algorithm to compute the graph animations. The algorithm consider all graphs in the sequence (offline) instead of just the previous ones (online) when computing the layout of the current graph. The Foresighted Layout is extended so that it trades aesthetic quality of the graph for dynamic stability and vice versa. The smooth transition between two graphs are animated in the following phases. First all deleted edges shrink and disappear. Then all deleted nodes disappear. Next all remaining nodes and edges are moved to their new positions using linear interpolation. Finally all new nodes appear and all new edges expand. However, the generic nature of the approach comes at the cost of performance.

Frishman et al. [FT04] presented a force-directed based approach for visualizing clustered graphs. To maintain the clustered structure, dummy vertices and edges are

introduced. Additionally, invisible place-holders, called spacer vertices, are used to reduce the change in clusters' boundaries and minimize the movement of clusters between successive layouts, whereas the weight and length of the edges are used to control the changes made to the layout. When calculating the outline of each cluster (the bounding box) the spacer vertices are treated the same as the visible vertices (see Figure 2.1). Due to the complexity of the algorithm, the running time is higher than other methods.



**Figure 2.1:** Force-directed based approach for visualizing clustered graphs [FT04].

Taking into account the research question we are trying to address in this thesis, animated diagrams are considered worse for exploration tasks [APP11; TMB02], in particular, for comparisons between different time steps and time periods, also on different time scales as proposed in this work. The reason behind this is the preservation of the mental map that is responsible for reducing cognitive efforts caused by the weaknesses of our short term memory [War04]. Moreover, sophisticated algorithms are needed to generate stable dynamic layouts [DG02] that still show graph structures like clusters.

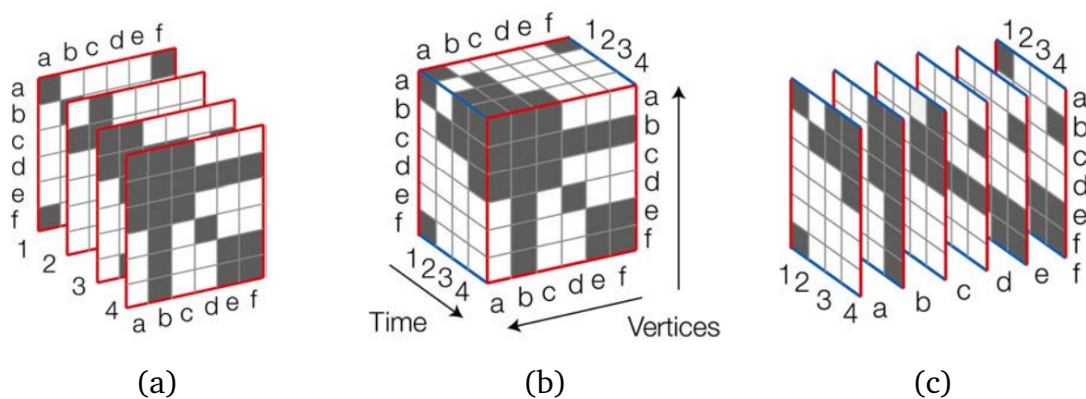
## 2.3 Timeline (Time-to-Space Mapping)

In this section, we explore a big family of dynamic graph visualization methods where the graph can be drawn onto timeline instead of using the animation. Timeline based approaches are considered better at providing an overview of the graph evolution over the time since the entire graph sequence get visualized in a single static image.

Based on node-link diagrams, the prominent visual metaphor, Collberg et al. [CKN+03] introduced *Gevol*, a system for visualizing the evolution of software. The system extracts the inheritance graphs, the call graphs, and the control-flow graphs of Java programs and displays the changes the graphs have gone through since the inception of the program. The node color is used to encode the change. They start out being red and grow paler and paler for every time-slice they have remained unchanged until finally, they become blue. The system utilizes the force-directed method of Kamada and Kawai to lay out

the nodes in the graph. To construct the "time-slice-graph" the daily graphs are merged together by identifying nodes correspondence in subsequent graphs. This method, however, does not scale visually with large graphs. Additionally, it is space-inefficient considering graphs with thousands or more time steps.

By using adjacency matrices as a visual metaphor, Bach et al. [BPF14] presented *Cubix* an easy-to-understand approach to visualize dynamic networks based on the space-time cube metaphor. It is created by stacking adjacency matrices in chronological order, one for each time step (see Figure 2.2). Different interaction techniques are supported. i.e., slicing, projection, rotation, small multiples, filtering, brushing and linking.

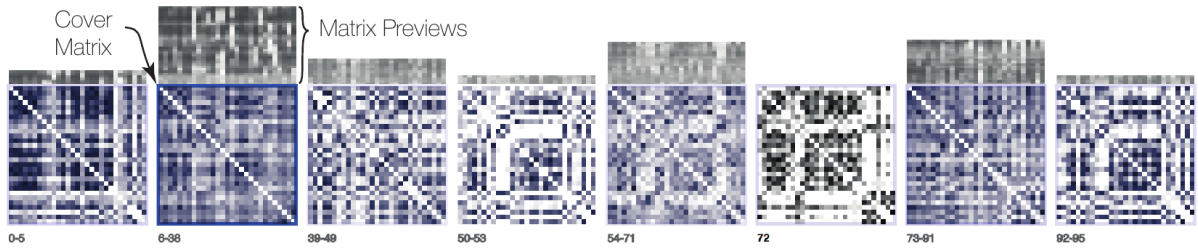


**Figure 2.2:** *Cubix* an easy-to-understand approach to visualize dynamic networks based on the space-time cube metaphor. (a) Adjacency Matrices (b) Matrix Cube (c) Vertex Slices [BPF14].

*Small MultiPiles* [BRD+15] is another matrix-based approach designed to explore time-series of dense, weighted networks. The technique is based on the physical analogy of piling adjacency matrices, each one representing a single temporal snapshot. The similar matrices are grouped together forming a pile. The user can adapt the distance threshold used to create piles by steering a piling slider. High threshold results in fewer piles, while lower threshold results in more and smaller piles but with a higher similarity within each pile. A matrix preview is stacked on the top each pile to give quick pile size comparison (see Figure 2.3). Additional operations on piles including formation, exploration, aggregation, filtering, propagation.

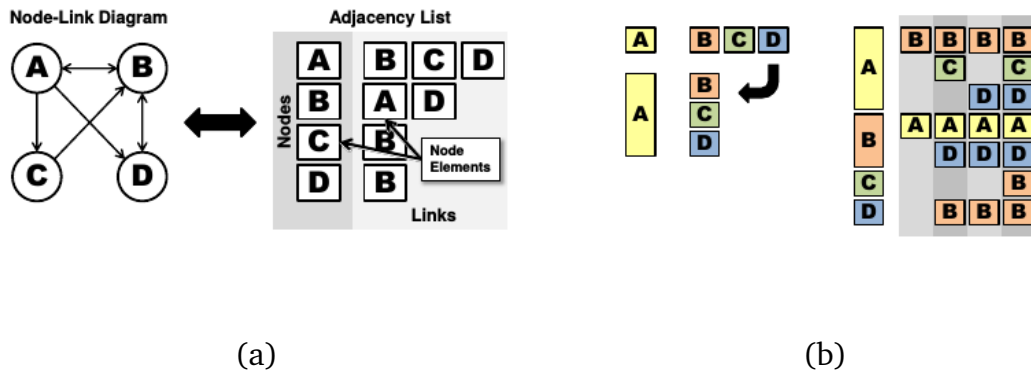
Although both *Cubix* and *Small MultiPiles* provide a compact visual representation and a more readable visualization due to their matrix representation, they don't scale to thousands of graphs nor do they reflect the evolution over time on different time scales. Also, visual and algorithmic comparison tasks among time scales are not supported.

Hlawatsch et al. [HBW14] introduced a visual representation for dynamic weighted graph based on the concept of adjacency lists. In the basic layout, two orthogonal axes



**Figure 2.3:** *MultiPiles* is based on the physical analogy of piling adjacency matrices, each one representing a single temporal snapshot [BRD+15].

are used: one for all nodes in the graph, the other for the corresponding links (see Figure 2.4(a)). Labeling can be used to identify the nodes. However, using color improve the visual scalability. To encode the time dimension, nodes are positioned on the node axis in the center and groups the time steps of incoming (left) and outgoing (right) links. Time evolves from the center to the outside. Due to the difficulty of comparing the incoming and outgoing links of a specific time step using this layout. a Gantt Layout is introduced where it is possible to subdivide the row for a single node. In this case, if a single link exists for several time steps, it is visually connected through these time steps (see Figure 2.4(b)).



**Figure 2.4:** Adjacency Lists. (a) The concept of visual adjacency lists, the classic node-link diagram (left) and the corresponding adjacency list (right) (b) Gantt layout. it is possible to subdivide the row for a single node. [HBW14].

Adjacency lists provide clutter-free visual representation just like the adjacency matrices. However, in the case of sparse graphs, adjacency list provides a more compact design compared to adjacency matrices. Furthermore, it obtains better results with respect to the weight-related tasks. On the other hand, it is very hard to detect clusters and other



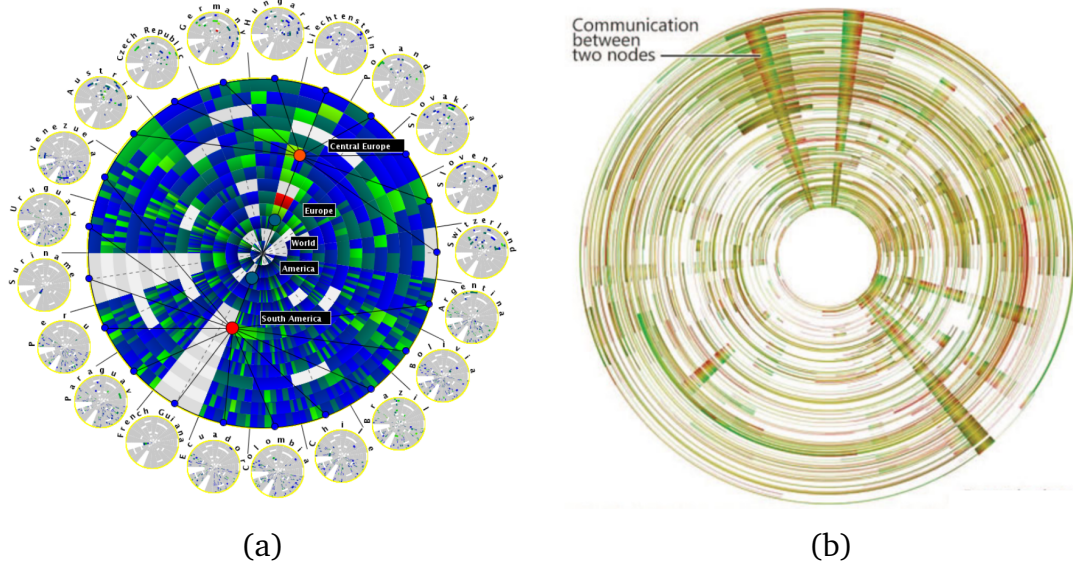
graph structures since the color coding of visual adjacency lists usually provides a lower resolution on typical output devices. Additionally, the current method doesn't visually scale with respect to the network size and the number of timesteps.

Burch et al. [BD08] introduced *TimeRadarTrees*, a radial-layout based approach to visualize sequences of compound graphs (see Figure 2.5(a)). Each node in the graph is represented by a sector of the inner circle (Time Radar). The sectors are subdivided into a number of smaller sectors depending on the number of incoming edges of the associated node. The outer circles (Thumbnails) attached to each sector show the outgoing edges of the related node. To encode time, both inner and outer circles are subdivided into tracks. Each track resembles a graph sequence. The first graph of the sequence is represented by the innermost track, the next graph sequence is represented by the next outer track and so on. In such way, time evolves from the center to the outside.

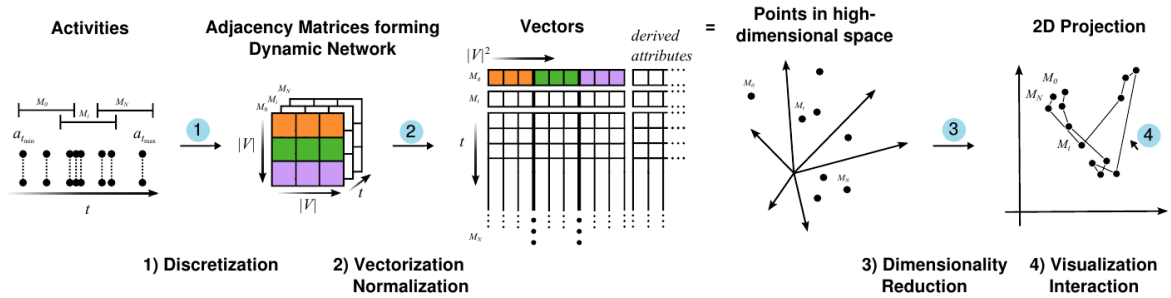
Van den Elzen et al. [EHBW14] proposed a similar approach based on Extended Massive Sequence View (short: MSV) [CHZ+07]. The idea is to extend the MSV by allowing edges to wrap around, which results in a (circular) MSV with much reduced visual clutter (see Figure 2.5(b)). Using the circular representation an edge can always be drawn in two ways, using either the longest or shortest path between two nodes. Similar to *TimeRadarTrees*, time is encoded by the radius of the circle. As time progresses edges are drawn with an increasing amount of ink. To avoid that bias, an interaction technique is added to reverse the time direction.

Circular visualizations need much screen space to visualize large graphs. Additionally, comparisons of time moments are harder to conduct using radial-layout based visualizations.

A clever way to tackle the problem of visualizing large dynamic graphs is to reduce the dimensions of the problem. Inspired by that, Van den Elzen et al. [EHBW16] proposed a novel approach where graph sequences, called snapshots, are vectorized and considered as points in high-dimensional space (see Figure 2.6). Each point in this high-dimensional space represents the graph at a different time-interval. Points corresponding to similar graph sequences are positioned closer to each other and form a cluster. While for timesteps where the graph is different from the common sequences, the points will be outliers. To project the points in two dimensions, dimensionality reduction techniques are applied. Finally, two juxtaposed linked views are introduced: one shows the projection of the snapshots and the second provides a network visualization for a selected point. Due to the memory requirement, this approach has visual scalability concerns, with respect to graph size and number of timesteps involved. Also, non-linear dimensionality has the downside that it is harder to interpret the resulting dimensions.



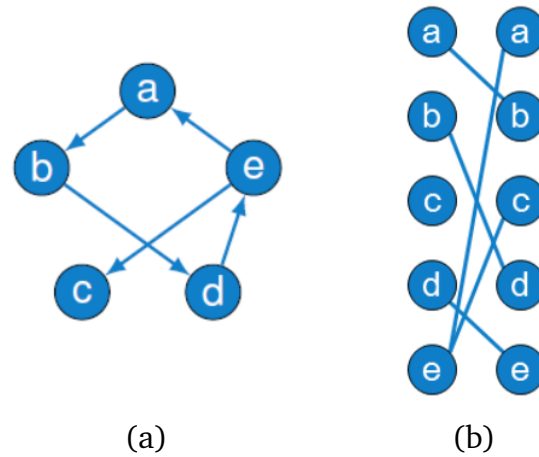
**Figure 2.5:** Dynamic graph visualization based on radial layout. (a) *TimeRadarTrees* [BD08] (b) *Circular MSV* [EHBW14].



**Figure 2.6:** Visual analytics approach for the exploration and analysis of dynamic graphs [EHBW16].

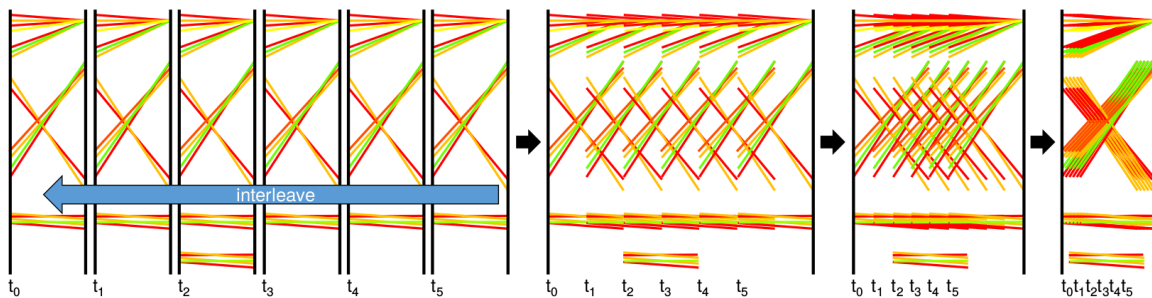
Similarly, Burch et al. [BVB+11a] reduced the dimensionality of the problem by introducing the Parallel Edge Splatting technique where graph vertices are placed on a 1D vertical line (bipartite layout) perpendicular to the horizontal time line. The graph sequences are drawn side-by-side from left to right as a sequence of narrow stripes. Each stripe is composed of two vertical, parallel lines, where the hierarchically organized vertices are placed on both lines with the same order. The edge direction information is encoded in the left-to-right reading direction (see Figure 2.7). To address massive overplotting of edges in huge graphs, a splatting approach is employed that transforms the edges to a pixel-based scalar field. The technique is more scalable because of the special 1D vertex layout and the edge splatting technique. a graph becomes space-efficient,

ambiguities caused by edges crossing vertices are avoided, and the alignment leads to a better comparability of graphs throughout the sequence.



**Figure 2.7:** Parallel Edge Splatting technique. (a) Classic node-link diagram (b) Bipartite layout [BHW17].

In his recent work [BHW17]. Burch et al. extended the algorithm one step further by interleaving the vertical stripes together with only one pixel offset. As a result, the horizontal distance occupied by the graph is reduced even more freeing more space for visualizing thousands or more time steps. Also, the visual fusion resulted from the interleaving emphasizes the graph structures and allow easy detection of the temporal patterns (see Figure 2.8).



**Figure 2.8:** Interleaving the vertical stripes together with only one pixel offset [BHW17].

In this thesis, we are trying to extend this method [BHW17] by displaying different time scales in a vertically stacked scale-to-space mapping showing finer time granularities in linked side-by-side views, which is in particular useful for comparison tasks.

## 2.4 Hybrids

Other visualization techniques try to combine both animation and timeline in a hybrid approach. Beck et al. [BBV+12] proposed such approach that combines the concepts of Parallel Edge Splatting [BVB+11a] and Rapid Serial Visual Presentation (RSVP) [Spe02]. In this method, the concept of a sliding time window is used to represent a subsequence of a longer dynamic graph dataset. Instead of showing the single graphs of a sequence at the same location on the display one after each other, many graphs are shown side-by-side, all contained in the same time window, which can be moved by a given delay. When two consecutive graphs differ strongly from each other, the animation automatically slows down.

While this approach provides a good alternative by combining both paradigms; animation and timeline, it still requires some time to analyze the dynamics of a large dynamic graph. Additionally, it is difficult to detect patterns over longer time. This is due to our short term memory [War04] and challenging perceptual problems like change blindness [HE12].

## 2.5 Our Approach

The recent work of Burch et al. 2017 tries to integrate both the scalability aspect as well as the visibility of the graph structure over time into an individual static diagram with the goal to provide an overview of the dynamics. We follow the design of Burch et al. and also support intermediate representations for multi-timescale views [AMST11] on dynamic graph data. On the one hand, we focus on visually preserving the graph structure to some degree and on producing a visualization technique that is visually scalable in any of the three data dimensions (vertices, edges, and time). On the other hand, our visualization is based on a visual concept that supports the design of multi-timescale representations providing a technique for comparison tasks of several time periods selected from a longer graph sequence. To reach this goal, we first make the individual graphs artificially bipartite [BVB+11a] since this layout is compact, then we interleave the resulting node-link diagrams to achieve an even more space-efficient and compact diagram, and finally, apply a splatting technique to generate density fields that are colored and perceptually enhanced by contour lines [Bur16]. As a novel contribution, we use this compact layout of a dynamic graph for stacking it on several layers consisting of several selected time periods in any of the layers while still providing an overview of the dynamic relational data on multiple time scales.

Some approaches from the field of dynamic graph visualization try to solve this multi-timescale challenge interactively, e.g., the egoSlider technique [WPZ+16] describes a microscopic, mesoscopic, and macroscopic view on the dynamic graph data. But these multiple interactively coordinated views do not show the dynamic graph structures on multiple time scales directly in a combined fashion, but rather show different partially summarized network data. Moreover, the graph visualization component does not scale to hundreds or thousands of vertices, edges, and time steps with the goal to provide overviews about the graph dynamics.



# 3 PROCESSING DYNAMIC GRAPH DATA

## 3.1 Introduction

In the previous chapter, we discussed the state-of-art in the dynamic graph visualization area. In this chapter, we present the preprocessing phase of dynamic graph data which equivalent to the data transformations step in the visualization reference model as described by Card et al. [CMS99]. First, we introduce the data model to get a clear picture of vertices and edges in the graph layout (Section 3.2). Then, we show how, by applying the clustering (Section 3.3) and reordering techniques (Section 3.4), we could obtain a good visual representation. Finally, we see how the quality of the reordered vertices can be measured compared to the original ordering or to other orderings and subsequence selections (Section 3.5).

## 3.2 Data Model

We describe dynamic graph data

$$\Gamma = (G_1, \dots, G_n)$$

consisting of  $n \in \mathbb{N}$  individual graphs

$$G_i = (V_i, E_i)$$

Each graph  $G_i = (V_i, E_i)$  exists at a certain time point  $t_i \in T_\Gamma \subset \mathbb{N}$  where  $T_\Gamma$  is the time period the entire dynamic graph lives in.

The preprocessing algorithms can be applied to individual time points  $t \in T_\Gamma$ , to a certain time period  $T \subset T_\Gamma$ , or to the entire graph sequence  $T_\Gamma$ . We denote the involved vertices in the clustering by  $\mathbb{V}_T$  and the edges by  $\mathbb{E}_T$ , while  $T$  expresses the time interval the graph subsequence is based on, i.e.,  $T = [t_s, t_e]$  with  $t_s \in T_\Gamma$  being the start point and  $t_e \in T_\Gamma$  being the end point in the corresponding time interval. If the context is clear, we omit the subscript  $T$  in both the vertex set  $\mathbb{V}$  and the edge set  $\mathbb{E}$ . A time scale in this context is the visual granularity of the temporal graph data, i.e., it depends on the selected time period.

### 3.3 Vertex Clustering

Visualizing dynamic graphs with hundreds of vertices and thousands of edges in an efficient and effective way remains a challenging task. A good visualization strategy has to reduce the amount of visual clutter, overdrawing, and occlusions. One way to reach this goal is through applying graph clustering algorithms where vertices that are topologically similar are placed next to each other in the same cluster or group. Hence, reducing the amount of visual clutter. In addition, those clusters provide an aggregated visual representation which is in particular useful for understanding the evolution of graph patterns over the time.

Hierarchical clustering meets our needs since it is generally applicable to most types of graph data with an acceptable runtime complexity of  $O(m^2 \log m)$  if  $m \in \mathbb{N}$  is the number of vertices in the clustered graph. It does not require any predefined parameter and, hence, is suitable for handling large real-world data where finding a suitable set of parameters can be tricky, hence, it can be applied as an interactive clustering approach in our dynamic graph visualization. Our approach investigates the usefulness of agglomerative (bottom-up) hierarchical clustering with an average linkage. In this algorithm, we start with  $\mathbb{N}$  clusters and iteratively merge or *agglomerate* the two most similar clusters until all clusters are hierarchically merged into one single cluster that contains all graph vertices.

---

**Algorithmus 3.1** Hierarchical Clustering

---

```
1: procedure CLUSTERGRAPHVERTICES( $v_1, \dots, v_N$ )
2:   for  $i := 1$  to  $N$  do
3:     for  $j := 1$  to  $N$  do
4:        $C[i][j] := \text{DISTANCE}(v_i, v_j)$            // calculate the distance matrix
5:     end for
6:   end for
7:    $A := []$ 
8:   for  $k := 1$  to  $N - 1$  do
9:      $\langle i, m \rangle := \arg \min_{i \neq m} C[i][m]$            // find two most similar clusters
10:     $A[k] := \langle i, m \rangle$ 
11:    for  $j := 1$  to  $N$  do           // update rows and columns of merged clusters
12:       $C[i][j] := \text{DISTANCE}(\langle i, m \rangle, j)$ 
13:       $C[j][i] := \text{DISTANCE}(\langle i, m \rangle, j)$ 
14:    end for
15:  end for
16:  return  $A$ 
17: end procedure
```

---



The clustering algorithm is listed in Algorithm 3.1. We first compute distances for all vertex pairs  $v_i \in \mathbb{V}$  and  $v_j \in \mathbb{V}$  and take into account the weights of the edges between those vertices leading to two different similarity matrices. We have used the Jaccard coefficient to exploit the probability that two vertex sets  $V_i$  and  $V_j$  have a common feature  $f_{V_i, V_j}$  that expresses how similar two vertices  $v_i, v_j \in \mathbb{V}$  are.

- The first metric describes a measure for a common neighborhood of vertices, i.e., if those are strongly related they should belong to the same hierarchical cluster which should also be indicated by the visualization. For this reason, we build two sets of direct neighbors (reachable by a path of length 1) of each vertex pair  $v_i$  and  $v_j$  denoted by  $\bar{V}_i \subseteq \mathbb{V}$  and  $\bar{V}_j \subseteq \mathbb{V}$ . Then, the Jaccard coefficient generates those pairwise distances expressed by

$$J(\bar{V}_i, \bar{V}_j) = \frac{|\bar{V}_i \cap \bar{V}_j|}{|\bar{V}_i \cup \bar{V}_j|} \in [0, 1] \quad (3.1)$$

- The second metric solves the problem where we consider two vertices along with their edge weight  $w$  attached to them, a scenario that typically holds for general network data. Then, the weighted Jaccard coefficient can be computed by the following expression if  $\bar{V}_i \subseteq \mathbb{V}$  and  $\bar{V}_j \subseteq \mathbb{V}$  are the two sets of direct neighbors of  $v_i$  and  $v_j$ :

$$J_w(\bar{V}_i, \bar{V}_j) = \frac{\sum_{u \in \bar{V}_i \cap \bar{V}_j} W(u)}{\sum_{u \in \bar{V}_i \cup \bar{V}_j} W(u)} \in [0, 1] \quad (3.2)$$

while

$$W(u) := w(u, v_i) + w(v_i, u) + w(u, v_j) + w(v_j, u).$$

To inspect the usefulness of the clustering algorithm, we applied it to dynamic graph data consisting of several hundred vertices, several thousand edges, and thousands of time steps (see Figure 3.1). In this visualization, The y-axis represents the vertices, while the x-axis represents time. The vertical axis is depicted as two bar charts placed back-to-back at the right of visualized layout. Green bar-charts encode the frequency of the in-edges, whereas blue ones encode the frequency of the out-edges.

As shown in Figure 3.1(a), visualizing the graph raw data without applying any clustering or ordering techniques resulted in a very noisy graph with much visual clutter. We can hardly detect or identify the temporal patterns of the graph. On the other hand, Figure 3.1(b) shows the same data after applying the hierarchical clustering algorithm. As one may notice, some of the graph patterns started to emerge. Also, a less oscillating

behavior can be noticed on the vertical axis since similar vertices get placed next to each other, as a result of the clustering algorithm. To gain some insights about the main clusters in data, we depicted the clusters hierarchies as horizontal dendrograms where each merge is represented by a vertical line and the x-coordinate of the vertical line is the similarity distance at which the two clusters have been merged. In Figure 3.2(a), we annotated the six main clusters in visualized dynamic graph data.

### 3.4 Vertex Reordering

All available vertices after clustering can further be ordered in multiple ways. In graph theory, the problem of linearly ordering the vertices to optimize a cost function is known as minimum linear arrangement [GJ79], or MinLA problem, if the goal is to reduce the sum of link lengths between all pairs of vertices if those are visually mapped to a linear axis. This NP-hard problem is of particular interest for our dynamic graph visualization since a bipartite layout with vertices placed on vertical axes builds the basis for obtaining a visually scalable and graph structure-preserving visualization. We investigate several heuristical approaches to achieve good visual results that are still interactively applicable and that generate cluster-preserving structures over time.

Inspired by the successive augmentation approach [Sil98], we apply a vertex reordering technique to the hierarchy tree produced by the hierarchical clustering algorithm. The reordering algorithm is listed in Algorithm 3.2. In this technique, we are trying to find a better arrangement of vertices while preserving the hierarchical relationships between clusters. The cluster tree is traversed level by level (Breadth-first Approach). For each traversed cluster, we swap the left and right children and calculate the edge-crossing cost for the whole hierarchy before and after the swapping, so that we keep the arrangement that achieves the minimum cost. The edge-crossing cost of the whole hierarchy is computed by accumulating the weighted costs of all graph edges. The weighted edge cost is expressed by the distance between the edge end-points weighted by the frequency of that edge.

Figure 3.1(c) shows the resulted graph after applying the vertex reordering algorithm. As one may notice, the reordering of vertices has produced a less cluttered representation. However, due to the preserved hierarchy relationships, the overall graph structures generated by the hierarchical clustering algorithm (see Figure 3.1(b)) remains the same. In Figure 3.2(b) we can see how the swapping of the left and right children changed the order of six main clusters. If we look closer, we can detect changes at the bottom levels of the hierarchy as well.

**Algorithmus 3.2** Vertex Reordering

---

```

1: procedure EDGECROSSINGCOST(tree)
2:   c := 0
3:   vertices := GETTREELEAFS(tree)
4:   for all v ∈ vertices do
5:     neighbors := GETNEIGHBORS(v)
6:     for all n ∈ neighbors do
7:       c := c + (| vpos − npos | × weight(v, n))
8:     end for
9:   end for
10: end procedure
11: procedure REORDERCLUSTERHIERARCHY(root, tree)
12:   Q := []
13:   ENQUEUE(Q, root)
14:   while Q is not empty do
15:     cluster := dequeue(Q)
16:     if cluster is not leaf then
17:       costLR := EDGECROSSINGCOST(tree)           // cost before swapping
18:       SWITCHCHILDREN(cluster, tree)           // swap children
19:       costRL := EDGECROSSINGCOST(tree)           // cost after swapping
20:       if costLR < costRL then
21:         SWITCHCHILDREN(cluster, tree)           // swap children back
22:       end if
23:       left := GETLEFTCHILD(cluster)
24:       right := GETRIGHTCHILD(cluster)
25:       ENQUEUE(Q, left)
26:       ENQUEUE(Q, right)
27:     end if
28:   end while
29: end procedure

```

---

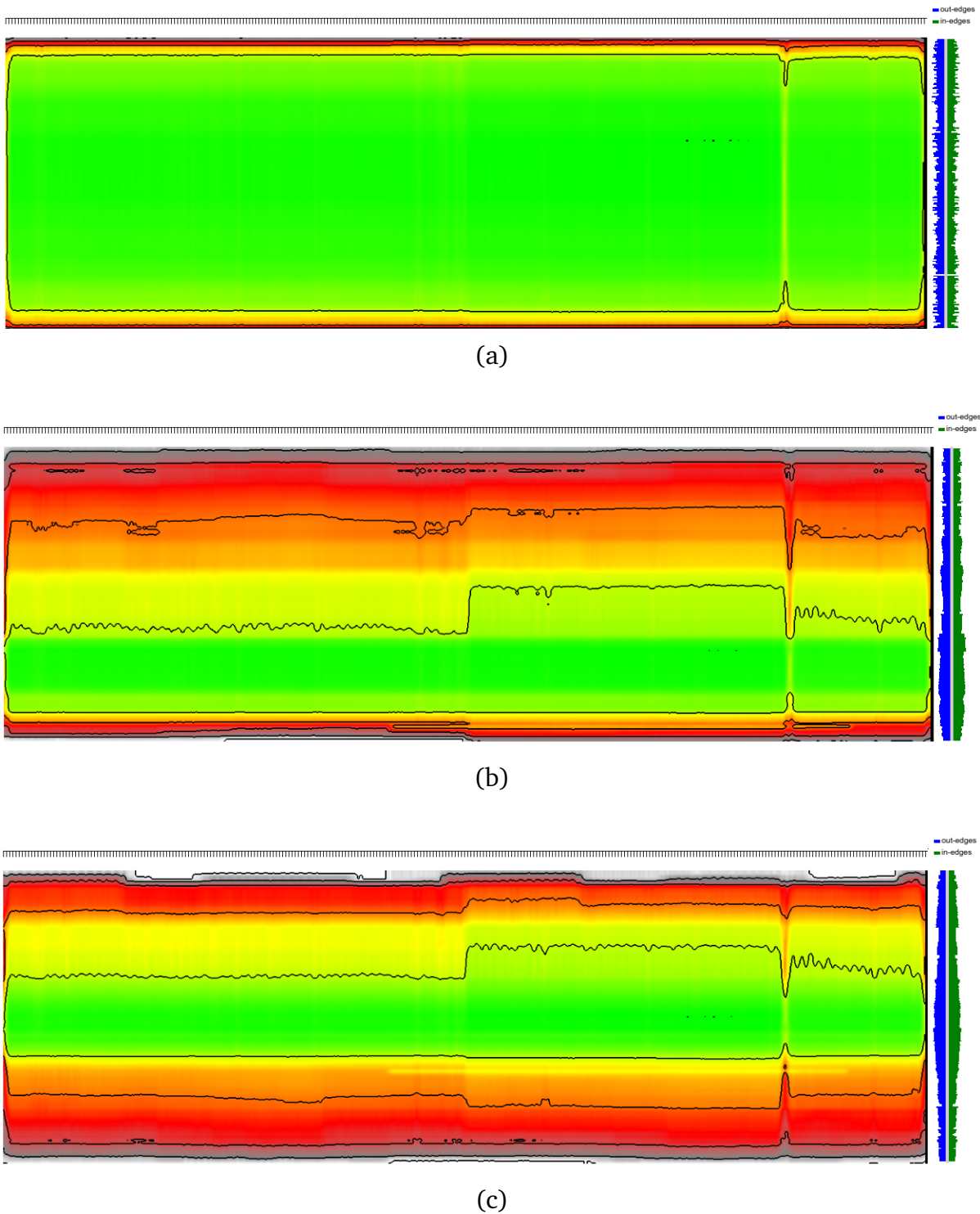
### 3.5 Quality Judgements for the Vertex Order

In the case a dynamic graph dataset does not contain a clear evolving cluster structure, the hierarchical clustering and ordering algorithms based on the entire time period  $T_\Gamma$  cannot produce a visually suitable dynamic graph visualization that reflects these clear dynamic patterns. For example, if a group of strongly connected vertices disappears over time and new clusters are build containing a subset of the vertices from the original cluster, the algorithms based on the entire time period are not able to take into account

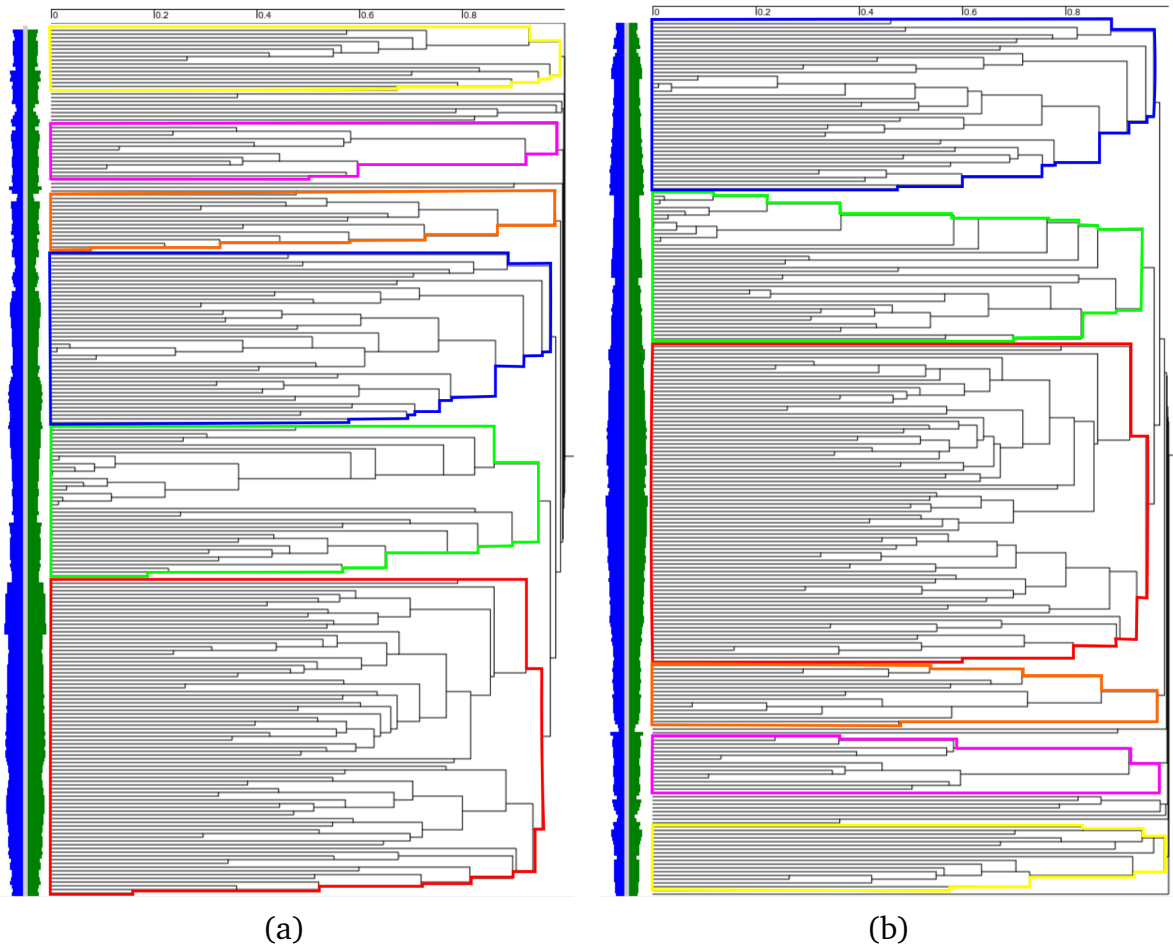
these data changes to produce a visually effective diagram. The reason is the time-aggregated clustering and ordering strategy that does not care for local dynamic patterns and cannot treat the clusters as intertwined ones separately. This is the drawback of the mental map preserving dynamic graph visualization focussing on a high degree of dynamic stability, i.e., a certain well-defined vertex order has to be kept stable over time never mind how bad a vertex order may become for later time steps.

However, to provide some solution to this problem, we can compute a quality value based on a certain clustering and ordering which can be applied to individually selected time periods. This means if a generated vertex order does not meet the requirements in a different time period, this can be algorithmically reflected by a quality measure based on a user-defined suitable vertex order that outputs a value of, e.g., the sum of link lengths which is one criterion we base the vertex ordering on. If this sum exceeds a certain number at a certain time point, we might consider reordering the corresponding vertices of the graphs.

### 3.5 Quality Judgements for the Vertex Order

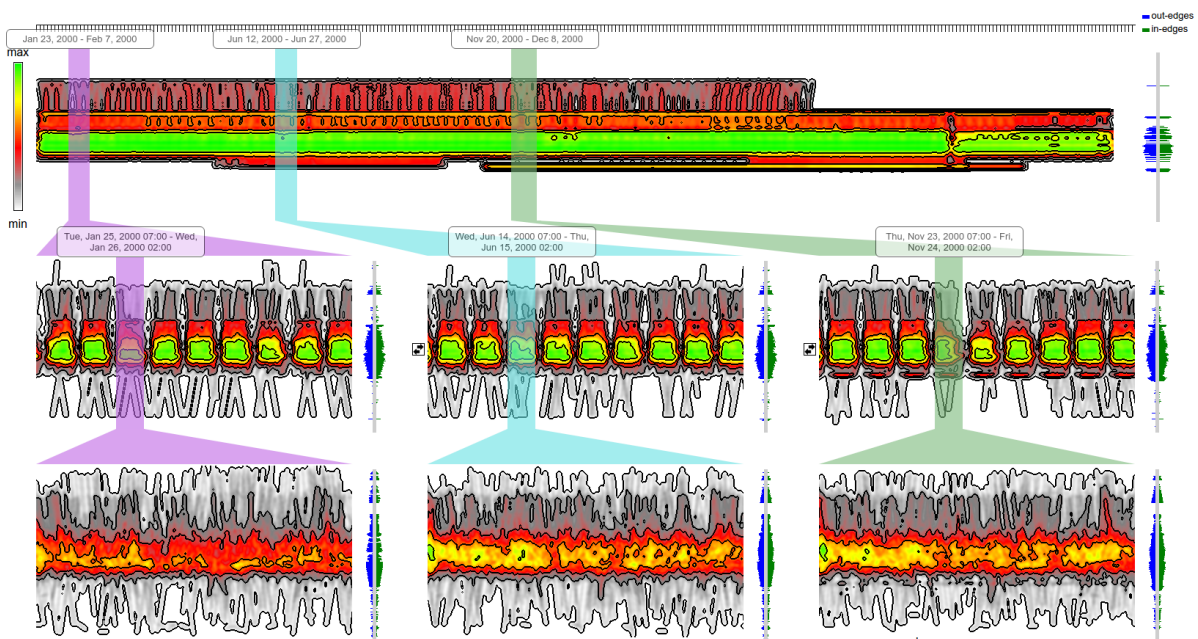


**Figure 3.1:** Applying clustering and reordering techniques to dynamic graph data consisting of several hundred vertices, several thousand edges, and lots of time steps: (a) Raw data (b) After clustering (c) After clustering and reordering



**Figure 3.2:** Clusters hierarchies visualized as dendrograms. (a) Dendrogram generated by the clustering algorithm without reordering. (b) Dendrogram after applying vertex reordering

# 4 MULTI-TIMESCALE DYNAMIC GRAPH VISUALIZATION



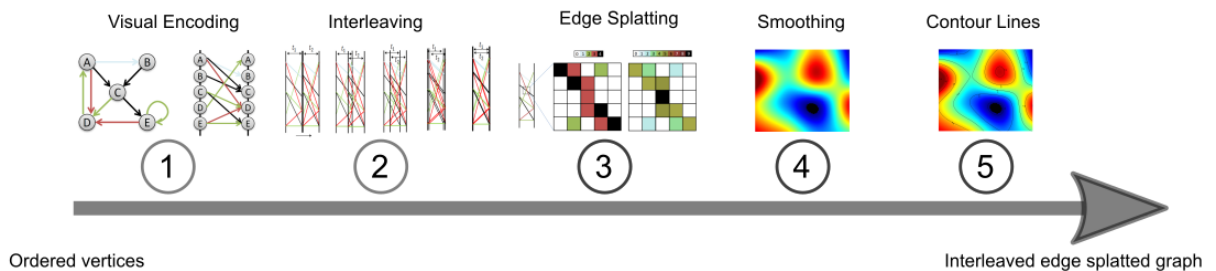
**Figure 4.1:** A multi-timescale view on a dynamic graph dataset acquired from the US domestic flight database consisting of several hundred vertices (airports), several million edges (flight connections), and more than ten thousand time steps (from January 1st, 2000 to December 31st, 2001). Three levels from top to bottom represent monthly, daily, and hourly patterns, respectively.

## 4.1 Introduction

In the previous chapter, the clustering and ordering techniques adopted by our visualization method are presented. In this chapter, we introduce our novel multi-timescale dynamic graph visualization approach. The technique is visually scalable in vertex, edge,

and time dimensions. It is able to show evolving graph structures based on cluster properties, visually improved by a suitable vertex ordering. And it reflects the dynamics of the relational data on multiple time scales. It supports a graph analyst at the task of comparing the graph subsequences while setting them in context to each other. All views are linked to each other helping an analyst to track changes over time in each of them.

Following the information visualization reference model [CMS99], after raw data being transformed, hierarchically clustered and ordered, individual graph sequences go through a visual mapping pipeline (see Figure 4.2). First, they are transformed into bipartite layout then interleaved, splatted, smoothed, and finally, augmented by contour lines (Section 4.2). This is similar to the approach investigated by Burch et al. [BHW17], but we extended it to make it applicable on multiple timescales with inter-linked views and scales (Section 4.3). By doing so, we become able to conduct direct algorithmic comparisons between selected time periods (Section 4.4). The quality of the visual appearance depends on many user-defined parameters that we will be described in Section 4.5.



**Figure 4.2:** A visual mapping pipeline to visualize dynamic graphs with thousands or more time steps. Individual graph sequences are transformed into a bipartite representation, interleaved, splatted, smoothed, and finally, augmented by contour lines.

## 4.2 Visualizing a Sequence of a Thousand Graphs

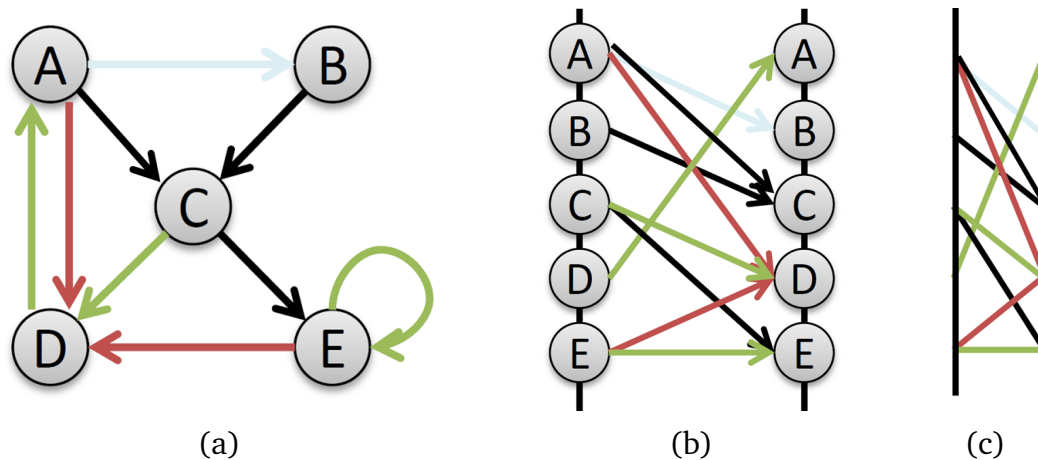
Inspired by the massive sequence view approach [EHBW14] and the scalability and stability effects of the parallel edge splatting technique [BVB+11b], Burch et al. [BHW17] proposed a visualization technique that is particularly useful in visualizing dynamic graphs with thousands or more time steps. The technique is not focusing on showing



every single detail in the dynamic graph but rather on showing the overview of the evolution of graph dynamics over time. It is based on a time-to-space mapping where the graph gets transformed into a bipartite representation, which is particularly useful in preserving the mental map [MELS95; PHG07] and reducing the cognitive efforts while doing the comparison tasks. Moreover, the interaction techniques can be easier integrated into this visualization in comparison to animated graphs where the analyst has to stop and replay the graph several times to gain insights into the data.

Each time step of the graph is represented by a small vertical stripe (see Figure 4.3(c)). To produce a less cluttered visual representation, the splatting technique [BVB+11b] is applied where a scalar density field that aggregates the overlapping edges is computed and displayed. To achieve an even more visually scalable visualization, an interleaving approach is introduced where the graphs are pushed close to each other resulting in overlapping bipartite graphs. Therefore, generating a time-compressed, compact, and space efficient representation while the dynamic graph patterns are still visible.

#### 4.2.1 Encoding of Individual Time Steps



**Figure 4.3:** Transforming a node-link diagram in 2D (a) into a bipartite layout with vertices mapped to vertical lines (b). The vertex labels and arrow heads can be removed for better visual scalability (c).

A central component of our techniques is the transformation of the graph into a bipartite representation. A general graph

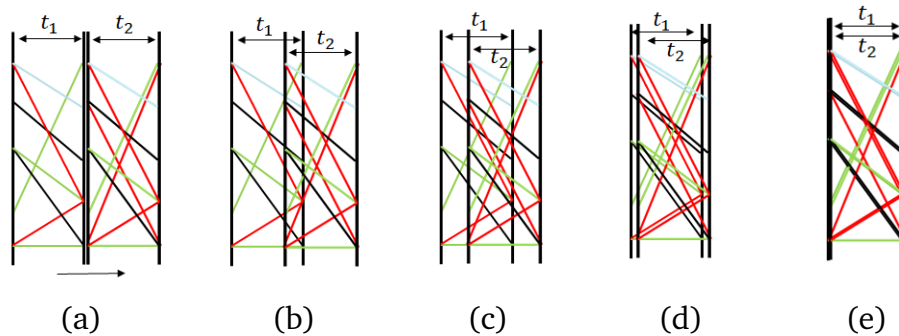
$$G = (V, E)$$

is made artificially bipartite by first copying vertices set  $V$  to another vertex set denoted by  $V_c$ . All vertices from vertex set  $V$  are equidistantly mapped to a vertical axis. A second

vertical axis is used to map all vertices from the copied vertex set  $V_c$ . We use a depth-first approach to traverse the hierarchical clustering tree. we follow subhierarchies according to their order in memory and position the leaf nodes accordingly. For the bipartite layout, it is important that each vertex and its copy is mapped to the same vertical position on both axes. This is particularly important for displaying dynamic graphs to preserve the dynamic stability in the time-to-space mapping.

The bipartite representation has several benefits. It is visually scalable in visualizing dynamic graphs with a large number of time steps. As shown in Figure 4.3(a), the traditional node-link diagram is good at showing the overall graph structures. However, such representation has a limited scalability when it comes to visualizing dynamic graphs with thousands or more time steps. Furthermore, in node-link diagrams, an explicit representation of the edge direction is required (the arrow). Such a representation is not needed in a bipartite layout since the direction is already encoded in the left-to-right reading direction (see Figure 4.3(b)). Even after removing the vertex labels and the arrow heads the diagram still remains readable (see Figure 4.3(c)). Using the bipartite layout we are able to visually encode the graph in one direction while still preserving the graph structures to some degree.

#### 4.2.2 Interleaving



**Figure 4.4:** Interleaving two graphs in a bipartite layout as node-link diagrams. The horizontal distance between the graphs is reduced step-by-step (a) to (d) until we reach a one-pixel offset (e).

The traditional approach for showing dynamic graphs with a bipartite layout is to draw all time steps in a side-by-side layout next to each other [BVB+11b]. Although this concept provides a natural mapping of the time aspect in the dynamic graphs, it is restricted to the size of the display medium. Therefore, it has a limited scalability with time-varying graph data with several hundreds or thousands time steps. A possible remedy could be reducing the width of the vertical stripes. However, the width can only

be reduced to a certain degree. Otherwise, it would be difficult to see the structures of the graph since the density is strongly compressed in the horizontal direction. Furthermore, placing the graph sequences next to each other does not work best for detecting the evolving structures of the graph. Typically, the analyst has to revisit different graph sequences several times until he could obtain insights into the data.

To avoid those problems and to improve the visual scalability, we, therefore, apply a concept that we refer to as *interleaving* (Figure 4.4). We keep a rather large horizontal width of the stripes representing the individual time steps but let the stripes overlap with only one pixel offset. As we can see, the horizontal distance is more and more reduced until all graphs are nearly placed on top of each other, with only one pixel offset. This results in a visual fusion of the links existing in several time steps; they appear as thicker lines. Hence, with every subsequent time step in which a link exists, the thickness of the representing line increases by one pixel due to the offset. Therefore, the thickness or width of the visible structures is an indicator for the time period covered by them. Since we use color coding for representing the edge weights, changes in color can indicate if weight changes occurred.

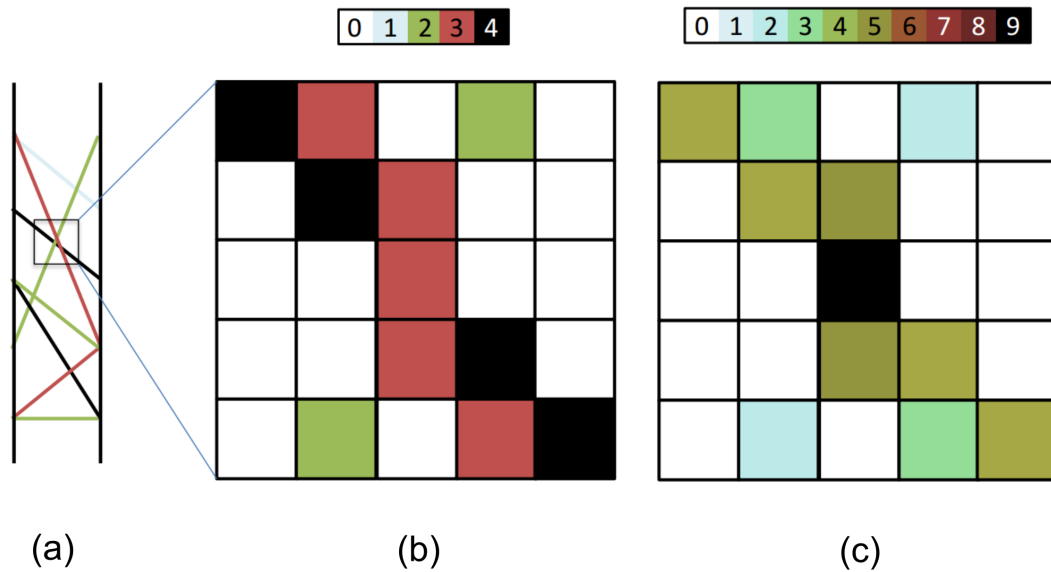
### 4.2.3 Edge Splatting

Edge splatting is a technique proposed by Burch et al. [BVB+11b] to address the problem of visual clutter that occurs when drawing many links onto a small area of screen space. In this technique, the edges are not drawn as line-based graphical primitives. Instead, they are transformed into a pixel-based scalar field. In this way, when two or more edges cross each other at a certain point, an aggregated scalar field of all overlapping edges is computed and displayed.

This is illustrated in Figure 4.5. Directly drawing three crossing links (Figure 4.5 (a)) would result in occluded parts (Figure 4.5 (b)), which might prevent the visibility of edge weights. With our splatting approach, the three links and their edge weights are aggregated into a density field which leads to new quantitative values based on all edge weights (Figure 4.5 (c)). As we can see, the crossing point of the three links is now shown with a larger value because the edge weights of all links are taken into account.

### 4.2.4 Low-Pass Filter

Our technique focused on providing an overview of the dynamic behavior rather than showing every small detail in the dynamic graph. Drawing the interleaved time steps directly can lead to a cluttered view due to the presence of many overlapping and

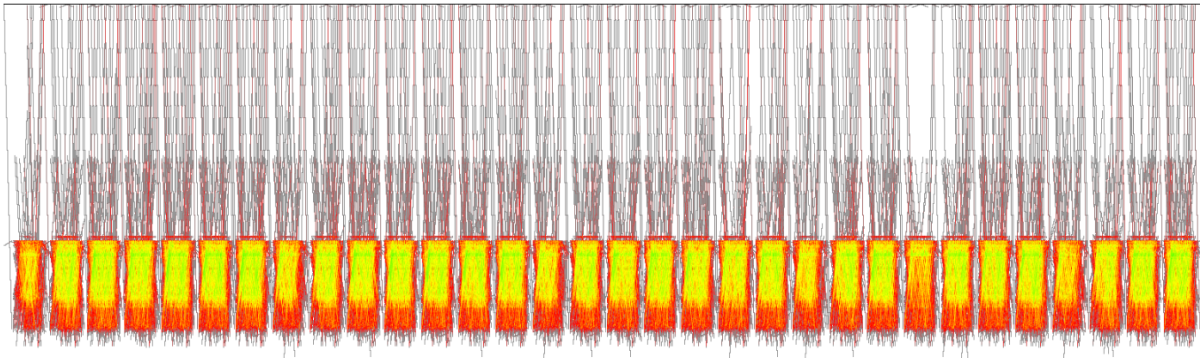


**Figure 4.5:** Edge splatting illustrated for three crossing edges with different weights. (a) Three edges crossing each other. (b) Directly drawing the edges would result in this image; the last drawn edges occlude parts of the others. (c) The edge weights are aggregated into a scalar density field in the rightmost subfigure. Color coding is changed due to a new maximum value [BVB+11b].

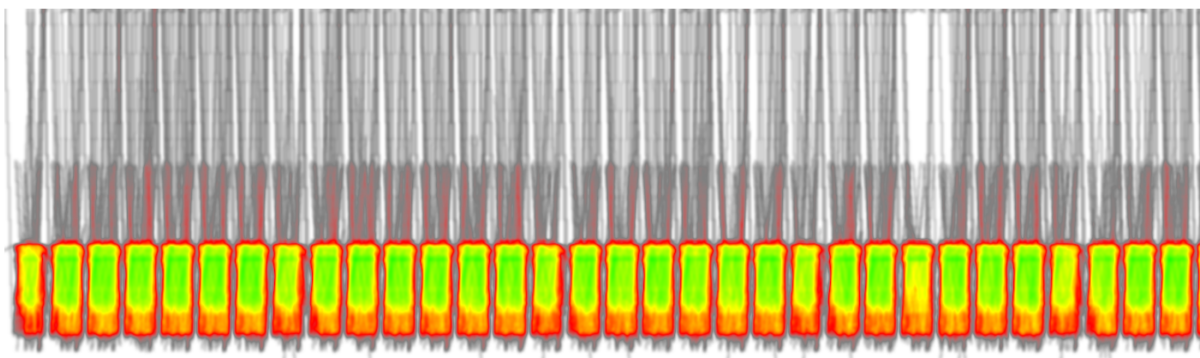
crossing structures. To obtain a less cluttered visual representation, we apply a low pass filter several times to produce a smoother result. Figure 4.6 shows the density field visualization before and after applying the filter. In this example, we used a  $3 \times 3$  box filter since it achieves good results at low computation cost. Figure 4.6(b) shows the result after applying the filter 5 times. As one may notice, the small details are not visible anymore. However, the large-scale structures still preserved.

### 4.2.5 Contour Lines

The visual cortex of our brains contains mechanisms specifically designed to seek out continuous contours [War04]. Considering that, we augmented black contour lines in the density field of our visualization. Therefore, the patterns and structures become easier to detect. Figure 4.7 shows the density field after contour lines augmentation. Notice how contours emphasize the shapes of the visible structures.



(a)



(b)

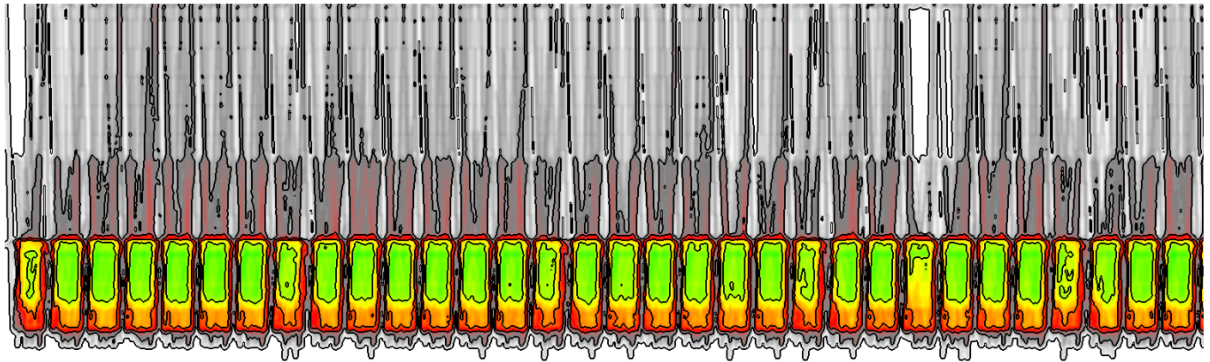
**Figure 4.6:** Smoothing the density field visualization using a  $3 \times 3$  box filter applied 5 times. (a) Before smoothing. (b) After smoothing. Notice the small details are not visible anymore. However, the large-scale structures still preserved.

#### 4.2.6 Color Mapping

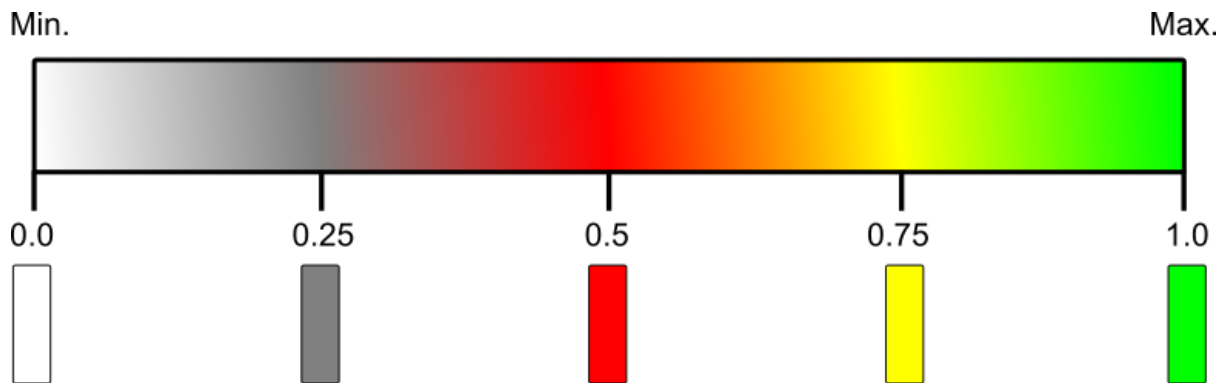
To obtain the final pixels color, A logarithmic function is applied to the pixels scalar weights. Next, all scalar weights are normalized on a scale of 0 to 1. For each pixel, the scaled value is then mapped to a discrete color bar with five bins (see Figure 4.8). Finally, a linear interpolation is done to approximate the final pixel color.

### 4.3 Visualizing on Multiple Time Scales

To provide a static overview of the dynamics of the graph data, we can vertically stack several interleaved diagrams since they are visually scalable. Moreover, even if they are scaled down vertically or horizontally, they still show the overall dynamic patterns, but



**Figure 4.7:** Augmenting contour lines in the density field of our visualization. Notice how contours emphasize the shapes of the visible structures.



**Figure 4.8:** Discrete color bar with five color bins (white, gray, red, yellow, and green).

negatively, individual details are hidden. Hence, the focus of this work is to provide an overview of time-varying patterns on multiple levels, and then supporting a graph analyst by additional interaction techniques to dig deeper into the dynamic graph data. Figure 4.1 illustrates how the stacking works based on user-defined time period selections in each of the time scales. To reach this goal, we provide two different ways of timescale stacking, i.e., top-down stacking and side-by-side stacking.

- **Top-down stacking:** The top-down stacking of time scales describes an order among the time granularities, i.e., from coarse granularities to fine ones going from top to bottom. The visualization is designed in a way to preserve the context of finer time granularities within the coarser granular time granularities by providing guiding clues (see Figure 4.1).
- **Side-by-side stacking:** We display views on the same time scales next to each other on the same level, a concept that is intuitive and that allows a comparison

between them. Moreover, these can be set in context to each other and also with coarser time scales (see Figure 4.1).

The top-down and side-by-side stacking of the timescales result in a hierarchy of timescales in which the vertices are the dynamic graph subsequences and the parent-child relations express the user-defined selections from one timescale to the next one on a finer level.

Following this idea, a graph analyst can start from a coarse-grained temporal overview of the dynamic graph data, identify details and patterns on several levels of temporal granularities and then explore those detailed patterns in all the coarser-grained time scales. The dynamic graph visualization starts from overview to detail and supports the way back, i.e., from detail to overview.

## 4.4 Algorithmic Dynamic Graph Sequence Comparison

Visually comparing dynamic graph sequences is a challenging task, but, the overall dynamic patterns in several of these timescales can be compared easily and rapidly due to the strengths of the perceptual abilities of the humans' visual system [War04]. Negatively, if two dynamic patterns look rather similar, tiny features in the visualization might nonetheless be different which cannot be found due to problems caused by the effect of change blindness [HE12]. To support a graph analyst with a more algorithmic approach we can directly compare two graph subsequences by computing the edge differences for each time step giving a new dynamic graph that typically contains fewer edges than the compared ones unless they have no edge in common.

In our approach we support the comparison of two equally long time periods  $T \subseteq T_T$  and  $T' \subseteq T_T$  ( $|T| = |T'|$ ) like, for example, two days or two hours of a dynamic graph. The individual graphs of each of the subsequences based on those time periods  $T$  and  $T'$  are then algorithmically compared by subtracting the edge set of the second graph  $G'_i$  from the first graph  $G_i$  while  $i \in \mathbb{N}$  denotes the  $i$ -th time step, i.e., the  $i$ -th graph in any of the time periods. The resulting dynamic graph can then be visualized in the same visual metaphor in a different view.

## 4.5 Parameter Adjustments

The visual appearance depends on many user-defined parameters that we will describe in the following.

- **Stripe width:** Changing the width parameter can be useful to make the graph structures more apparent but also the overlap of neighbored graphs in a sequence will be increased with a larger stripe width.
- **Horizontal offset:** Interleaving graphs from a longer sequence means pushing them horizontally together. Depending on the number of graphs, the horizontal display space, and the stripe width, we can achieve different visual pattern results. Typically we apply 1 pixel offset if enough display space is available. If less display space (in pixels) than graphs to be displayed is available we plot them on top of each other and compute densities from the overplotted graphs.
- **Density:** Computing densities can be based on the fact if an edge is present or not, or the edge weight can be incorporated into the density generation. Moreover, the level of detail and the granularity of the timescale plays a crucial role in the density computation.
- **Splattng:** Applying a box filter can be done multiple times which leads to some kind of washing out effect of the sharp contours. The box filter can be changed by interactively adapting the radius of the filter. The box filter can be applied to each timescale separately.
- **Logarithmic values:** If the density values differ a lot in their range we can apply logarithm functions several times to reduce the differences between maximal and minimal values. This effect can be applied to each of the timescales separately which is in particular useful if the node-link diagrams are rather dense and produce lots of hot spots in regions where many links are crossing and overlapping.
- **Clustering/ordering:** Since we are displaying the dynamic graph data on multiple timescales we provide an option to cluster and reorder the graph data on each of those time granularities. This can be particularly useful if one wishes to keep track of an identified cluster structure in a certain time period and how it evolves before or after the selected time period. Moreover, several of those graph structures can be compared, either visually or algorithmically.
- **Color coding:** We support several color scales to visually enhance the computed density fields. The color coding can also be used to map identified clusters in a certain time period to the dynamic data visualization on a different (probably coarser) timescale to get insights about the contextual information.
- **Contour lines:** Only showing the density values as color coded visualizations can be perceptually problematic in particular if the task is to track certain data changes over time. For this reason, we provide contour lines indicating larger regions of similar density values and how they change their shape over time.



- **Horizontal display space:** The display space plays a crucial role for the visual density structures in the dynamic graph visualization. If multiple timescales are used to display the data we probably have less horizontal space to display the selected region and hence, the densities will be different.
- **Vertical display space:** If multiple timescales have to be displayed those have to be mapped to the vertical display space, i.e., stacking many of them means that the vertical display space for each is restricted to only a few pixels, but as a benefit, the generated diagram is visually scalable and hence, allows for the detection of time-varying patterns even if individual details cannot be derived in a coarse-grained view.



# 5 INTERACTION

## 5.1 Introduction

In the previous chapters, we introduced our approach to visualize dynamic graph data on multiple timescales. The dynamic graph is displayed as an interleaved splatted diagram focussing on visual scalability to generate an overview of dynamic graph patterns first. Starting from such an overview, a graph analyst can build explorative hypotheses about the graph data which can be explored in more temporal detail and commented whether hypotheses need to be confirmed, refined, or rejected. This can be achieved by first looking into details and then setting those details into context with the coarser-grained overview representation of the dynamic graph data. To achieve this goal, we integrate several interaction techniques into our solution to help users steering the analysis according to their information need, and to drill down to useful peaces of information.

In this chapter, we present several interaction techniques supported by our visualization. First, we discuss the clustering and ordering techniques in Section 5.2. Next, in Section 5.3, we explore different filtering options. Then, we investigate the usability of the algorithmic compare feature in conducting comparisons between dynamic graph sequences in Section 5.4. Finally, in Section 5.5 we discuss the selection interaction technique.

## 5.2 Exploratory Analytics

The visualization technique is designed in a way to first generate a suitable vertex clustering and ordering based on the entire graph sequence since we do not know how chaotic the evolution of a graph will be right from the beginning. Clusters might be staying constantly over time, they might be growing, shrinking, splitting, merging, or moving to other graph substructures. For this reason we start with a time-aggregated graph clustering and ordering focussing on a preservation of the mental map in the dynamic graph visualization. Consequently, the clusters and ordering cannot be changed in an individual timescale, but typically, on multiple of those.

As a benefit of our approach, we are able to apply a reordering of the vertices in each of the timescales separately, i.e., if the graph structure changes frequently, the ordering can be based on individual time periods while the generated order can also be applied to the entire graph sequence. To explore the quality of each computed vertex ordering we compute the average link lengths based on an initially chosen vertex order. This quality measure describes if the generated vertex clustering and ordering are still appropriate in other time periods or if a new one should be computed. Consequently, the exploratory analysis starts with a cluster evolution overview, provides temporal details while allowing to reorder the vertices based on a certain time period or time step, and can either show the reordered vertices in context to the entire sequence or other selected time periods.



The detailed view can be useful to get an information about individual edges or graph properties that are hidden in the visually scalable dynamic graph visualization. Consequently, the graph analyst can look into spatial and temporal details, select interesting features or graph properties, and show those in the overview view by highlighting the selected details in context to the entire graph sequence. This interaction technique is applicable in any of the timescales. For example, a certain vertex may have various adjacent edges, but this vertex cannot be found in the overview visualization. Once it is found in a detail view, it might be selected, while all adjacent edges can be highlighted in the overview (coarser-grained views) to visually illustrate the evolution of this specific graph property.

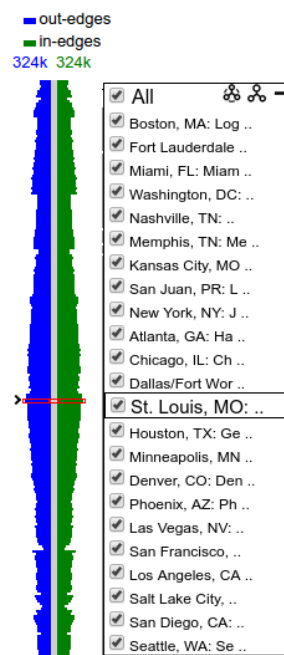
### 5.3 Interactive Filtering



If the graph data is rather dense, many links have to be rendered, i.e., the computed scalar density fields contain many large values that can lead to hiding important graph structures. This is a general drawback of this visually scalable visualization technique. But since we allow interactions with the diagram, we can apply several filter options in order to reduce the amount of visual clutter, overdrawings, and occlusions in the visualizations that would otherwise lead to a degradation of performance at some task while doing an explorative analytics of a large dynamic graph. Those filter options can be particularly based on:

- **Vertices:** Since the vertices are hierarchically clustered and ordered we can select certain subhierarchies or vertex groups that can be filtered out. This can be useful to reduce the amount of nodes that are displayed with all the involved edges, making evolving structures much clearer. As shown in Figure 5.1, the y-axis is used to represent the vertices in our dynamic graph layout, it is depicted as two bar charts placed back-to-back at the right of visualized layout. Green bar-charts encode the frequency of in-edges, whereas blue ones encode the frequency of

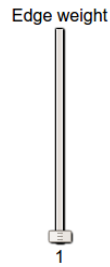
out-edges. The numeric values of those frequencies get displayed at the top of the axis when the analyst hover a certain vertex position. Additionally, a side menu popped-up showing the label of current hovered vertex.

The vertices side menu shows two different options to filter vertices, The *intra-relation*  option filters out edges if one of the endpoints does not belong to the group of selected vertices. While the *inter-relation*  option filters out edges if both end points do not belong to the group of selected vertices.



**Figure 5.1:** The vertices side menu. Two different options to filter vertices are shown on the top-right corner, the inter-relation  and the intra-relation .

- **Edges:** In the bipartite 1D vertex layout, edges can be classified into upward, downward, or horizontal (self) edges. A filter option is provided to select only edges with a certain pointing direction property.
- **Edge weights:** Given a threshold value, edges can be filtered out whose weights lie below the given threshold. This is in particular useful for networks in which the direction and the weights play a crucial role and in which the evolution over time and on several timescales is important. In our tool, we provided the analyst with an edge weight slider (see Figure 5.2) where he can steer the threshold value and directly see the how the graph layout changes along with the vertex list.



**Figure 5.2:** The edge weight slider. Given a threshold value, edges can be filtered out whose weights lie below the given threshold.

- **Edge lengths:** Apart from filtering for edge directions or weights, also the link lengths can be important features to base a filter on. For example, after a clustering and reordering it may be interesting to filter out long links and reorder the vertices.
- **Time:** Since we provide a multi-timescale dynamic graph visualization we support time period selections in any of the timescales. Filtering out several time periods provides more space for the remaining ones and hence, shows a more detailed view on the finer granular temporal graph structures.
- **Densities:** In certain regions the densities are higher than in others. While the evolution of the densities provides an overview about dynamic graph structures it can still be difficult to explore the regions with smaller density values overlapped by others.
- **Textual:** If textual descriptions are given (like airport or function names) those can be filtered, e.g., by searching for a certain substring contained in the labels.
- **Spatio-temporal:** The dynamic graph visualization maps vertices to vertical positions and time to horizontal positions. Hence, a spatio-temporal filter can be applied that only selects a specific time period and a group of connected nodes.

## 5.4 Algorithmic Comparison of Subsequences

The strengths of the human’s visual system and the ability to detect visual patterns, compare them and to derive conclusions about similarities or differences already serve as suitable approaches to find insights in the dynamic graph data. But to fully exploit the strengths of our algorithmical approaches we provide a comparison algorithm that takes two timescales as input and computes a dynamic graph difference. This algorithm might only be applied in cases in which it is not perceptually and visually clear if two

dynamic graph subsequences are similar or not, i.e., it serves as direct support to the visual comparison.

The comparison algorithm can work on any kind of dynamic graph subsequence and generates a difference dynamic graph step by step. This difference graph is then displayed in the same visual metaphor as the original multi-timescale dynamic graphs in a separate view. The algorithm can work on the edges alone, the edge weights, or on the densities.

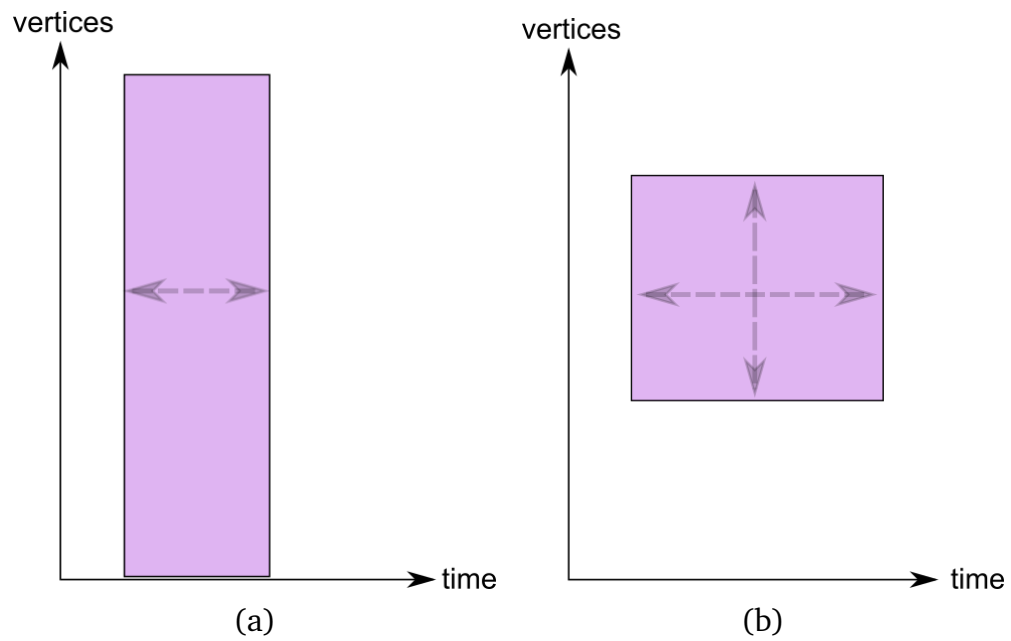
## 5.5 Selection

Selection is a central interaction technique in our visualization since other interaction techniques rely on the selection in the first place (i.e. time filter, spatio-temporal filter, algorithmic comparison). In our visualization tool, there are two modes of selection:

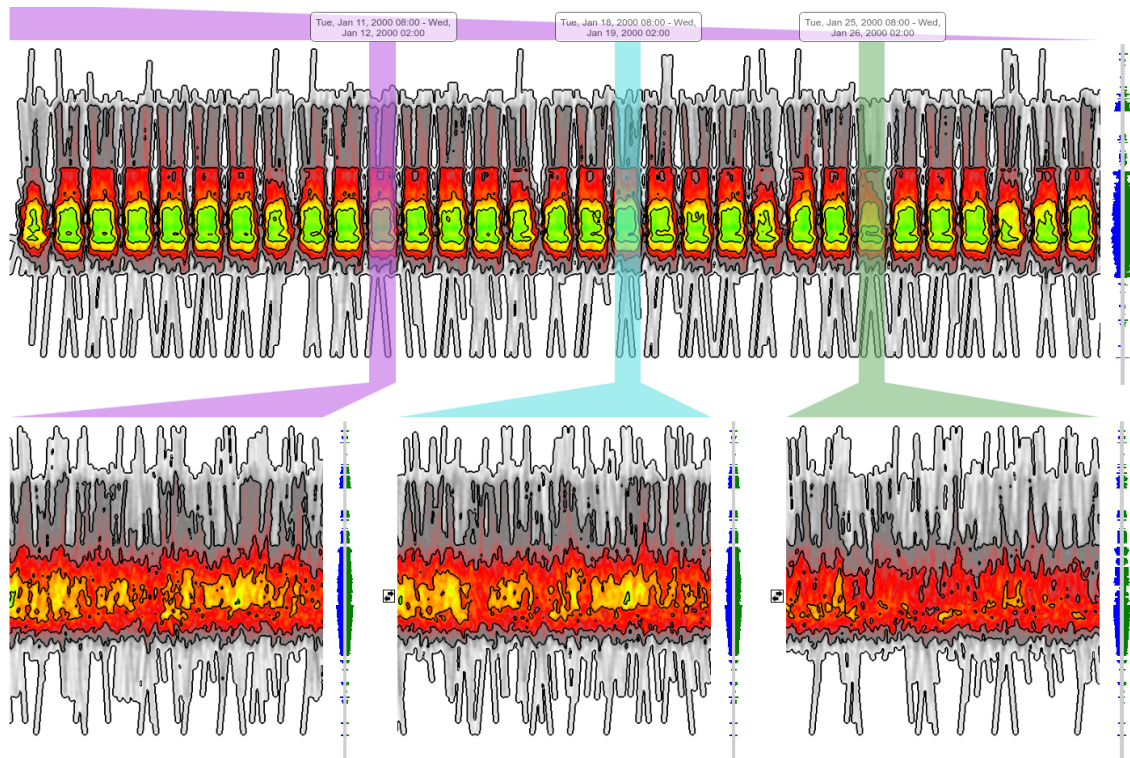
- **Time Selection:** Typically, this mode is used in conjunction with the time filter. It allows the analyst to apply further interaction techniques to the period of time that is bounded by the selection rectangle. In this mode, the selection rectangle can grow and shrink in one dimension only; the time dimension (see Figure 5.3(a)).
- **Space-time Selection:** Similarly, this mode is used in conjunction with the spatio-temporal filter. In this mode, the selection rectangle can grow or shrink in both dimensions; the time and the vertices (see Figure 5.3(b)).

To support side-by-side stacking on the same time scale, multiple selections can be performed on the same time period where each selection has a different color (see Figure 5.4), a concept that is intuitive and allows a comparison between them. Moreover, it supports the tracking of selected periods across multiple timescales.

It may be noted that all interaction techniques can be applied to all timescales separately. For example, applying the filtering on a timescale can show a reduced amount of data points, i.e., the evolving graph structure may become clearer if the right amount of data points is filtered out. Moreover, the remaining data can also be used for vertex reordering and can be propagated to any other timescale view.



**Figure 5.3:** Modes of selection interaction technique. (a) Time Selection (b) Space-time Selection.



**Figure 5.4:** Multiple selections can be performed on the same time period where each selection has a different color.



# 6 APPLICATION EXAMPLE

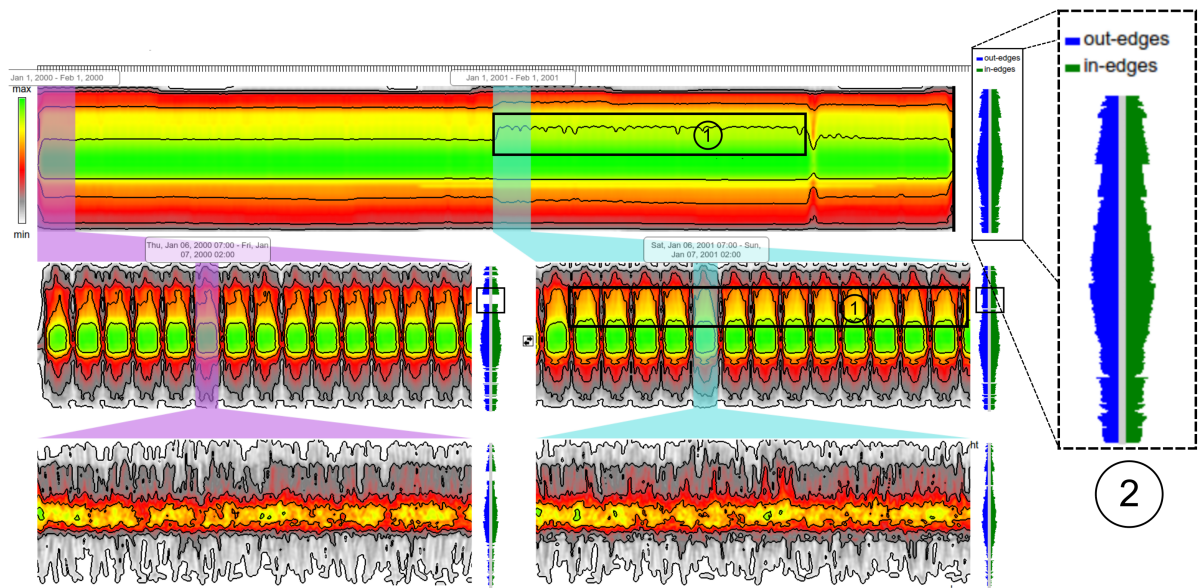
## 6.1 Introduction

In the previous chapter, we discussed different interaction techniques implemented in our system. In this chapter, we illustrate the visual scalability and the usefulness of the multi-timescale dynamic graph visualization by applying it to a portion of the US domestic flight traffic dataset [Tra17]. We extracted the flight dataset for two years starting from January 1st, 2000 and ending at December 31st, 2001. The data is available on a per-minute basis, but we have temporally aggregated the data on an hourly basis to reduce the amount of data in the time dimension. This is possible since changes do not occur that frequently, that we need an exploration on a per-minute basis. The dataset under exploration finally contains 234 vertices which are the airports, more than 10 million weighted edges which are the flight connections and their frequency, and also 17,544 time steps, which are given by the graphs per hour.

In this chapter, we start first by showing how the visualization of the dataset looks like in Section 6.2. Next, we identify different graph structures and temporal patterns in Section 6.3. Finally, we show how the visualization allows us to gain some insights into the dataset by visualizing it on multiple timescales (Section 6.4) and by comparing between differently selected time periods (Section 6.5).


## 6.2 Visualizing Dynamic Graph Data

Getting an overview of any dynamic relational dataset is challenging since most of the existing approaches do not scale well in all three data dimensions, i.e., vertices, edges, and time in combination. Consequently, the real benefit of our technique is to provide a dynamic graph analyst with an overview that is visually scalable on the one hand, but still reflects the evolving graph structures to some degree on the other hand. Figure 6.1 shows the dynamic graph layout for data, after applying the interleaving method illustrated in Figure 4.4. The y-axis represents the vertices (airports), while the x-axis represents time.

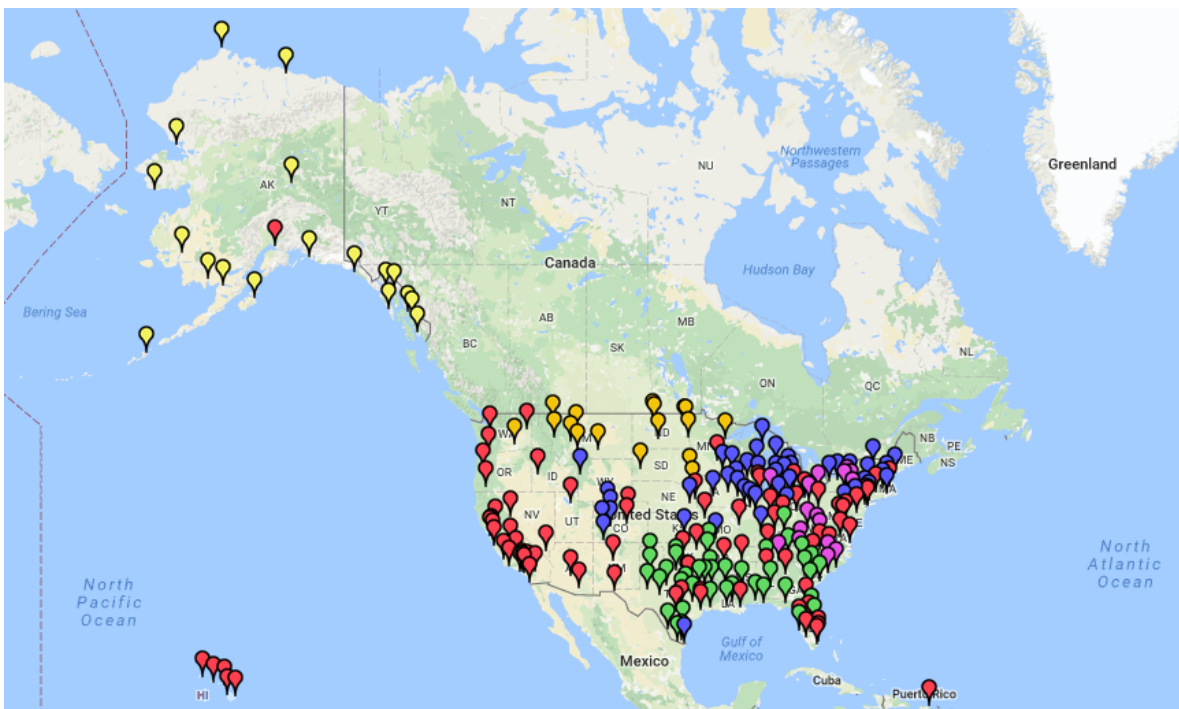


**Figure 6.1:** The dynamic graph visualization can show several views based on multiple timescales. The timescale gets finer from top to bottom, but remains similar from left to right in each level, supporting reliably conducting comparison tasks.

To cluster the vertices hierarchically, we treated the flight connections as weighted edges since we already aggregated the temporal data on an hourly basis. This means that the hierarchical clustering variant from Equation 3.2 in Section 3.3 can be applied to the vertices, which generates a hierarchical organization among the airports (vertices) and groups together those airports that are frequently connected by flights also taking into account the flight frequencies. To further reduce the link lengths and the visual clutter we apply a vertex reordering technique that is based on heuristics of the minimum linear arrangement problem (MinLA), which preserves the hierarchical order, and reduces the sum of link lengths. The resulted dendrogram after applying clustering and ordering algorithms are shown previously in Chapter 3 (see Figure 3.2).

To investigate the correlation between the clusters resulted from the clustering algorithm and the geographical locations of the airports, we visualized the airports of the top six clusters on the map of the United States by depicting them as small colored glyphs , where the airports that belong to the same cluster have the same unique color (see Figure 6.2). As one might notice, there is indeed some correlation between the clusters and the geographical location. For example, all airports belong to the *green* cluster are located in the south east region, while the airports of the *blue* cluster are located in the north east part. Also, out of 19 airports in Alaska, 18 of them belongs to the same cluster (the *yellow* cluster). the *orange* cluster spread over the north area,

while the *purple* one concentrated more in the east region. The *red* cluster is the largest cluster of all six clusters, the busiest airports of US belong to this cluster. While the airports of the west coast region are assigned to the *red* cluster. However, we can also see other red glyphs within the decision regions of other clusters. That is due to the fact that those airports are major airports in these regions.



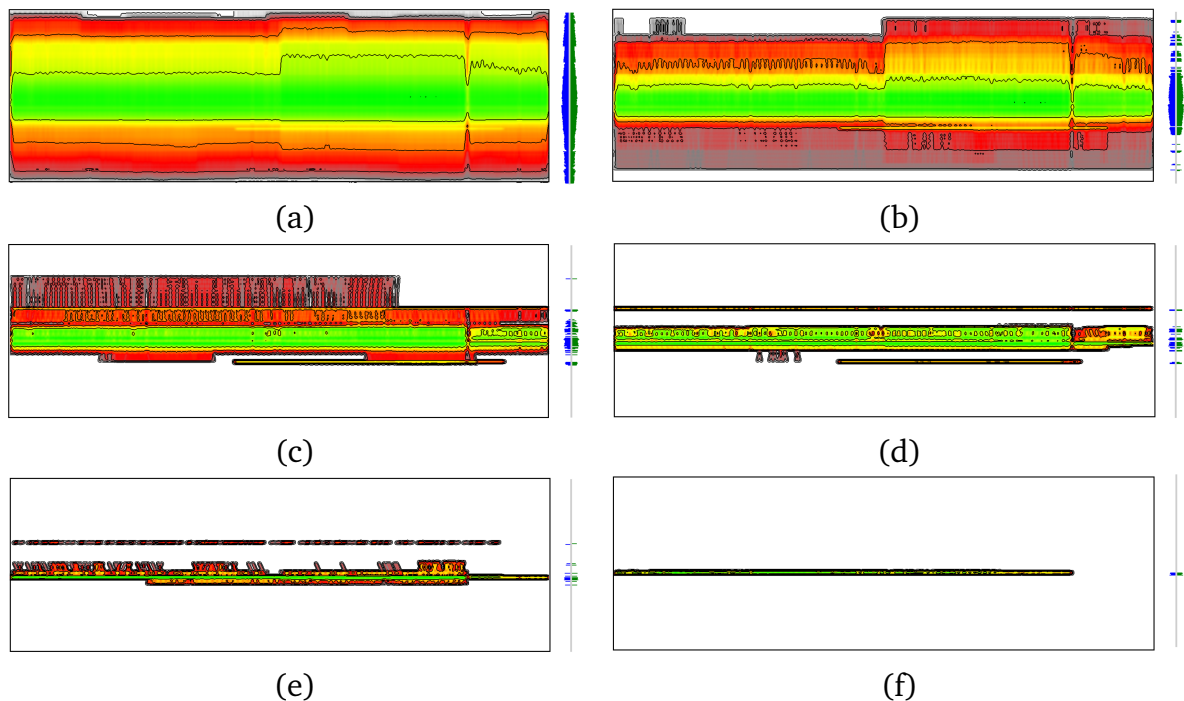
**Figure 6.2:** Clusters visualized on the map of the United States, where the airports that belong to the same cluster have the same unique color.

As shown in Figure 6.1, the ordered airports are placed on the vertical axis at the right of visualized layout. Each airport is depicted as two bar charts placed back-to-back. Green bar-charts encode the frequency of incoming flights (in-edges), whereas blue ones encode the frequency of outgoing flights (out-edges). This helps an analyst get a quick overview of flight frequencies. Figure 6.1 (see annotation ②) shows a symmetric relationship between the frequency of both in-edges and out-edges.

In the generated diagram, although there is no filtering applied yet, one can notice some time-varying structures. We can easily find out that all the airports with maximum number of flights are placed at the center of the graph (the green area), while airports with lesser number of flights are far and grouped together on both the end. By applying filtering techniques and multiple time scales we can spot some interesting patterns as shown in the following sections.

### 6.3 Identifying Spatio-Temporal Patterns

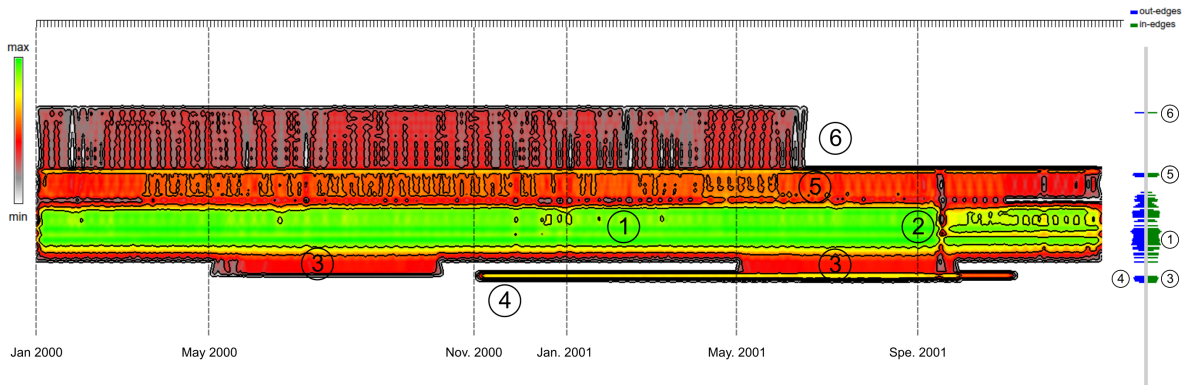
To get rid off too many details in the visualization, we provide interactive filtering for vertices, edge weights, and density values. Figure 6.3 shows the filtered version of the top-level of Figure 6.1 at different thresholds values so that we keep only the edges that have weights greater than or equal the given threshold. Notice the reflection of the chosen threshold on the visibility of vertices of the vertical axis, only a few vertices have an edge weight value greater than or equal 18. Those vertices correspond to the busiest airports in the US, most notably, the airports of *Los Angeles International*, *Phoenix Sky Harbor International*, *Chicago O'Hare International*, *Dallas/Fort Worth International* and *Hartsfield-Jackson Atlanta International*.



**Figure 6.3:** Filtering the graph at different thresholds values (a) Edge weight  $\geq 1$  (b) Edge weight  $\geq 8$  (c) Edge weight  $\geq 18$  (d) Edge weight  $\geq 25$  (e) Edge weight  $\geq 35$  (f) Edge weight  $\geq 35$ .

Filtering at threshold value 18 (see Figure 6.3(c)) reveals some interesting patterns that can be used as a starting point for further analysis. In Figure 6.4, we annotated some of these patterns and utilized different interaction techniques of our visualization tool to further inspect the region of interest. Figure 6.4 (see annotation ①) is a constant pattern over the entire period of 12 months. This pattern is a result of a high number of flight connections between a small group of airports (notice the vertical axis). The busiest airports in the US typically belong to that group. This pattern gets distorted in

### 6.3 Identifying Spatio-Temporal Patterns



**Figure 6.4:** Interesting spatial and temporal patterns can be detected in the dynamic graph visualization although details about individual graphs are not shown.

the last four months of the time period. A closer look at the time axis reveals that this event happened on September 11, 2001, i.e., the terror attacks on 9/11 (see annotation ②), which led to cancellations of the flights in the US for several hours (but also all over the world). We can also see that the flight traffic never really regained the same visual patterns again after 9/11, while some of the frequent connections remain nearly unchanged.

It is also seen that the frequency of flights during the months of May to September in both 2000 and 2001 (see annotation ③) is comparatively higher than other months of the same year. Even after filtering, it persists throughout even for the airports with a fewer number of flights. Looking into details reflects that this pattern is a result of frequent flight connections that go from *Ted Stevens Anchorage Airport* to several airports belongs to the group of the busiest airports, most particularly, the airport of *Seattle/Tacoma International*.

A similar pattern can be seen in annotation ⑤ where the flight connections between the airports of *Dallas Love Field* and *William P Hobby* on one hand and *Austin - Bergstrom International* and *Louis Armstrong New Orleans International* on another, persists throughout even after filtering. This applies also to the pattern in annotation ⑥ which is between *Theodore Francis Green State* airport and *Baltimore/Washington International Thurgood Marshall* airport. However, in this pattern, the frequency of the flights dropped starting from mid of June 2001.

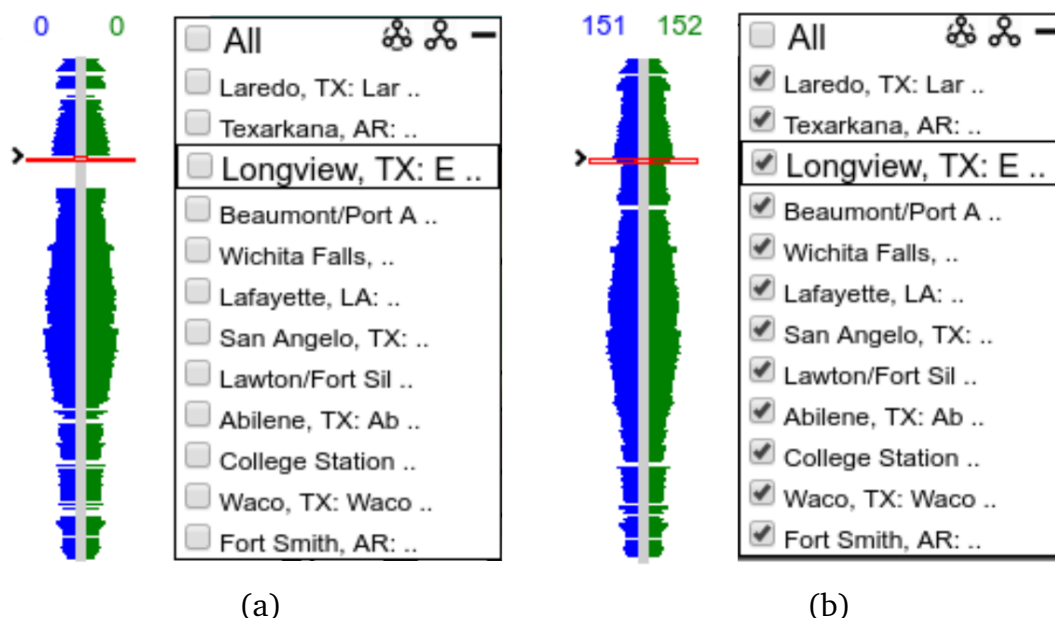
A different pattern can be noticed in annotation ④, which corresponds to a group of flight connections that emerged starting from November 2000. We notice that this pattern did not last for a long time as it started to disappear after 9/11. Looking into detail reveals that these airports (Kona, Honolulu, Lihue, and Kahului) located in Hawai'i.

## 6.4 Multiscale View

Although the overview of dynamic graph visualization can be used to derive interesting visual patterns, it does not support the task of looking into fine-grained temporal time periods. Such a multi-timescale detailed feature is particularly useful if we wish to compare several time periods and set them in context to the larger picture as shown in Figures 4.1 and 6.1. A viewer might be interested in seeing similarities or differences between exactly two or more selected time periods which is difficult to grasp in the overview representation as shown in Figure 6.4, which is too compressed for this task.

Comparing periodic patterns visually on weekly or daily basis can be interesting, hence we decided to select several patterns to show them side-by-side and also stacked on top of each other. In Figure 6.1, January for both 2000 and 2001 is selected and scaled down to a monthly comparison. Looking at both months confirms the patterns we have found previously in the upper level in Figure 6.4.

Comparing the vertical axis of both months, one may notice a group of flight connections that are introduced in January 2001 and were not exist in January 2000. Notice how these new connections changed the graph dynamics in Figure 6.1 (see annotation ①). By looking at the upper level we can confirm that this pattern emerged starting from January 2001 till 9/11. Hovering the vertical axis reveals the list of airports that belong to this pattern as shown in Figure 6.5.

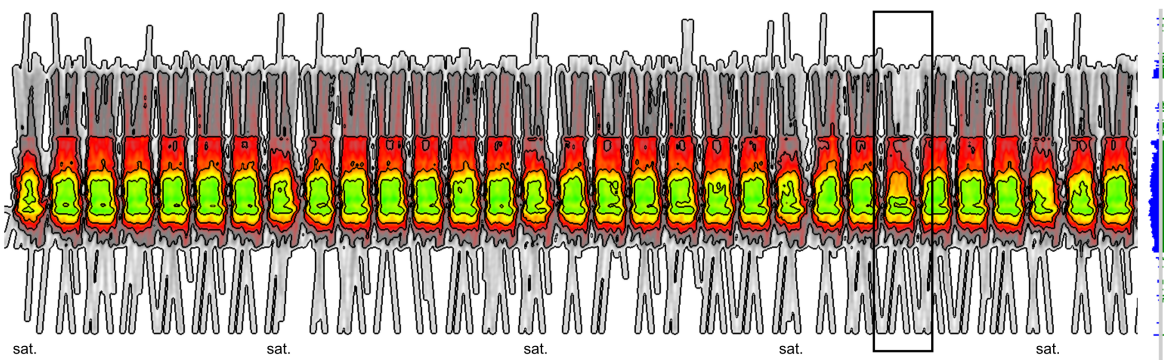


**Figure 6.5:** Comparing the vertical axis of January 2000 and January 2001. (a) January 2000 (b) and January 2001.

Similar to the month level, filtering the graph on the day level highlights the periodic patterns that exist on a daily basis. As we can see in Figure 6.6, the dynamic patterns are visually comparable now since the number of time steps is reduced and hence, the remaining subsequence of the dynamic graph reveals the dynamic patterns more clearly. Comparing daily patterns can show that on specific days there is drop in the flight traffic. Particularly, 25<sup>th</sup> of January, 2000 shows such an event, particularly in the airports of *Newark Liberty International*, *Philadelphia International*, and *LaGuardia*.

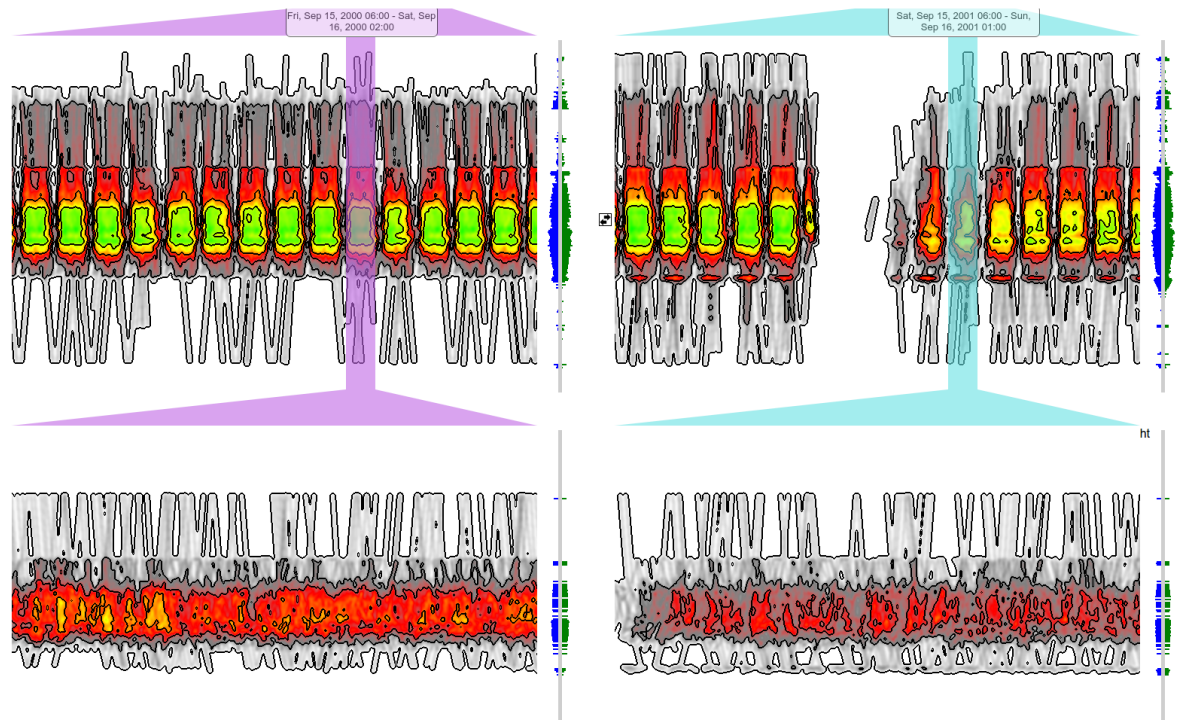
As a hypothesis, we considered bad weather conditions on this day to be blamed for this drop. A request on the world wide web confirmed our hypothesis that a strong winter storm occurred that resulted in a record amount of snow in North Carolina and the Virginia region causing many cancellations.

Figure 4.1 illustrates other abnormal patterns such as 14<sup>th</sup> of June, 2000 and 23<sup>rd</sup> - 24<sup>th</sup> of November, 2000, which correspond to the Thanksgiving public holiday in the US. It is seen that such patterns were aggregated and compressed on the top-level, which makes it difficult to detect without scaling the graph down to the daily basis.



**Figure 6.6:** Filtered flight data for January in 2000 with an unusual pattern behavior on 25<sup>th</sup> of January.

Looking at the monthly comparison between September 2000 and September 2001 shown in Figure 6.7 reflects the differences in the flight behavior after this particular event which becomes clearer and quite evident. Scaling down to an hourly basis for the selected date, i.e., the 15<sup>th</sup> of September for both the years tells the story that even after four days after the attacks, the flight frequency has not regained to normality which we have claimed after looking into the overview of Figure 6.4.



**Figure 6.7:** A visual comparison of September 2000 and September 2001.

## 6.5 Algorithmic Dynamic Graph Comparison

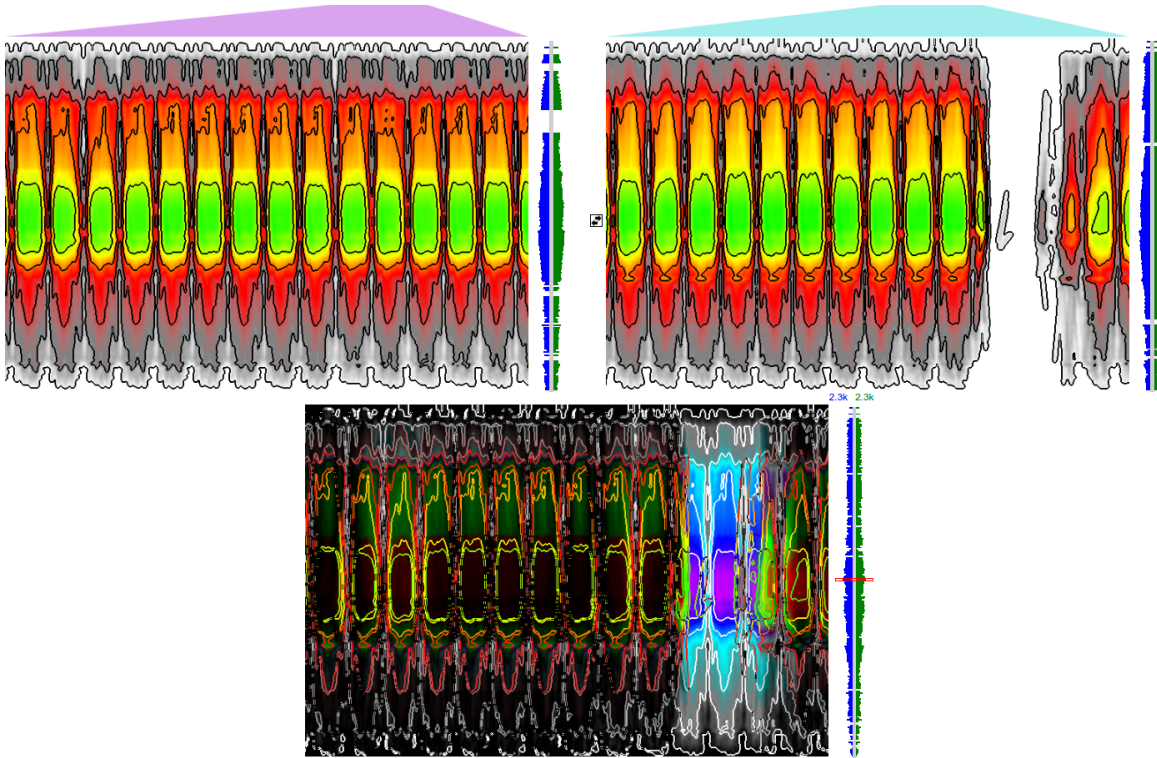
Visually comparing the dynamic data (day by day, week by week) is already useful due to the strengths of perceptual abilities and pattern recognition, but negatively, we cannot detect small changes in the compared figures due to the weak short term memory and effects caused by change blindness [HE12; War04]. Consequently, the visualization is suitable, but as an extension we provide an algorithmic approach that can compare two time periods more exactly and provides the difference in a visualization in the same visual metaphor (see Figure 6.8 and Figure 6.9).

In Figure 6.8, we algorithmically compared two time periods, i.e., the September for the years 2000 and 2001, respectively. Additionally, Figure 6.9 shows algorithmic comparison of three selected Tuesdays (11<sup>th</sup>, 18<sup>th</sup> and 25<sup>th</sup>) of January 2000, respectively. While the difference image between 11<sup>th</sup> and 18<sup>th</sup> shows few changes, the second difference image between 18<sup>th</sup> and 25<sup>th</sup> reflects the drop in flight connections that occurred on 25<sup>th</sup>. This can be observed also by looking at the vertical axis of the difference images.

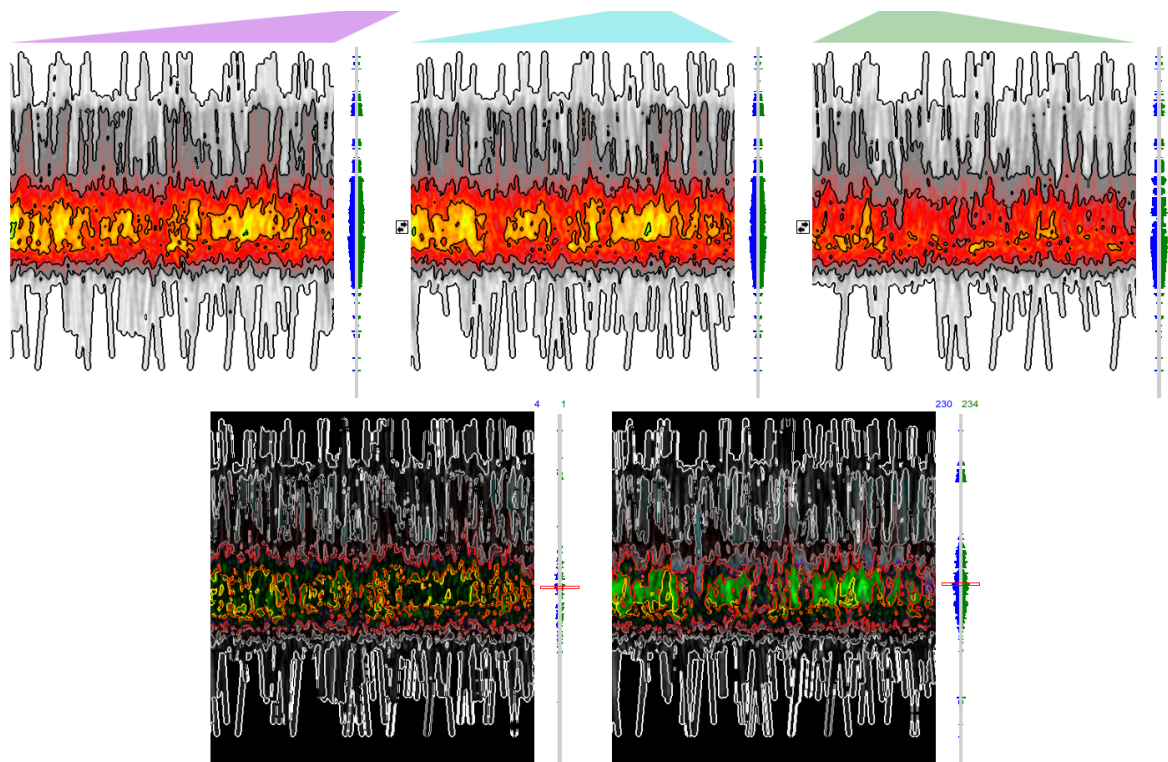
The algorithmic comparison computed several differences in the time periods that are not directly visible by a visual comparison. Also, it shows the difference in terms of the



flight frequency on the vertical axis. The comparison results are displayed in an intuitive way in the same visual metaphor next to the compared time periods.



**Figure 6.8:** Algorithmic comparison between September in both 2000 and 2001. Notice the difference image and the difference in the vertical axis.



**Figure 6.9:** Algorithmic comparison of three selected Tuesdays (11<sup>th</sup> , 18<sup>th</sup> and 25<sup>th</sup>) of January 2000 respectively. Notice the difference between the two images and the difference in the vertical axis

# 7 IMPLEMENTATION DETAILS

## 7.1 Introduction

In the previous chapter, we demonstrated the applicability of our visualization approach using a real world dataset. In this chapter, we provide the implementation details necessary to reproduce the claimed results. First, we discuss the formatting of data in Section 7.2. Next, we introduce the system architecture and the database schema in Section 7.3. The implementation of our system is discussed in Section 7.4. Finally, we review the running time in Section 7.5.

## 7.2 Data Format

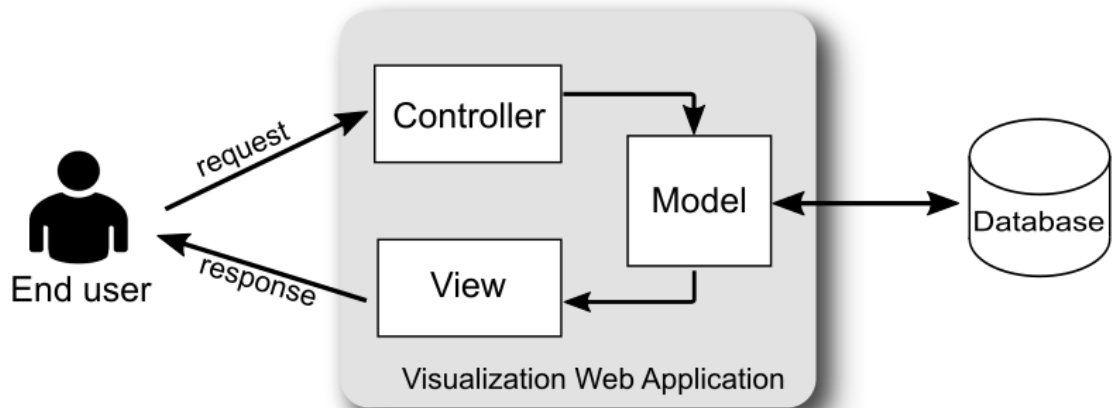
As mentioned in the previous chapter, we used a portion of the US domestic flight traffic dataset [Tra17] to illustrate the scalability and the usefulness of our visualization approach. To import data into our system data files must be formatted correctly with respect to the following:

- The data files must be uploaded in CSV (comma separated values) format.
- The first four columns in the CSV are: the flight date, the origin airport ID, the destination airport ID, and the flight time, respectively.
- First row in the CSV file is reserved for the header.
- The format for the flight date is `yyyy-MM-dd` and the flight time format is `HHmm` (24-hour clock).

### 7.3 System Architecture

In this work, we decided to implement the visualization system as a web-based application. This decision reflects our desire to make the system accessible to the visualization domain experts as well as non-expert people. Additionally, it allows us to benefit from the rich graphical user interface (GUI) capabilities of the web-based applications.

The general architecture of our system is shown in Figure 7.1 the web application is connected to a database. The database is used to establish a mapping between the vertices of the graph and the pixels of the image. So that for each vertex we store the start- and end-points of the outgoing edges along with edges weights. In that way, the visualization images are generated on the fly by querying the database. Such mapping is done for each time level (hours, day, month, year) in a separate database table.



**Figure 7.1:** High level architecture of our visualization system.

Recall Figure 4.2, the steps of clustering, vertex reordering, visual encoding, and interleaving are all completed before edges weights are stored in the database. When the user interacts with the graph, four operations have to be applied to data after being retrieved from the database: the edge splatting, contour lines augmentation, smoothing, and color mapping. To optimize the response time, the system uses a progressive approach to render the graph layout by dividing the layout into smaller segments to be processed and rendered independently. Hence, a responsive user interaction can be maintained.

| Field Name      | Description  |
|-----------------|--|
| code            | The unique ID of the airport between (10001 and 99999) |
| normalized_code | The normalized code value between (1 and 9999)         |
| description     | The name of the airport                                |
| coordinates     | The longitude and latitude coordinates of the airport  |

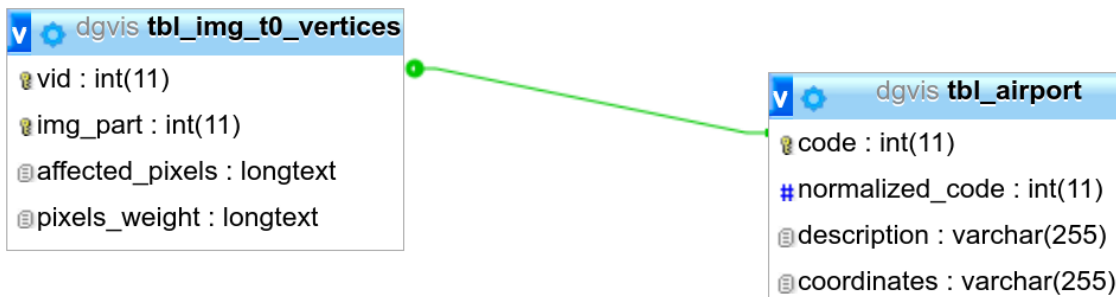
**Table 7.1:** tbl\_airport database table description.

| Field Name    | Description   |
|---------------|---|
| vid           | The vertex ID (airport code)  |
| img_part      | spatial location of the vertex edges with respect to the graph sequence timeline            |
| pixels_weight | The start- and end- points of the vertex outgoing edges stored as semi-colon separated text |
| coordinates   | The weights corresponding to the outgoing edges stored as semi-colon separated text         |

**Table 7.2:** tbl\_img\_t0\_vertices database table description.

### 7.3.1 Database

Figure 7.2 shows the database design of our system. The schema is rather simple. It contains only two tables `tbl_airport` and `tbl_img_t0_vertices`. The description of each table is provided in tables 7.1 and 7.2 respectively.



**Figure 7.2:** The database design of our visualization system.

The `coordinates` column is not available in the original dataset [Tra17], We added this column and filled its data manually for 225 airports in the US to obtain the geographical context of the clustering algorithm result (see Figure 6.2).

| vid   | img_part | affected_pixels  | pixels_weight   |
|-------|----------|--|---|
| 13367 | 0        | $\overbrace{0, 20, 258}^{e_1}; \overbrace{2, 20, 258}^{e_2}; \overbrace{4, 20, 258}^{e_3}; \dots\dots$ $\underbrace{\quad}_{x_1} \quad \underbrace{\quad}_{y_1} \quad \underbrace{\quad}_{y_2} \quad \underbrace{\quad}_{x_1} \quad \underbrace{\quad}_{y_1} \quad \underbrace{\quad}_{y_2} \quad \underbrace{\quad}_{x_1} \quad \underbrace{\quad}_{y_1} \quad \underbrace{\quad}_{y_2} \quad \dots\dots$ | $\overbrace{2}^{w_1}; \overbrace{4}^{w_2}; \overbrace{5}^{w_3}; \dots\dots$ |
| ⋮     |          |  |   |

**Figure 7.3:** `tbl_img_t0_vertices` maps between the vertices and outgoing edges.

The `tbl_img_t0_vertices` table used to establish the mapping between the vertices and image pixels. The `affected_pixels` column contains the outgoing edges for each vertex. The edges are saved semi-colon separated. For each edge, we store the start and end-points. The `pixels_weight` column store the scalar weight associated with each edge. The entries of this column are semi-colon separated and maintain the same entry order as in `affected_pixels` column (see Figure 7.3). There is no need to store the  $x$  position of the end point since it can be directly driven from the  $x$  position of the start point plus the stripe width.

The `img_part` column is used to store the spatial location of the outgoing edges with respect to the entire graph sequence. We virtually subdivide the timeline of the graph sequence into short intervals, called parts, each part is 1000 pixels width. In that way, edges that belong to the first 1000 pixels (from 0 to 999) of the timeline have `img_part` value 0. Similarly, edges that belong to the next 1000 pixels (from 1000 to 1999) of the timeline have `img_part` value 1 and so on. By querying the database using specific `img_part`, we reduce the size of the returned edges significantly since we only retrieve those edges correspond to the current explored time interval. Thus, optimizing the response time.

To establish the mapping between the vertices and edges pixels on multiple timescales, the table `tbl_img_t0_vertices` is duplicated for each time scale separately. Notice that such design is in favor of optimizing the query time rather than normalizing the database tables. Also, `tbl_img_t0_vertices` table is structured that way to support the vertices filtering in an effective way. Finally, the current database design is not final and up to changes depending on system functionalities to be implemented in the future.

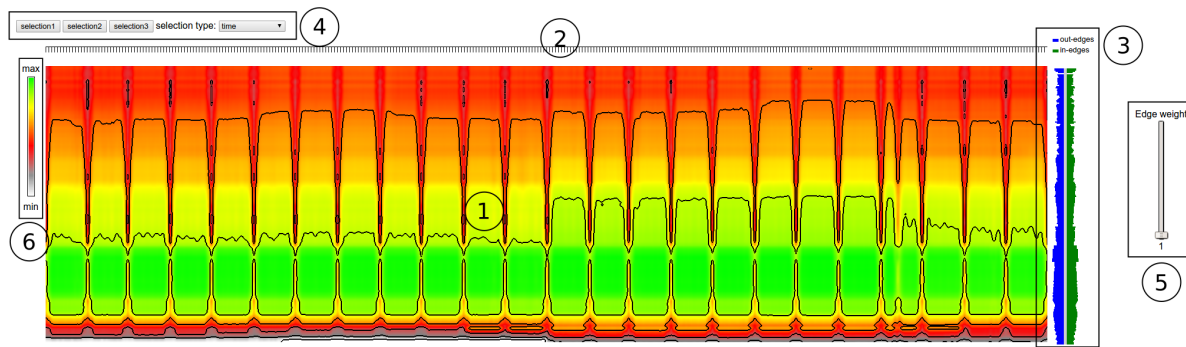
## 7.4 Implementation

To implement the system, we followed an experimental approach in which a prototype is developed and continuously enhanced and improved to reach our proposed design goals. Figure 7.4 shows the main screen of our system. We tried to keep a simple Model-view-controller (MVC) architecture. The core system functionalities are implemented in the model class `DynamicGraph`. There are two controllers to handle user

|                             |   |
|-----------------------------|---|
| <b>Programming Language</b> | Java 8                                  |
| <b>Database</b>             | 10.1.9-MariaDB                          |
| <b>Server Side</b>          | Java Servlet 3.0                        |
| <b>Client Side</b>          | HTML, CSS, Javascript, and JQuery 3.1.1 |
| <b>Web Server</b>           | Tomcat v8.0                             |

**Table 7.3:** Technologies used during the implementation phase.

requests, The first controller `getDynamicGraph` for handling the initialization process. i.e., importing CSV files, building the graph, and populating database tables. The second controller `getImageSegment` to handle the user interactions with the graph layout. i.e., filtering, selection, scale down. For the view component, we designed a single HTML page to display the visualization results. Figure 7.6 shows code snippet of `getDynamicGraph` controller.

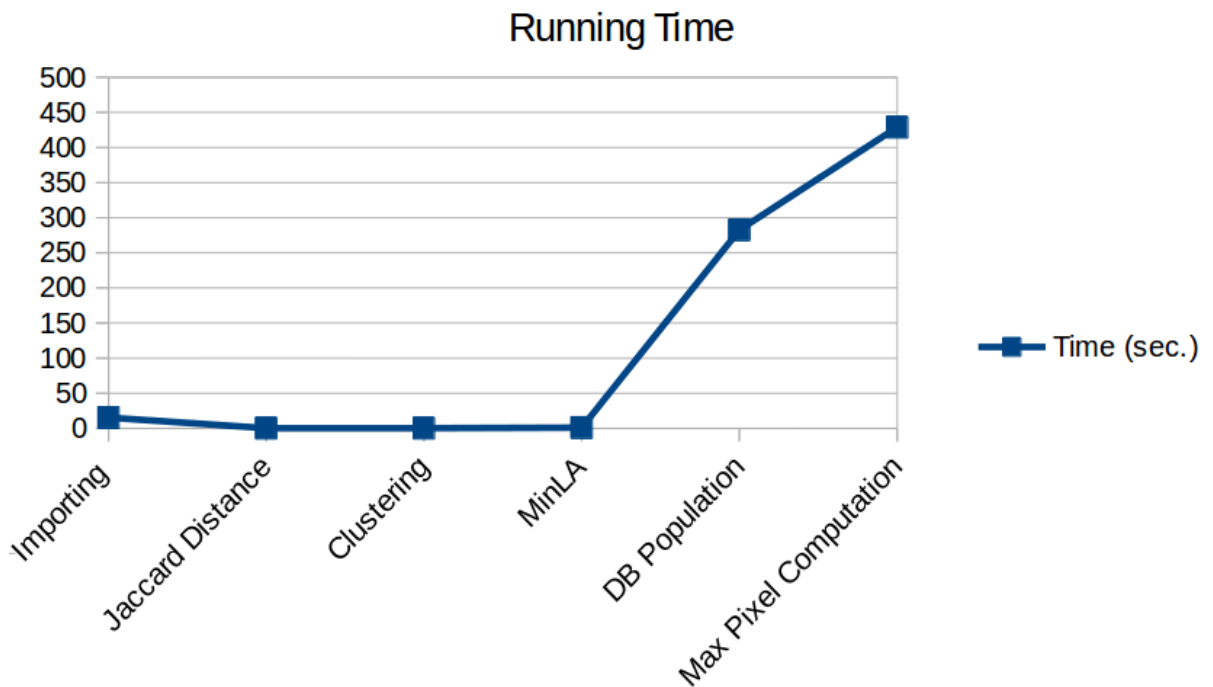


**Figure 7.4:** The main screen of our system. (1) Main graph layout (2) Time axis (3) Vertices axis (4) Selection controls (5) Edge weight slider (6) Color legend

Table 7.3 list the technologies used during the implementation phase. The core functionalities of the system are implemented using Java programming language. For the hierarchical clustering module, we used an existing open source Java implementation [lbe]. The web application is connected to MariaDB database. To keep the consistency with Java technology, we used Java Servlet as server-side technologies and HTML, CSS, Javascript, and jQuery as client-side technologies. The compiled source files are later deployed to a Tomcat web server. To give back to the visualization community, we made the source code along with the database script online available for download [Abd].

## 7.5 Running Time

Figure 7.5 shows the running times of different initialization tasks. The results this is obtained by running our implementation on a portion of the US domestic flight traffic dataset [Tra17]. We extracted the flight dataset for two years starting from January 1st, 2000 and ending on December 31st, 2001. The dataset contains 234, more than 10 million weighted edges, and 17, 544 time steps. We used an Intel® Core™ i7-4710HQ CPU machine with 7,7 GiB memory and GeForce GTX 860M graphics. We utilized the batch processing to optimize the time of database population task. Although it takes several minutes to complete, it is tolerable since it is executed as a one-off job.



**Figure 7.5:** The running times of initialization tasks.



```

DynamicGraph.java  getImageSegment  getDynamicGraph  result.html  main.js  common.js  web.xml  MinLAVertexOrder
99
100 // compute jaccard distance
101 startTime = System.currentTimeMillis();
102 int k=0; // matrix index
103 double outSim,inSim;
104 for(i=0;i<vertNames.length-1;i++){
105     int vertI = Integer.parseInt(vertNames[i]);
106     for(int j=i+1;j<vertNames.length;j++){
107         int vertJ = Integer.parseInt(vertNames[j]);
108         outSim = Common.getWeightedJaccardDistance2(vertexMap.get(vertI).getOutNeighbors(), vertexMap.get(vertJ).getOutNeighbors(),
109             vertexMap.get(vertI).getOutNeighborsWeights(), vertexMap.get(vertJ).getOutNeighborsWeights()); // out-Neighbor
110         inSim = Common.getWeightedJaccardDistance2(vertexMap.get(vertI).getInNeighbors(), vertexMap.get(vertJ).getInNeighbors(),
111             vertexMap.get(vertI).getInNeighborsWeights(), vertexMap.get(vertJ).getInNeighborsWeights()); // in-Neighbors
112         pdist[0][k] = (outSim + inSim)/2.0; // similarity is computed as the average between the two similarities
113         k++;
114     }
115 }
116 endTime = System.currentTimeMillis();
117 System.out.println("compute jaccard distance took " + (endTime - startTime) + " milliseconds");
118 System.out.flush();
119
120
121 // cluster
122 int order = 0; // leaf order in the tree
123 startTime = System.currentTimeMillis();
124 ClusteringAlgorithm alg = new PDistClusteringAlgorithm();
125 cluster = alg.performClustering(pdist, vertNames, new AverageLinkageStrategy());
126 endTime = System.currentTimeMillis();
127 System.out.println("cluster took " + (endTime - startTime) + " milliseconds");
128 System.out.flush();
129
130
131 // reorder vertices according to MinLA successive algorithm
132 startTime = System.currentTimeMillis();
133 cluster = MinLAVertexOrder.reorderClusterHierarchy(cluster, vertexMap);
134 //cluster.appendLeafNames(cluster.getClusterLeavesByTraversing());
135 endTime = System.currentTimeMillis();
136 System.out.println("MinLA took " + (endTime - startTime) + " milliseconds");
137 System.out.flush();
138
139
140
141 for(String leaf : cluster.getClusterLeavesByTraversing()){

```

Figure 7.6: Code snippet of getDynamicGraph controller.



# 8 CONCLUSIONS

## 8.1 Summary

In the previous chapter, we presented the implementation details necessary to reproduce the claimed results of this thesis. In the following lines, we provide a brief summary of each chapter. Next in Section 8.2 we discuss scalability issues of our proposed dynamic graph visualization technique. Finally, we discuss the directions for the future work in Section 8.3.

In Chapter 1, we introduced the motivation, the research question, and the main contributions of this thesis. The most recent ideas and methods in the area of dynamic graph visualization are explored in Chapter 2. We discussed three major concepts for the representation of the time aspect in graphs. i.e., time-to-time mappings, time-to-space mappings, and hybrids, with more focus towards time-to-space mapping approaches.

In Chapter 3, we presented the preprocessing phase of dynamic graph data. First, we introduce the data model to get a clear picture of vertices and edges in the graph layout. Then, we show how, by applying the clustering and reordering techniques, we could obtain a good visual representation. In Chapter 4, the corner-stone in this thesis, we proposed our multi-timescale dynamic graph visualization approach. We explained different steps of the algorithm and showed how it is applicable on multiple timescales with inter-linked views and scales.

The interaction techniques supported by our system are introduced in Chapter 5. We presented the clustering and ordering techniques followed by different filtering options. Finally, we investigated the usability of our technique in conducting comparison tasks between dynamic graph sequences, both visually and algorithmically. In Chapter 6, we illustrated the visual scalability and the usefulness of our visualization approach by applying it to a real world dataset. Finally, in Chapter 7, we provided the implementation details necessary to reproduce the results claimed in this thesis.

## 8.2 Discussion and Limitations

We designed a visually scalable dynamic graph visualization based on the visual metaphor of node-link diagrams and on a bipartite layout of the graph vertices. We applied splatting, smoothing, and augmented the generated scalar density fields with contour lines to achieve a perceptually enhanced visualization to rapidly identify evolving graph structures. Those structures were generated by clustering and ordering techniques while we displayed multiple timescales on different vertically stacked layers, also in side-by-side views which benefit from visual comparison tasks. After having experimented with many layout, clustering, ordering, and rendering parameters, we identified a list of possible scalability and limitation aspects worth mentioning.

- **Graph layout:** The bipartite layout is useful since it produces a visually scalable 1D layout of the vertices that are suitable for aligning them over time on horizontal lines. This generates a high degree of dynamic stability and hence, the mental map can be preserved in this time-to-space mapping of a dynamic graph. Negatively, the bipartite layout lives in a more or less one-dimensional display space leading to an increase of visual clutter caused by a higher probability of link crossings. For this reason, edge splatting has to be applied to regain the otherwise cluttered structures in the graph dynamics.
- **Clustering and Ordering:** Visualizing the raw data without any preprocessing can already give some insights in the dynamics of the data, but, a clustering and ordering is definitely useful to generate more visual structures due to the reduced amount of visual clutter. A problematic issue in this respect is the time period on which the clustering and ordering is based. If a dynamic graph behaves rather chaotically over time, there is no global good clustering or ordering the visualization can be based on.
- **Rendering:** Many graphs in a sequence produce many vertically dense stripes leading to an occlusion of dynamic graph patterns. If the stripe widths, splatting parameters, or color codings are not chosen in an adequate way, this may lead to missing patterns in the visualization. A similar aspect holds for the multiple timescales stacked on top of each other which produce smaller horizontal stripes and hence, a vertical compression of the individual timescales. The display space a dynamic graph sequence is rendered on has an influence on the density patterns, but positively, they are treated similarly over time, still allowing to visually compare the dynamics in a graph.
- **Data Comparison:** Visually comparing dynamic graph subsequences can be done rapidly due to the strengths of the humans' perceptual abilities and fast pattern recognition. But, if only a few pixels are the difference between two dynamic graph

visualizations, those can hardly be recognized in a visual way. As an additional technique we provide an algorithmic comparison, but as a drawback, this can only be applied to well-defined dynamic patterns, while a more general approach based on temporal graph structures would be more suitable, but also more time-complex.

- **Interaction:** Interacting with the dynamic graph visualization is an important feature since we can start with an overview about the dynamics and then look into details step by step. Although several interactions like selections, filtering, brushing and linking, or details on demand are implemented, we are aware of the fact that many more should be added in future. With growing datasets in vertex, edge, and time dimensions, such interaction techniques have to be carefully implemented and based on well-defined effective data structures to guarantee an interactive visualization tool.

## 8.3 Conclusion and Future Work

In this thesis we described a multi-timescale dynamic graph visualization based on interleaved splatted bipartite node-link graph visualizations. The computed diagrams are visually scalable on the one hand and are able to reflect varying graph structures to some degree on the other hand. To achieve clearer visual structures we preprocessed the dynamic graph data by applying a hierarchical clustering as well as a reordering of the vertex positions focussing on reducing the sum of link lengths and hence, visual clutter. The major focus of this work is that a dynamic graph can be displayed in an overview representation although it contains many vertices, edges, and time steps. Based on such an overview, several time levels can be displayed at the same time, providing an overview of more than one time granularity of the graph data while setting all views in context to each other. The benefit is consequently, that a graph analyst can do comparison tasks between graph subsequences on multiple timescales while still preserving the overview in temporally long and large graph datasets.

For the future work, there are four possible directions that could be taken to extend the current work:

- Evaluating the current visualization approach by comparing it to a multi-timescale visualization in which all timescales are shown in separate views.
- Designing an algorithm that automatically generates a default setting for the multiple timescales freeing the graph analyst from initially deciding himself about the time granularities.

## 8 CONCLUSIONS

---

- Implementing different clustering and reordering strategies and investigating the results.
- Implementing new interaction techniques based on well-defined effective data structures to guarantee an interactive visualization tool.
- Further assessing the capability of our visualization approach by applying it to other artificial and real-world dynamic datasets.

# Bibliography

- [Abd] Abdelaal. *githubRepo/DynamicGraphVis at master · mottazabdefattah/githubRepo*. <https://github.com/mottazabdefattah/githubRepo/tree/master/DynamicGraphVis>. (Accessed on 07/31/2017) (cit. on p. 71).
- [AMST11] W. Aigner, S. Miksch, H. Schumann, C. Tominski. *Visualization of Time-Oriented Data*. Human-Computer Interaction Series. Springer, 2011. ISBN: 978-0-85729-078-6 (cit. on p. 28).
- [APP11] D. Archambault, H. C. Purchase, B. Pinaud. “Animation, Small Multiples, and the Effect of Mental Map Preservation in Dynamic Graphs.” In: *IEEE Transactions on Visualization and Computer Graphics* 17.4 (2011), pp. 539–552 (cit. on p. 22).
- [BBDW17] F. Beck, M. Burch, S. Diehl, D. Weiskopf. “A Taxonomy and Survey of Dynamic Graph Visualization.” In: *Computer Graphics Forum* 36.1 (2017), pp. 133–159. ISSN: 1467-8659 (cit. on p. 21).
- [BBV+12] F. Beck, M. Burch, C. Vehlow, S. Diehl, D. Weiskopf. “Rapid Serial Visual Presentation in dynamic graph visualization.” In: *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2012, pp. 185–192 (cit. on p. 28).
- [BD08] M. Burch, S. Diehl. “TimeRadarTrees: Visualizing Dynamic Compound Digraphs.” In: *Computer Graphics Forum* 27.3 (2008), pp. 823–830 (cit. on pp. 25, 26).
- [BHW17] M. Burch, M. Hlawatsch, D. Weiskopf. “Visualizing a Sequence of a Thousand Graphs (Or Even More).” In: *Computer Graphics Forum* (2017) (cit. on pp. 17, 18, 27, 28, 40).
- [BPF14] B. Bach, E. Pietriga, J.-D. Fekete. “Visualizing Dynamic Networks with Matrix Cubes.” In: *CHI Conference on Human Factors in Computing Systems*. 2014, pp. 877–886 (cit. on p. 23).
- [BRD+15] B. Bach, N. H. Riche, T. Dwyer, T. M. Madhyastha, J. Fekete, T. J. Grabowski. “Small MultiPiles: Piling Time to Explore Temporal Patterns in Dynamic Networks.” In: *Computer Graphics Forum* 34.3 (2015), pp. 31–40 (cit. on pp. 23, 24).

- [Bur16] M. Burch. “Isoline-Enhanced Dynamic Graph Visualization.” In: *Proceedings of the International Conference on Information Visualisation (IV)*. 2016, pp. 1–8 (cit. on p. 28).
- [BVB+11a] M. Burch, C. Vehlow, F. Beck, S. Diehl, D. Weiskopf. “Parallel Edge Splatting for Scalable Dynamic Graph Visualization.” In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2344–2353 (cit. on pp. 26, 28).
- [BVB+11b] M. Burch, C. Vehlow, F. Beck, S. Diehl, D. Weiskopf. “Parallel edge splatting for scalable dynamic graph visualization.” In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2344–2353 (cit. on pp. 40–44).
- [CHZ+07] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. J. Van Wijk, A. Van Deursen. “Understanding execution traces using massive sequence and circular bundle views.” In: *Program Comprehension, 2007. ICPC’07. 15th IEEE International Conference on*. IEEE. 2007, pp. 49–58 (cit. on p. 25).
- [CKN+03] C. S. Collberg, S. G. Kobourov, J. Nagra, J. Pitts, K. Wampler. “A System for Graph-Based Visualization of the Evolution of Software.” In: *Proceedings ACM 2003 Symposium on Software Visualization*. 2003, pp. 77–86, 212–213 (cit. on p. 22).
- [CMS99] S. K. Card, J. D. Mackinlay, B. Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999 (cit. on pp. 31, 40).
- [DG02] S. Diehl, C. Görg. “Graphs, They Are Changing.” In: *Proceedings of the International Symposium on Graph Drawing*. 2002, pp. 23–31 (cit. on pp. 21, 22).
- [DGK01] S. Diehl, C. Görg, A. Kerren. “Preserving the Mental Map using Foresighted Layout.” In: *VisSym*. 2001, pp. 175–184 (cit. on p. 21).
- [EHBW14] S. van den Elzen, D. Holten, J. Blaas, J. J. van Wijk. “Dynamic network visualization with extended massive sequence views.” In: *IEEE transactions on visualization and computer graphics* 20.8 (2014), pp. 1087–1099 (cit. on pp. 25, 26, 40).
- [EHBW16] S. van den Elzen, D. Holten, J. Blaas, J. J. van Wijk. “Reducing Snapshots to Points: A Visual Analytics Approach to Dynamic Network Exploration.” In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (2016), pp. 1–10 (cit. on pp. 25, 26).
- [FT04] Y. Frishman, A. Tal. “Dynamic Drawing of Clustered Graphs.” In: *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*. 2004, pp. 191–198 (cit. on pp. 21, 22).



- [GJ79] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN: 0-7167-1044-7 (cit. on p. 34).
- [HBW14] M. Hlawatsch, M. Burch, D. Weiskopf. “Visual Adjacency Lists for Dynamic Graphs.” In: *IEEE Transactions on Visualization and Computer Graphics* 20.11 (2014), pp. 1590–1603 (cit. on pp. 23, 24).
- [HE12] C. G. Healey, J. T. Enns. “Attention and Visual Memory in Visualization and Computer Graphics.” In: *IEEE Transactions on Visualization and Computer Graphics* 18.7 (2012), pp. 1170–1188 (cit. on pp. 28, 47, 64).
- [lbe] lbehnke. *lbehnke/hierarchical-clustering-java: Implementation of an agglomerative hierarchical clustering algorithm in Java. Different linkage approaches are supported.* <https://github.com/lbehnke/hierarchical-clustering-java>. (Accessed on 07/30/2017) (cit. on p. 71).
- [LKS+11] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J. Fekete, D. W. Fellner. “Visual Analysis of Large Graphs: State-of-the-Art and Future Research Challenges.” In: *Computer Graphics Forum* 30.6 (2011), pp. 1719–1749 (cit. on p. 21).
- [MELS95] K. Misue, P. Eades, W. Lai, K. Sugiyama. “Layout adjustment and the mental map.” In: *Journal of Visual Languages & Computing* 6.2 (1995), pp. 183–210 (cit. on p. 41).
- [PHG07] H. Purchase, E. Hoggan, C. Görg. “How important is the “mental map”?—an empirical investigation of a dynamic graph layout algorithm.” In: *Graph drawing*. Springer, 2007, pp. 184–195 (cit. on p. 41).
- [Sil98] J. P. i Silvestre. “Approximation Heuristics and Benchmarkings for the MinLA Problem.” In: *Proceedings of Algorithms and Experiments (ALEX)*. 1998, pp. 112–128 (cit. on p. 34).
- [Spe02] R. Spence. “Rapid, Serial and Visual: a presentation technique with potential.” In: *Information Visualization* 1.1 (2002), pp. 13–19 (cit. on p. 28).
- [TMB02] B. Tversky, J. B. Morrison, M. Bétrancourt. “Animation: can it facilitate?” In: *International Journal of Human-Computer Studies* 57.4 (2002), pp. 247–262 (cit. on pp. 18, 22).
- [Tra17] U. S. D. of Transportation. *United States Department of Transportation*. [https://www.transtats.bts.gov/DL\\_SelectFields.asp?Table\\_ID=236&DB\\_Short\\_Name=On-Time](https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time). (Accessed on 07/23/2017). July 2017 (cit. on pp. 57, 67, 69, 72).
- [War04] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, 2004 (cit. on pp. 22, 28, 44, 47, 64).

- [WPZ+16] Y. Wu, N. Pitipornvivat, J. Zhao, S. Yang, G. Huang, H. Qu. “egoSlider: Visual Analysis of Egocentric Network Evolution.” In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (2016), pp. 260–269 (cit. on p. 29).

All links were last followed on July 31, 2017.

## **Declaration**

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature