Multimedia Tools and Applications manuscript No.
(will be inserted by the editor)

Volume-Based Large Dynamic Graph Analysis Supported by Evolution Provenance

Valentin Bruder[1]* · Houssem Ben Lahmar[1] · Marcel Hlawatsch[1] · Steffen Frey[1] · Michael Burch[2] · Daniel Weiskopf[1] · Melanie Herschel[1] · Thomas Ertl[1]

Received: 5 December 2018 / Revised: 18 April 2019 / Accepted: 10 June 2019

Abstract We present an approach for the visualization and interactive analysis of dynamic graphs that contain a large number of time steps. A specific focus is put on the support of analyzing temporal aspects in the data. Central to our approach is a static, volumetric representation of the dynamic graph based on the concept of space-time cubes that we create by stacking the adjacency matrices of all time steps. The use of GPU-accelerated volume rendering techniques allows us to render this representation interactively. We identified four classes of analytics methods as being important for the analysis of large and complex graph data, which we discuss in detail: data views, aggregation and filtering, comparison, and evolution provenance. Implementations of the respective methods are presented in an integrated application, enabling interactive exploration and analysis of large graphs. We demonstrate the applicability, usefulness, and scalability of our approach by presenting two examples for analyzing dynamic graphs. Furthermore, we let visualization experts evaluate our analytics approach.

Keywords Dynamic Graphs · Volume Rendering · Evolution Provenance · Visual Analytics

1 Introduction

Graphs can be used to analyze and visualize relational data in many application fields, such as network traffic, biological processes or social relationships.

Valentin Bruder
E-mail: valentin.bruder@visus.uni-stuttgart.de

[1] University of Stuttgart, Germany

[2] Eindhoven University of Technology, Netherlands

Constant data growth poses new challenges for the visualization of those data structures [28]. Visualizing large graphs that are dynamic, i.e., they contain an additional dimension with the data evolution over time, is even more challenging and an active field of research [8].

In this paper, we present a visual analytics approach that allows for interactive analysis of dynamic graphs containing several thousand time steps and hundreds of nodes. With our techniques and application, we support typical analysis tasks such as detecting temporal patterns, e.g., clusters forming and disintegrating or repeating occurrences of similar node-link structures. We present and implement four classes of analytics methods that we believe are central to the data analysis in this context:

1. Data views
2. Aggregation and filtering
3. Comparison
4. Evolution provenance

This paper is an extended version of work published in [13]. We extend our previous work to contain evolution provenance tracking and navigation of the visual analysis process and integrate it into our system for visualizing large dynamic graphs.

We center our techniques around a volumetric representation of the dynamic graph similar to a space time cube [4]. In this representation, every time step of the dynamic graph is represented by an adjacency matrix of the node link structure. Representing time in the third dimension, we arrange those adjacency matrices after one another to generate a three dimensional cuboid structure. This cuboid is a concise and static representation of the whole dynamic graph that preserves the mental map [30,3]. To allow for fluent user interactions (e.g., rotating and zooming into the cuboid or filtering values) we employ GPU-accelerated volume rendering methods to generate the graph data visualizations. Using those techniques, we are able to render several thousand time steps and hundreds of nodes at the same time without loosing interactivity.

The main contribution are methods to enable the four different classes of analytics methods for large dynamic graphs we identified. We complement the volumetric graph representation with additional 2D views (timeline, slice views), aggregation and filtering methods, and a technique to interactively compare multiple selected sections of dynamic graphs against each other. Another main contribution is the proposal of an evolution provenance model for the visual analytics process and the implementation thereof by automatically tracking important analysis steps. Those recorded changes are presented to the user in a plot that supports the navigation of the analysis process. Former analysis steps can be dynamically compared and recovered. We implement and combine all methods in an integrated application targeting expert users.

We show two application scenarios to demonstrate the usefulness of our combined methods. One is the analysis of a software call graph (982 nodes and $1,231$ time steps), the other one a dynamic graph of flight connection data over time (234 nodes and 16000 time steps). Additionally, we present the design

and results of an evaluation study with five visualization experts using the software call graph data set mentioned above.

2 Related Work

Dynamic graphs: Visualization and analysis of dynamic graphs is an active field of research. In recent years, several techniques have been proposed to deal with increasing sizes of the graphs in all dimensions [8]. Parallel edge splatting [16] is one of these techniques that relies on the node-link metaphor, but arranges data in a scalable way. This technique has been extended in various ways [9,14]. However, those techniques still suffer from visual clutter due to over-plotting of many edges. Other approaches deal with this issue by using adjacency matrices [21,15,41] or adjacency lists [25] as visual representation of the graph data. Small multiples is an approach showing time steps beside one another in the form of adjacency matrices [31,6]. The drawback of this visualization is the layout of time steps on a plane, requiring increasing screen space for larger data sets and making it difficult to compare arbitrary time steps or sequences with one another and detect patterns in the temporal domain.

Another approach on visualizing dynamic graphs is by stacking adjacency matrices into a 3D structure, thereby generating a space-time representation [5, 32]. The advantage of this visualization is the stability over time, leading to a preservation of the mental map [30,3]. Clusters become easily identifiable, especially if clustering or matrix reordering are applied to the data [10,26]. We base on those works and extend the space-time cube metaphor with volume rendering techniques to allow for the interactive analysis of larger data sets.

There is not much previous work on techniques or systems that can handle and visualize graphs with several thousand time steps. Burch et al. [14] use an overlapping version of parallel edge splatting to show graphs with more than thousand time steps. This approach was extended by Abdelaal et al. [1] with clustering techniques to handle even larger time series. Both approaches mainly employ some kind of data aggregation (our second class of analytics methods), additional data views (our first class of analytics methods) are not used. Similar to our approach, the work by Abdelaal et al. additionally includes a difference view for graph sequences, our third class of analytics methods. In the work by van den Elzen et al. [38], a projection of the temporal graph sequence to points is used to visualize large dynamic graph data. The presented tool allows then to select individual time steps and see the respective node-link visualization. The massive sequence views by van den Elzen et al. [37] uses a compact representation with overlapping lines to visualize dynamic graphs with many time steps. In both cases, multiple data views and aggregation techniques are employed—our first two classes of analytics methods.

All those works on visualizing large dynamic graphs use aggregated 2D representations of the data, which typically does not allow to see details of the graph structure. In contrast, we use a 3D representation of the full graph structure. Since our 3D representation has also drawbacks regarding

occlusion and projection, we augment it with additional data views, analysis and comparison features as well as evolution provenance. None of these previous works features a similar comprehensive analytics systems for large dynamic graphs as our proposed approach. Only a subset of our analytics classes were implemented by them.

Volume rendering: Using volume raycasting for rendering of the space-time representation has several advantages over rendering geometry using rasterization, although being computationally demanding. It offers high quality and flexibility in that each sample taken along all rays may be adjusted as desired. Volume rendering techniques have been applied in many visualization areas, e.g., for rendering simulation data from scientific experiments. Nowadays, GPU raycasting has become the de-facto standard for volume rendering in workstation environments [35, 23]. Beyond the typical application areas in scientific visualization, volume rendering has also been applied to various other types of data. In this context, an active field of research is the visualization of time series of 2D or 3D data. Frey et al. [20] developed an interactive visualization approach for 2D temporal data. They focus on detection of temporal patterns, e.g. constant regions or periodic changes. Woodring et al. [40] use set operators to combine multiple fields into single values and render the resulting volume. In another work, Woodring and Shen [39] propose Chronovolumes, a technique to combine a full timeseries of volumes into a single static volume. Also for time-varying volume data, Balabian et al. [7] propose a method that employs compositors and temporal transfer functions to highlight areas of high change in the data.

Evolution provenance: Historically, evolution provenance was mainly linked to scientific workflows where it was used to document the (development) process. Thereby, it automatically tracks the changes made between two versions that can be inputs, parameters, or functions invoked. Callahan et al proposed the first approach [17] that captures this kind of evolution provenance. The rational behind that was to accelerate the decision process about the correct module semantics since multiple workflow executions can be compared visually using the proposed tool called VisTrails. In the same context, Ellkvist et al. [19] propose a solution that facilitates collaboration between developers. It tracks changes made by all users in realtime during a collaborative design of the workflow and renders them in the form of branches. This particular type of provenance got known as "workflow provenance" or "provenance of the development process". We use the term "evolution provenance" recently proposed in [24] as it was shown that this particular type of provenance also encompasses other applications besides scientific workflows. Evolution provenance is increasingly used to improve visual data exploration systems [29, 11, 22]. Typically, data exploration is a time consuming process where users are engaged to analyze iteratively. Those works collect provenance information and use them to improve the data exploration process, e.g, offer recommendations that may help users to speed up the analysis process.

We distinguish two categories of visual data exploration systems featuring evolution provenance: (i) those collecting provenance in order to improve data exploration experience by proposing recommendations and (ii) those leveraging provenance to provide a story about what was discovered and how. Examples of existing works that fall in the first category include [29], where evolution provenance collected is used to compute recommendations. In the same context, authors in [12,11] leverage evolution provenance to compute visualization recommendations, thereby helping users in the design process. CLUE [22], a history based visual exploration system is part of the second category.

CLUE automatically captures the provenance of a visual exploration and provides users with the possibility to assemble states of interest into a story that can be used for presentation and recall. It fosters reproducibility of an analysis process by letting the user resume exploration from a specific state stored in the evolution provenance. In this paper, we adopt a similar approach to analyze large dynamic graphs. In what follows, we present the model of the provenance information collected by our approach. We discuss also an application example that showcases how collected provenance is exploited to increase the efficiency of analysis and to remember steps performed by users during experiments.

3 Static Volumetric Graph Representation

A focus of our approach for visual analysis of dynamic graphs are temporal aspects. In this context, a static representation of the data has several advantages compared to one using animations [36]. In this section, we give an overview of the volumetric graph data representation that is the center of our analysis application. In our approach, (directed) graphs are represented as adjacency matrices, i.e., the rows and columns denote the nodes of the graph. Therefore, an entry (i, j) at the i-th row and j-th column of the matrix denotes a (weighted) edge from the node with index i to the node with index j. One such matrix is generated for each time step of the dynamic graph. An adjacency matrix representation of a graph has several advantages [10]:

1. Large graphs can be depicted without crossing or intersecting graphical elements (as would be the case for traditional node-link diagrams).
2. We can generate a volumetric representation without the use of layout algorithms and naturally keep it consistent for all time steps.
3. Adjacency matrices are well suited for revealing cluster structures.

Inspired by space-time cube approaches in general [4] and the work of Bach et al. [5], we create a volumetric representation of the dynamic graph based on its adjacency matrices. These concepts have in common that they stack representations of individual time steps to gain a three dimensional data structure. As illustrated in Fig. 1, adjacency matrices are 2D structures that are stacked to incorporate the temporal evolution of the graph. In the resulting 3D structure, the x- and y-axes represent nodes, while entries in the plane

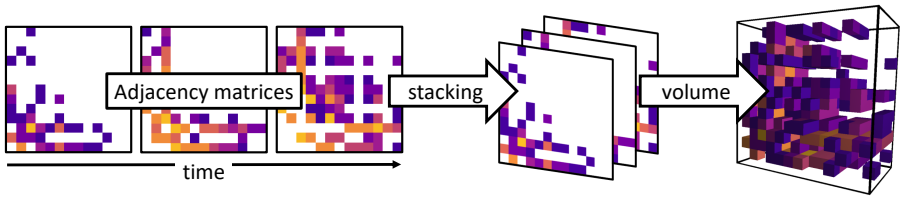


Fig. 1: For each time step of the dynamic graph, 2D adjacency matrices are stacked to create a 3D volume, forming our static graph representation.

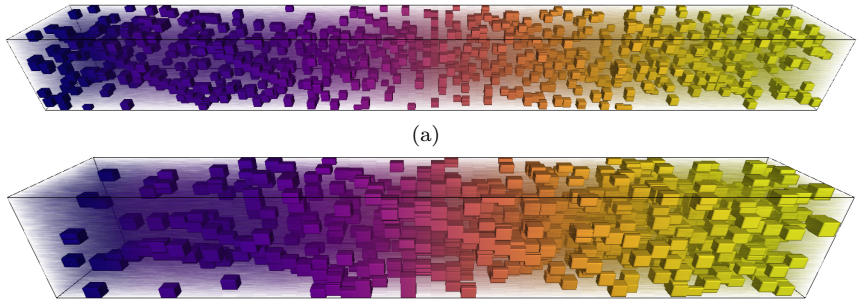
defined by those two axes represent edges (including their weights). The z-axis represents time.

The visual appearance of adjacency matrices strongly correlates with the ordering of the nodes, which can be an issue when analyzing the data: While a good ordering reveals clusters due to placing similar nodes next to each other, a bad one may result in noisy, irregular patterns with entries spread all over the matrix. Therefore, we support reordering of the nodes based on three different sparse matrix ordering algorithms: Cuthill-McKee [18], King [27], and Sloan [34]. We use the implementations of those provided by the boost graph library [33]. As basis for ordering, an arbitrary amount of time steps (i.e., matrices) is selected that are aggregated and used in the respective algorithm.

3.1 Scalability

The technical scalability of the dynamic graph size is limited by the available GPU memory. In our implementation, every edge including its corresponding weight is represented by an 8 bit scalar value. This results in a memory requirement of $n \times n \times t$ bytes, with n being the number of nodes and t the number of time steps. Additionally, we need approximately $\frac{1}{7}$ of this size to store a full 3D mipmap stack of the aggregated representations. For instance, using an NVIDIA Titan X (Pascal) GPU with 12GB VRAM, we can load a dynamic graph with 1,000 nodes and 10,000 time steps. It should be noted that the number of nodes affects the data size quadratically, while the number of time steps only has a linear impact. The number of edges has no influence on the memory requirements, i.e., a sparse and a dense graph with the same number of nodes and time steps requires the same size. Those size limitations could be mitigated in part by using compression, a different data structure, or out-of-core techniques to overcome GPU memory limitations. However, those extension remain subject to future work.

The visual scalability strongly depends on the used volumetric 3D representation. Here, the main issue is due to the occlusion of edges. Fig. 2 exemplifies this and allows an estimation of the scalability. The first example (Fig. 2a) demonstrates the scalability with respect to the temporal dimension of the graph. It shows a graph with 512 nodes, 10,000 time steps, a constant edge density of around 25,000 edges (around 10% of the fully connected graph).



(a)

(b)

Fig. 2: Scalability of the volumentric graph representation. (a) 10,000 time steps with constant cluster/edge density (8 clusters, around 25,000 edges). (b) 5,000 time steps with increasing cluster/edge density (2–24 clusters, around 20,000–51,000 edges). The graph has 512 nodes in both cases.

All time steps include 8 clusters, which are randomly redistributed every 50 time steps as well as some random edges. Fig. 2b shows the scalability with respect to the graph density. The shown graph features 512 nodes, 5,000 time steps, and the edge density increases from around 20,000 edges (8% of a fully connected graph) in the beginning to 51,000 edges (20% of a fully connected graph) in the last time steps. The number of clusters increases from 2 to 24, and the graph also contains random edges.

Comparing the two images, we can see that the scalability with respect to the temporal dimension is good with our approach. Doubling the number of time steps only slightly decreases the visibility of the data since the same image resolution is used for a larger volume. Furthermore, problems with a reduced resolution can be often alleviated with camera navigation, e.g., zooming and panning. However, looking at the example with increasing edge density, we can see that the visual scalability of our approach is worse in this case. Doubling the number of edges can introduce a lot of occlusion, which makes it difficult to analyze the data. To address this issue, we support various filtering and aggregation modalities, and complement our volumetric view with additional data views. This is described in detail in the next section.

4 Classes of Analytics Methods

With a single visualization technique, efficiently analyzing large dynamic graphs is typically not feasible due to the size and complexity of the data. We identified four important classes of methods that are central to analyzing large dynamic graphs, especially their temporal features (Fig. 3):

Data Views: Large dynamic graphs typically exhibit a lot of information in many interesting aspects. A single visualization is often not capable to

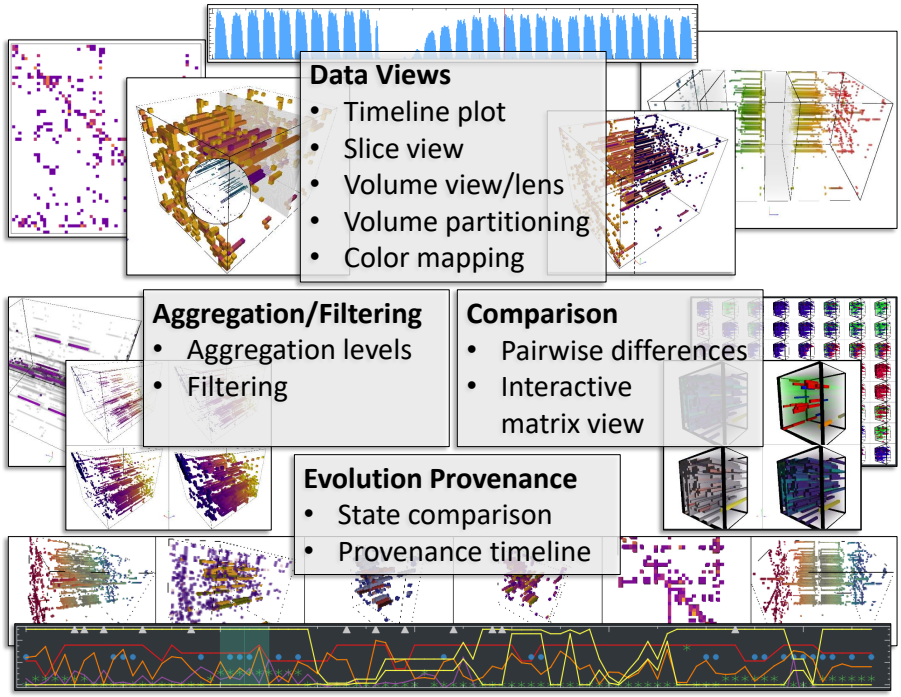


Fig. 3: Four classes of analytics methods for large dynamic graphs. We implement and discuss respective techniques, and show their utility.

convey them all. Therefore, it is important to offer distinct perspectives on the data using multiple linked views with suitable visualizations.

Aggregation and Filtering: Showing all edges of a large dynamic graph, especially a dense one, quickly leads to visual clutter, overload, and occlusion using our volumetric approach. This makes filtering and aggregation of adjacent edges an essential part of the analysis process to reduce the visualization to relevant information.

Comparison: Comparing different sections within a temporal graph or even several distinct dynamic graphs becomes challenging using the methods discussed above. Including a dedicated visualization for this specific task is therefore important for the analysis process.

Evolution provenance: Oftentimes during an analysis process, a parameter configuration is selected that shows interesting features of the data. Going back to such a analysis step can be very useful during data exploration. Furthermore, tracking and navigating the provenance evolution also helps to understand and reconstruct the analysis process. Our implementation of this class in described in Section 5.2.

We implement methods for all classes in our integrated graph analytics system (see Fig. 3). In the following, we describe these methods and their implemen-

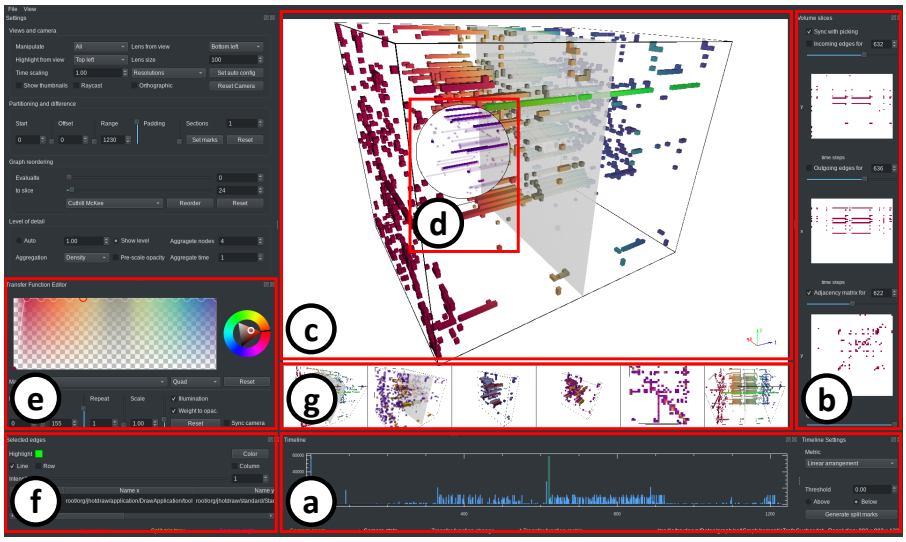


Fig. 4: The GUI of our dynamic graph analysis tool, consisting of (see Section 4.1): (a) timeline/provenance plot, (b) slice views, (c) volume view, (d) lens, (e) color map editor. The list view (f) shows information of selected edges and nodes, selected provenance states are shown as thumbnails (g).

tation in detail and discuss how they support the analysis of large dynamic graphs.

4.1 Data Views

A key to providing an analysis system that supports many different data sets and tasks is to offer multiple data views with different types of visualizations, because some are typically better suited for a specific task than others. We offer multiple views to the analyst that can reveal different aspects of the data, which are described in the following.

Timeline Plot: The timeline shows different graph metrics on a 2D plot over time (Fig. 4a). Those metrics include for instance, the number of edges in each time step, or the linear arrangement of the matrices. This visualization is intended to give an overview of the temporal evolution of the graph and thereby reveal recurring patterns. The timeline plot is also an interface for interaction tasks such as selecting and highlighting a single time step, browsing through the time slices or setting split marks (see Volume Partitioning below).

Slice View: While the timeline plot often provides a good overview on the data, it is highly aggregated and lacks details. Therefore, we also provide information of individual, selected time steps with 2D slice views (Fig. 4b).

We provide one view for each of the three dimensions, i.e. the slices show all incoming edges of one node (x-axis), all outgoing edges of one node (y-axis), or one adjacency matrix (time axis). The slices are aggregated (Section 4.2) and colored according to the current selection and may be linked to picking of elements and the section on the timeline plot. The respective slice planes can be visualized inside the volumetric view to give context in the spacial structure (cf. Fig. 4b).

Volume View: While the slice views provide a convenient method to analyze single nodes or time steps, it is hard to analyze the temporal evolution of structures on a global level. For this task, we provide a direct visualization of the full graph volume (Fig. 4c). Using this view, the analyst can interactively rotate, zoom, pan, and tilt the camera and thereby change the view on the data. Three viewing modes are supported for the volume view: a 2×2-tiled-view, a 1+3-view, and a view featuring the current and recent states (as depicted in Fig. 4). The 2×2 tiled view allows to compare different parametrization of the visualization (i.e., aggregation and filtering), by showing four representations of the data side by side that can be individually configured. This view can be changed to a $1 + 3$ view mode that shows a large view in the center and three thumbnails showing the three other configurations in the corners. The main view can be changed to any of the other configurations with a click on the respective thumbnail. To ease comparison, all camera parameters (rotation, zoom, etc.) are synchronized. The third possible view setup features a main view and a thumbnail band containing six consecutive provenance states (shown in Fig. 4g). The states to be shown can be selected via the evolution plot (see Section 5.2).

Detail Views: Besides the possibility to see four different configurations at the same time, we provide a lens that is controlled via mouse (Fig. 4d). This lens shows a different visualization parametrization inside the volume view, such as color mapping and/or aggregation level. We support interactive selection and highlighting of edges (single elements or aggregated blocks) inside the volume view. The selected edges are shown in a list view (Fig. 4f) in which corresponding nodes and weights are displayed as well.

Volume Partitioning: To improve the interaction possibilities with regard to temporal analysis of the graph, we support partitioning and splitting of the volume into sub-volumes along the time axis (see Fig. 5). The partitioning is helpful in reducing visual clutter and potentially occlusion. Split marks can dynamically be added and removed along the timeline plot, a visual gap between the sections in the volume indicates the locations. The gap size can be adjusted and single blocks selected for individual inspection. We also offer the possibility to automatically generate split marks based on different graph metrics and a threshold.

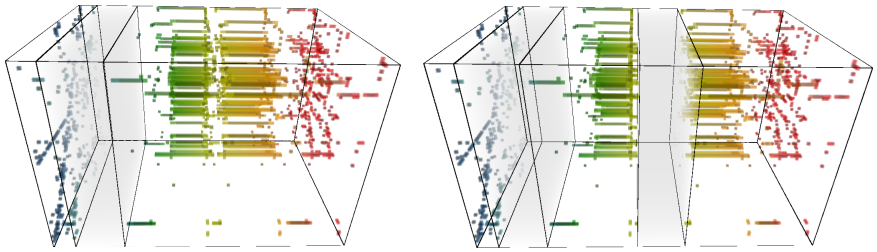
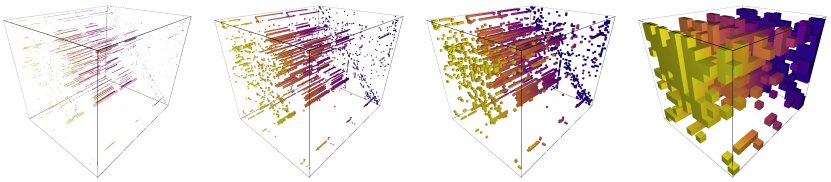


Fig. 5: Splitting the graph volume into sub-volumes along the time dimension enables dedicated analysis of specific sections.



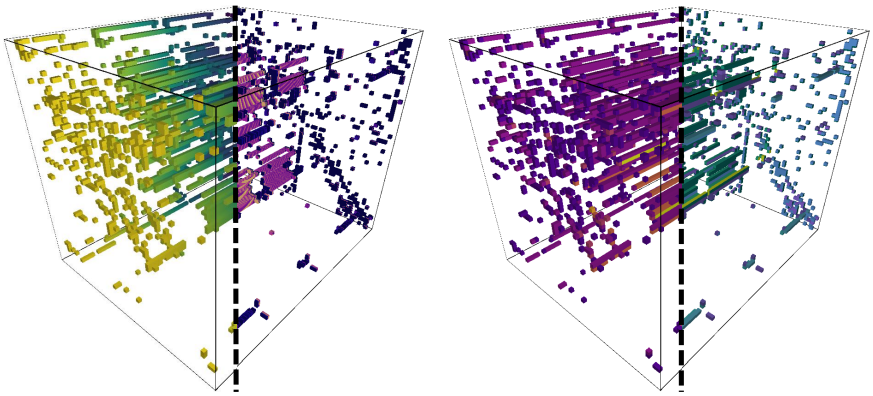
(a) Level 1 (b) Level 3 (c) Level 4 (d) Level 6

Fig. 6: Aggregation levels (aggregated by edge density) of a software call graph. An analyst may choose a level based on visibility and coarseness.

Color Mapping: Several properties of the dynamic graph may be mapped to color by using a color map editor (Fig. 4e). That means, the volumetric representation as well as the slice views are colored according to a selected metric and a user defined color map. Renderings for several possible color mappings are shown in Fig. 7). They include the weights of the edges and the order of the nodes in the adjacency matrix. Additional metrics support the analysis of the temporal evolution in the graph by mapping graph metrics, such as density or linear arrangement, as colors directly onto the volume. Further possible mappings include the temporal dimension and the lifetime of edges. We implement the color mapping for our static volumetric representation via the well-known concept of transfer functions from traditional volume rendering as it is used in scientific visualization. It bears some similarity to our filtering discussed in Section 4.2: color mapping defines the color, while filtering steers the opacity.

4.2 Aggregation and Filtering

Directly visualizing all edges of a large graph (e.g., containing millions of edges) is usually not feasible, especially for dense graphs. This is due to visual clutter, occlusion, and space constraints, depending on the visualization technique. In the case of our volumetric representation, the former two apply. To mitigate those problems, we offer several techniques for aggregation and filtering.



(a) time (left), edge count (right) (b) edge weight (left), edge lifetime (right)

Fig. 7: Four color mappings of dynamic graph properties: (a) time (left), edge count per time step (right); (b) edge weights (left), global edge lifetime (right).

Aggregation: In the volumetric and the slice views, individual edges (rendered as small cuboids/rectangles) can be very small, especially for larger graphs. Those small elements may be hardly visible or not at all, due to the projection and rasterization, which is limited by the screen resolution. Irritating visual artifacts, such as Moiré patterns may occur. To mitigate those issues, we support multiple data aggregation methods that can be applied to the original data and therefore generate a stack of representations with smaller resolutions. For aggregation methods, we support minimum, maximum, average weight, and density of edges that can all be applied either only in the spacial domain, i.e. aggregating neighboring edges, or in the time domain as well. Using this aggregation, one voxel in the volumetric view may represent multiple edges (see Fig. 6). Depending on the aggregation method used as well as the level, various features in the data may be revealed. For instance, coarse patterns can be seen in a high aggregation level, while details may be better analyzed in a low level and a high zoom into the volume. The aggregation level and method can be dynamically adjusted, it is applied to the volumetric as well as the slice views. Four different levels are presented initially in the main view to let the analyst select a suitable one.

Filtering: Filtering is a method essential to reducing visual clutter and occlusion, or highlight parts of the graph with specific properties. We support it by applying transparency to edges with selected properties, for instance those with low weights. Therefore, we use a transfer function mapping that can be dynamically adjusted and is applied to our volumetric representation.

4.3 Comparison

With our system, we focus especially on supporting the analysis of temporal aspects. Therefore, we support another modality for this task: the comparison of arbitrary time sequences (see Fig. 12). The analyst can select starting points and a range of time steps to compare them against one another. A matrix view is presented that shows the different time sequences on the main diagonal. We use the upper and lower triangles of the matrix to show different aspects: The upper one shows a comparison of the existence of edges, i.e. edges that appear, disappear, or exist in both sequences are highlighted by color and/or filtered by transparency. The lower triangle shows the difference in weights between the sections. The number of time sequences that are shown can be dynamically adjusted.

5 Evolution Provenance

With the variety of analytics methods presented so far, our approach offers users the possibility to perform different analysis operations and thereby obtain various results and observations. The collection of "history" of manipulations was shown to be effective for reproducibility as well as in remembering intermediate steps conducted to reach insights [24]. To this end, we have extended our approach to support the collection of evolution provenance. Traditionally, users are engaged in a visual analysis session where they perform various *visual analysis steps* iteratively. In this case, evolution provenance keeps track of the set of visual analysis steps performed and thereby comprises the "full story" of a visual analysis session.

5.1 Model and Definitions

Definition 1 (Visual analysis step) Given an initial dynamic graph G that contains a set of timesteps $\{t_1..t_k\}$, we define a visual analysis step as $S = \{G_s, V_s\}$ where a sub-graph G_s contains timesteps $\{t_i..t_j\}, 1 \leq i \leq j \leq k$ and is visualized using a visual analytics configurations V_s defined by the parameters $\{v_1..v_p\}$ (cf. Section 4).

In our current implementation, the evolution provenance collector tracks visual analysis steps encompassing the volumetric representation as our main view. For future work, we also plan to integrate tracking of slice views and the timeline plot. Table 1 summarizes supported analytics operations enabling the transition from a visual analysis step $S = \{G_s, V_s\}$ to another $S' = \{G'_s, V'_s\}$. It presents also the set of parameters needed for each operation. We distinguish six operation types that allow the application of different analytics methods presented in Section 4. These operations are captured with our evolution provenance model.

Table 1: Permitted analytics operations

Operation type	Parameters	Output
Selection	$\{t_{i'}, t_{j'}\}$; selected range of timesteps	$G'_s = \{t_{i'}..t_{j'}\}$, V_s
Partition	$\{m_1..m_y\}$; the set of split marks	G_s, $V'_s = \{v'_1..v'_p\}$
Aggregation	l, where l is a level	G_s, $V'_s = \{v'_1..v'_p\}$
Filtering	$d \times \alpha$, where α is opacity and d a graph property	G_s, $V'_s = \{v'_1..v'_p\}$
Color mapping	$d \times rgb$, where rgb is color and d a graph property	G_s, $V'_s = \{v'_1..v'_p\}$
Camera configuration	angles ϕ, θ, zoom ζ	G_s, $V'_s = \{v'_1..v'_p\}$

The *selection* operation is associated to the timeline plot feature (e.g., Figure 4.a) where the user gets a 2D-visualization whose x-axis depicts a set of timesteps $\{t_i..t_j\}$. Using this function, the user can select a single or a range of timesteps $\{t_{i'}..t_{j'}\}$ with $t_i \leq t_{i'} \leq t_{j'} \leq t_j$ to analyze them visually in the next step. The *partition* operation corresponds to the volume partitioning feature (e.g., Figure 5) where the user interactively specifies split marks $\{m_1..m_y\}$ to split the timesteps $\{t_i..t_j\}$ of a sub-graph G_s into sub-ranges $[\{t_i..t_{m_1}\}, \{t_{m_1}..t_{m_2}\}, \ldots, \{t_{m_y}..t_j\}]$.

Our evolution provenance model encompasses also the *aggregation* and the *filtering* operations. The former aggregates the data according to a level l to alleviate the complexity of the visualization, while the latter operation defines the visibility of parts of the data based on a factor α. Our provenance model also contain the *color mapping* operation where a user maps a graph property d (e.g., weights of edges) to color rgb. Finally, we record selected *camera configurations*, where the user selected a certain zoom level ζ and rotation angles ϕ, θ.

Overall, the evolution provenance is modelled by an analysis session graph that gathers all visual analysis steps made by the analyst. Figure 8 shows an example of such an analysis session graph (augmented with exemplary screenshots of the analytics step), defined as follows.

Definition 2 (Analysis session graph) An analysis session graph summarizes user's manipulations when exploring a dynamic graph data set. The analysis session graph is a labeled directed acyclic graph consisting of $n \in N$ nodes and $e \in E$ labeled edges. Each node n corresponds to a visual analytics step S. An edge $e = (n, n', L)$ represents the transition from one visual analytics step $S = \{G_s, V_s\}$ to the next visual analytics step $S' = \{G'_s, V'_s\}$. L is a pair $\langle op, param \rangle$ where op is an identifier of the analytical operation type (see Table 1) and $param$ is the set of parameters used to navigate from S to S'.

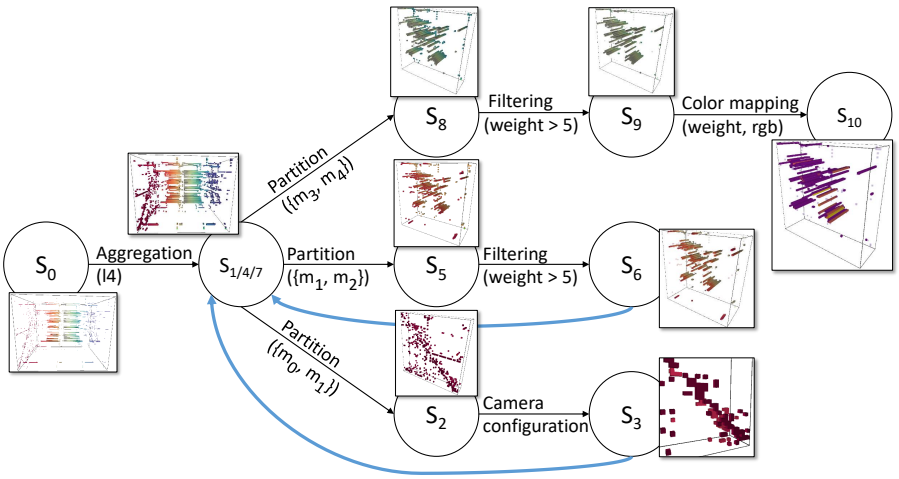


Fig. 8: Example of an analysis session graph, augmented with images of the respective analytics steps. Blue arrows indicate going back to a former state. Indices indicate the temporal order of the states S.

5.2 Visualization and Navigation

We implement the provenance model described in Section 5.1. Important analysis steps are saved automatically or on demand by the user: selection, aggregation, partitioning, camera configuration (on demand), color mapping and filtering via opacity. The evolution of the changes can be tracked and navigated in a visualization that shows important parameter changes in a combination of lines and glyphs.

With the number of recorded analytics steps, rendering the node-link diagram of the analysis session graph (see Fig. 8) can get quite complex and the visualization space demanding. Therefore, we chose a simplified representation of the provenance graph as a 2D plot. Fig. 9 shows a section of such a plot. The tracked analysis steps are plotted consecutively along the x-axis (ordered by time of configuration), while the value of the respective properties is plotted either binary (using a glyph) or by using a normalized y-value. In this plot example, the red line indicates changes to the aggregation level. The two yellow lines represent the sub-volume size (partitioning and selection) of the respective state by plotting the normalized minimum and maximum value of the visible time steps. For instance, values one and zero indicate that the analyst investigated the whole volume. The magenta and orange curves show the camera configuration at the respective state, while a triangle glyph indicates that the user saved this camera setup explicitly. A blue circle shows a change of the color mapping or filtering, a green star the selected graph property that is used for the color mapping in a categorical order.

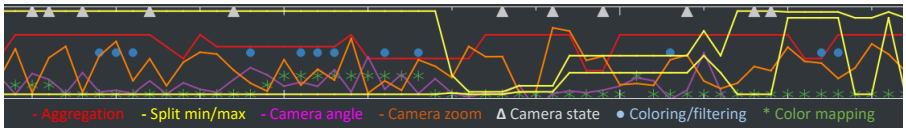


Fig. 9: Section of an example of our evolution provenance visualization, indicating different analysis steps during a session with plots and glyphs.

The user may interact with this plot to browse through the history of analysis steps, showing six consecutive ones as thumbnails below the volumetric view (see Fig. 4g). Those thumbnails can optionally be synchronized to the main camera configuration (i.e., they interact similar to the main view) to ease comparison tasks. All tracked states may be selected to apply their configuration to the main view, i.e., "going back" to this state.

6 Implementation

To enable interactive exploration of the volumetric data structure even of large dynamic graphs with thousands of time steps and hundreds of nodes, we accelerate the compute heavy calculations by using parallel processing on GPUs. Those calculations include the rendering of the volume and slice views, as well as the hierarchical generation of different aggregation levels. Some of the computations are also performed in parallel on the CPU such as reordering operations on the matrices. By using the OpenCL framework, we can accelerate data generation and processing across heterogeneous platforms such as GPUs, CPUs or parallel accelerators. When using GPUs, we can additionally take advantage of their integrated texture units and interpolation capabilities making them the preferred devices. The stacked adjacency matrices are then saved and processed as a 3D texture.

We support two different front-to-back raymarching algorithms that have been adapted for the needs of visualizing dynamic graphs. The first is a parallel 3D digital differential analyzer [2], featuring local illumination of voxel surfaces. Although having slightly less runtime performance than our second algorithm, it has several advantages: single and isolated voxels can be easily distinguished because of the lighting approach. The same applies to opaque structures, making this technique well suited for analyzing sparse graphs. The second rendering algorithm is based on a standard raycasting with equidistant sampling points along viewing rays through each pixel, as is typically used in GPU-accelerated volume rendering for scientific visualizations. In contrast to the first method, local illumination is performed based on the gradient that is calculated using central differences. This approach makes the technique better suited for the usage on dense graph data when making use of transparency for filtering.

In both techniques, we map sampled density values (i.e., edge weights) to opacity and color with our filtering respective color mapping features. Those mapped values are then composed into the final pixel color and opacity.

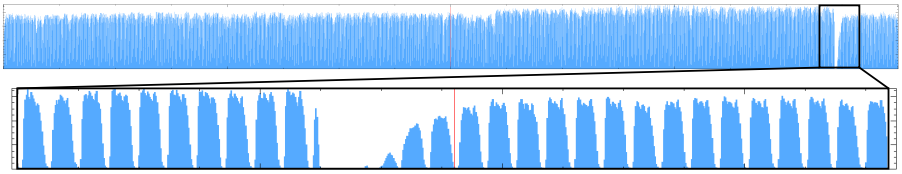


Fig. 10: Timeline plot of the flight data graph showing the number of edges per time step. The complete data set is shown on the top, while the bottom figure depicts data of September 2001 only.

Using raycasting instead of rendering geometry with rasterization has several advantages. Foremost, we do not need to order elements and can therefore easily apply transparency for filtering or highlighting purposes. This improves performance and enables interactive explorations with high frame rates even for large graphs with millions of edges. Furthermore, raycasting enables us to easily display multiple 3D views with synchronized camera configurations without a significant overhead.

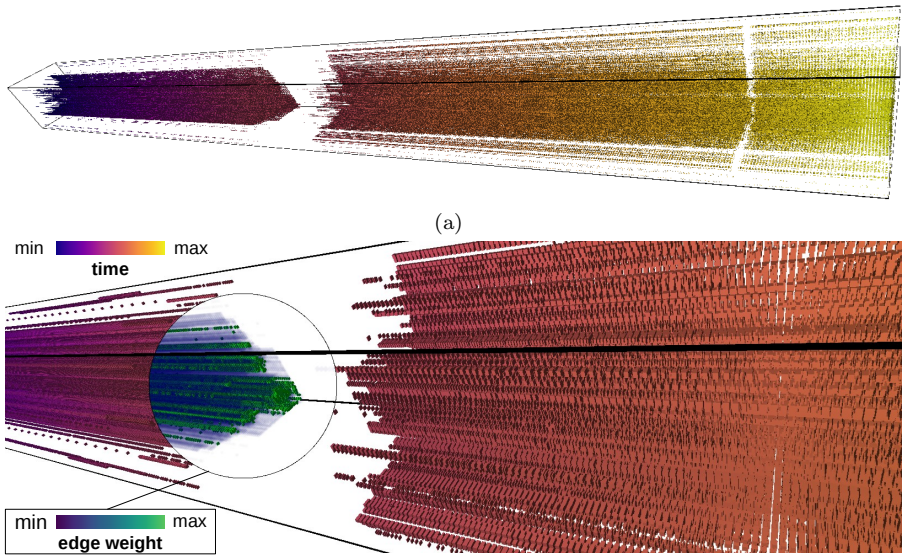
We tested our application on a workstation equipped with an NVIDIA Titan X (Pascal) GPU featuring 12GB of VRAM, an Intel Core i7-6700 and 16GB of RAM. Using a data set with over 1000 nodes and 8000 time steps, we were able to achieve interactive frame rates with a minimum of 20fps during exploration of the data set (i.e., camera interactions such as rotation and zoom). Due to the use of early ray termination, dense graphs can even be rendered faster depending on the opacity mapping.

7 Application Examples

We demonstrate our techniques with two application scenarios, focusing on the different classes of analytics methods we describe in Section 4. In the first one, a graph data set containing flight connections over time is analyzed (Section 7.1). The main focus lies on the comparison features of our application. In the second application scenario, a temporal software call graph is analyzed (Section 7.2). In this scenario, the main focus lies on using the evolution provenance tracking features for the analysis. The suitability of different classes of analytics methods is discussed in Section 7.3, also in the context the presented application examples.

7.1 Flight Connection Analysis Using Difference Views

Our first application example is a dynamic graph representing the domestic flights in the United States during the years 2000 and 2001. The graph contains 234 nodes (airports) and more than 16,000 discrete time steps (one per hour). An edge represents a connection between two airports, the weight indicates the frequency. The aggregated amount of edges in every time step is depicted



(a)

(b)

Fig. 11: (a) Volume representation of the full flight data, time is mapped to color. The volume is split between the two depicted years. (b) Close-up of the area between the years: the lens overlay shows edge weights mapped to color.

in the timeline plot (Fig. 10). There, a gap in September 2001 is immediately visible, which was caused by the 9/11 terrorist attacks and a shutdown of all air traffic in the hours following this incident. Furthermore, a regular pattern is immediately noticeable that repeats on a daily basis, visualizing the day-night changes.

Fig. 11a depicts the complete data in the volumetric representation in an overview, time is mapped to color. We applied the Cuthill-McKee graph ordering algorithm to change the node order based on the first 24 hours. To better see the temporal pattern and reduce visual artifacts, we selected aggregation level 2, i.e., 2^3 neighboring voxels are aggregated using the average weight values. To investigate the changes between the year 2000 and 2001, the volume is split at the according time step. That makes the increase of the edge count in the second year visible. The structure of the adjacency matrices shows that new edges appear especially at the boundaries. Closer investigation reveals that several airports do not have any flights in the year 2000, leading to the conclusion that they either opened, or were only added to the data base in 2001, which seems more likely.

Fig. 11b shows the usage of our lens functionality. We overlay the visualization with another visual representation showing a different color mapping that highlights edges with a higher weight (frequency of flights on the same connection). We can infer from this visualization, that higher frequencies especially

occur at larger airports. Those get clustered in the bottom right corner by our reordering algorithm, because of the many connections to other airports.

A direct comparison of time sequences is of special interest in this application example because of similar recurring patters. Using our difference view (Fig. 12), we can directly visualize differences between several consecutive days of the dynamic graph. We select eight days around the 9/11 attack to further investigate the disruption of air traffic. The difference matrix (8×8 grid view) gives an overview of the coarse patterns: canceled flights (blue) and resumed ones (red) are immediately visible. To investigate the differences of two consecutive days more closely (e.g., Sunday and Monday), we can enlarge/reduce the view to only show those two in a 2×2 tiled view. We can infer from the detailed view that many of the connections are stable over the two days (transparent green). However, there are more flights on Sunday morning compared to Monday morning. In contrast, there are more flights scheduled Monday evening than on Sunday evening.

7.2 Software Call Graph and Evolution Provenance Tracking

In the second example, we report the analysis of a dynamic graph representing software calls in a Java drawing application. We especially use this example to demonstrate the utility of provenance information that is captured during an analysis session and their navigation as supported by our application.

This data set has been previously used by Beck et al. [9] to demonstrate their dynamic graph visualization technique. It contains 982 nodes representing functions of the application on the code level, $32,259$ weighted and directed edges representing calls of these functions, and $1,231$ time steps that were recorded during the execution of the drawing application. Using a Java profiling tool, the following phases were captured: program start, creation of a new document, drawing a rectangle and a circle, and finally writing text into the circle. Weights of the edges represent execution times of the called functions. Nodes of the graph are ordered according to the hierarchy of functions in the software project. Therefore, we did not apply matrix reordering to keep the semantics of this hierarchy.

As described in Section 5.1, we capture analysis steps performed by the user and store them as provenance information. The evolution provenance is then visualized in our tool in the form of a plot to enable navigation (see Section 5.2). Fig. 13 depicts provenance information captured during the analysis of the call graph data set. Three examples for sequences of analysis steps and the related thumbnails (see Section 5.2) are shown in Fig. 14.

In the provenance plot, we can see that the camera zoom level and angle is permanently adapted during the whole duration of the analysis session. This is not surprising since a 3D representation of data usually requires a lot of camera changes to explore the data. Furthermore, the aggregation stays above a certain level during the whole analysis session. This means that a too fine grained representation of the data is not used during the exploration. One

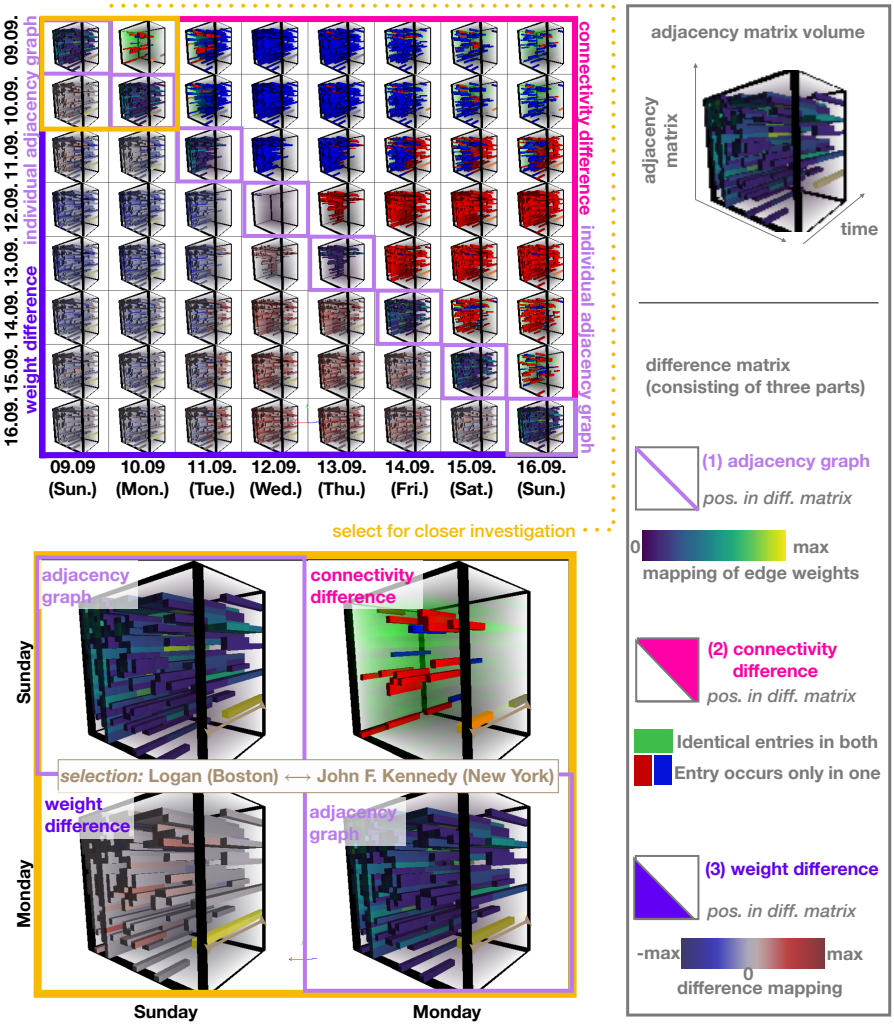


Fig. 12: *Left:* Difference visualization matrix to investigate changes in temporal sequences of dynamic graphs. This instance (cf. Section 7.1) depicts connections between US airports in September 2001 (diagonal), with differences between days in the upper and lower triangles. *Right:* Changes from Sunday to the following Monday (enlarged difference view). Connections are largely similar (indicated in transparent green), but there are less flights in the morning on Sunday and more in the afternoon; e.g., flights between Boston Logan and New York JFK (highlighted in yellow/orange).

reason for this might be the large size of the dynamic graph with around 1000 nodes and time steps leading to very small visual representations of the edges if no aggregation is used (see Section 4.2)

We partitioned the provenance plot based on the split parameter (yellow lines) that defines the sub-volume that is shown in the 3D graph visualization (see Fig. 5). From the beginning to the middle of the sequence, the complete graph volume is analyzed (Fig. 13a). The user initially adapted the aggregation level of the graph volume to his requirements, which is visible as changes in the red line plot. After she/he found a suitable setting, the user modified the transfer function. One purpose of this could be to highlight certain aspects of the data or filter out parts that are not of interest. To better understand these changes, we can look at the thumbnails for these parameter changes which provide a preview for the respective states. Fig. 14a shows the thumbnails for the first modifications of the color mapping and filtering. We can see that opacity for some of the edges is reduced to filter respectively emphasize edges with high weights (visible in orange).

In the next phase (Fig. 13b), the user applied the volume partitioning feature to explore a short time span at the beginning of the data set. The respective thumbnails and one example state are shown in Fig. 14b. In the plot, we can recognize that the aggregation level is increased in the beginning and after that, mainly the camera zoom level is adapted. We can also notice that two analysis steps were explicitly saved by the user indicating that she/he might have found something interesting and wanted to save the camera configuration. The thumbnail sequence reveals how the sub-volume is selected and that the camera is adapted to a more or less 2D view onto the graph volume similar to a classical adjacency matrix. This might be reasonable here because the graph volume captures only a few time steps and the interesting aspects of the data might lie in the connectivity information.

Next, a larger sub-volume almost in the middle of the time range of the data was selected for analysis. Looking at the plot (Fig. 13c), we can see a similar user behavior as for the previous phase: The aggregation level is changed first, followed by adaption of the camera configuration. The same aggregation level as in the previous phase is used. The next short phase marked in the plot (Fig. 13d) is interesting because we can see a short peak in the parameter defining the sub-volume selection (yellow lines). The sub-volume is changed to a later time span and then the previous time span is restored. This may indicate that the user visually compared these two time ranges, since other parameters besides the camera settings were not changed. The thumbnails confirm this (Fig. 14c): the two different sub-volumes are shown with the same visualization parameters. In the second to last of the marked phases (Fig. 13e), a shorter time range at the end of the data set is analyzed. We can trace changes in almost all parameters (color mapping, filtering, aggregation level, camera configuration) as well as how the user switches between selecting the short time range and the remaining time range of the data set.

In the last annotated phase (Fig. 13f), the user seems to switch between the different time spans she/he identified in the data set. Other parameters besides

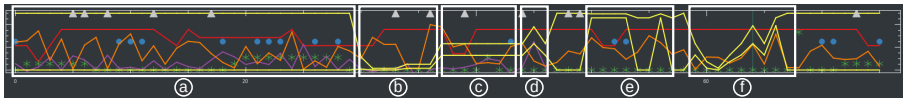


Fig. 13: Evolution provenance plot for an analysis session of the call graph data set. Different time spans of the analysis sessions are annotated.

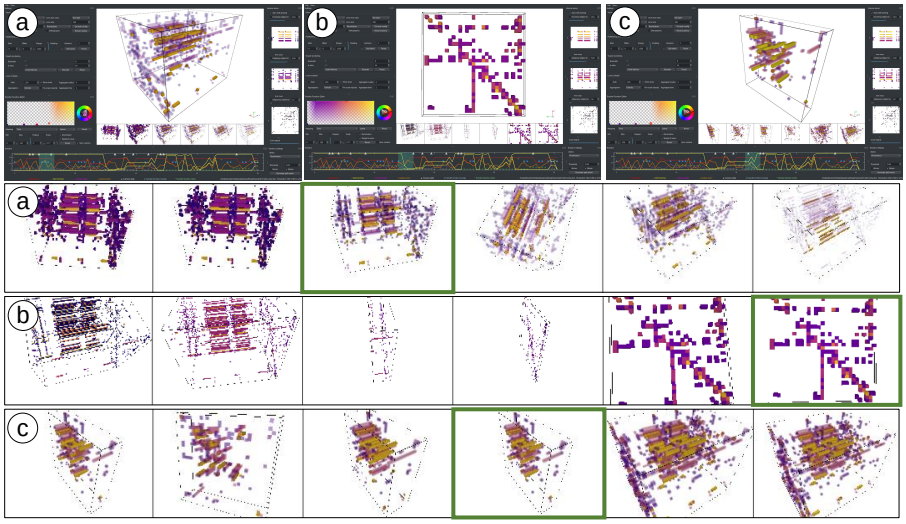


Fig. 14: Selected analysis steps (top row) and the respective thumbnail list (bottom three rows) for the exploration of the call graph data set. The respective analysis steps shown in the top row are highlighted with a green border in the thumbnail lists.

sub-volume ranges and camera parameters are not changed. An explanation for this can be that the user wants to recap what she/he has found in the data set in different time spans.

Based on the previously discussed provenance information, we now try to summarize how the call graph data set can be explored with our visualization approach. Furthermore, we also report information we have obtained about the call graph. Some of the features we mention in the following are not visible in the provenance plot because capturing the related analysis steps and parameters is not implemented yet. The illustration in Fig. 15 summarizes the different analysis steps during investigation of the call graph. The procedure is a typical top-down approach: starting at the overview, then splitting the graph into sections along the temporal domain, and finally investigating details and differences. Typically, a similar procedure should be applicable for different dynamic graphs using our system.

On the overview level showing the complete data set, an adequate aggregation level is selected for further analysis, which clearly shows the graph

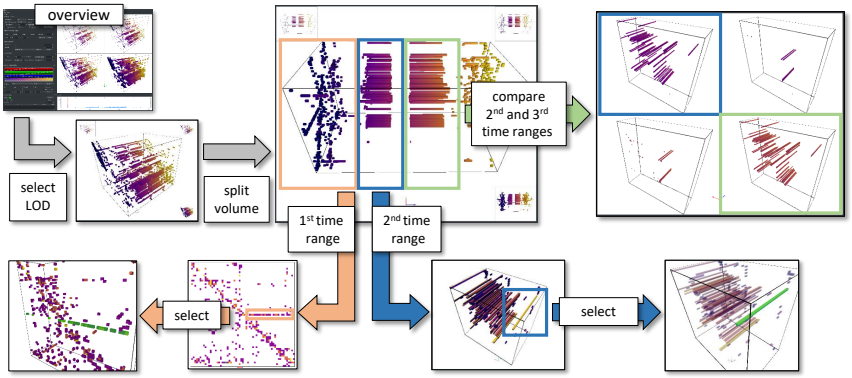


Fig. 15: Summary of the different steps during the analysis of the call graph data set with our visual analytics approach. First, an adequate aggregation level (LOD) is chosen for the temporal analysis of the graph. After that, the graph volume is split into different time ranges with prominent visual patterns, which are then analyzed in detail. Besides selecting and picking edges, the graph difference feature is used to compare two time ranges.

structure without hiding too many details. For this, the 2×2 tiled view is to used to get an overview on the different aggregation levels. On the overview level, different temporal phases in the data set are visible. Based on this finding, the graph volume is partitioned into sub-volumes that are then used for a detailed exploration of the individual time ranges. When we compare these different phases of the call graph, we can see that the first phase has more edges then the following phases but with a shorter time span of existence. The second and third phase exhibit similar structures and consists of many edges that exist for longer time spans. The last phase is more similar to the first phase and contains more edges that exist for shorter time ranges. Recapping the different phases of program execution that were captured for the call graph data set, we can draw the conclusion that starting and terminating the program requires more function calls that are executed in shorter time than the phases where the program handles user interaction, e.g., when a circle is drawn.

The first time range respectively sub-volume is selected for detailed exploration. Since there are many different edges visible, the graph volume representation might be less useful to analyze the graph structure. Therefore, we use the slice view (see Section 4.1) to see all edges that exist in the selected time span. This view reveals interesting features in the distribution of edges, e.g., many edges appear along the diagonal which means that many of the functions call neighboring functions with respect to the code hierarchy. Furthermore, we can see a function that calls many other functions (marked in Fig. 15) visible as a single row with many entries. We conclude that this function plays an important role in the execution of the software. By picking this row, we obtain the following information: a function “createTools” calls different “init”

functions. We assume that this function is responsible for creating the drawing tool bar. Using the volumetric representation, we can easily determine that these function calls are only executed in the first time step.

Selecting the next phase of the graph data set, we can see in the volumetric representation that there are less edges, i.e., less function calls, with most of them existing for a comparably long time span. By using the global edge life time for the color mapping, we identify which of the edges exist longer than others. With the transfer function in our example, edges with yellow color exist for the longest time span. To understand why these edges exist longer than others, i.e., the respective functions are called quite often, we can select them by picking and use the list view to obtain more information about them. In this case, a function that is responsible for redrawing the background of the application was selected. It is no surprise that such a function is called many times during the execution of the program.

Looking at the third phase, we already recognized that it looks very similar to the second phase. The same edges seem to appear for a slightly longer time range. Since a visual comparison is not accurate due to the aggregation and the volumetric representation, we use the graph difference visualization (see Section 4.3) to check the similarities of these two phases in detail. As we can see in this visualization, almost the same edges occur in both time spans with the same weights, i.e., the same execution times. Only a few edges around the center area are different. Picking these edges and using the list view reveals that function calls to "RectangleFigure/displayBox" were replaced by calls to "EllipseFigure/displayBox". Hence, almost the same functions are used when a rectangle or an ellipse is drawn in the application.

7.3 Discussion of Analytics Classes

In the following, we discuss our proposed four classes of analytics methods regarding their advantages and limitations in the context of the two presented application examples. We provide a more general discussion including advantages and disadvantages of the different classes in Section 4. The flight data set is characterized by considerably more time steps than nodes, a high density, and repetitive patterns. In contrast, the call graph features a similar number of nodes and time steps, is rather sparse, and has different visual patterns. Therefore, our analysis focuses on distinct aspects and uses the analysis classes differently.

The advantage of using our different data views becomes immediately noticeable in the flight data application scenario. There, the 2D timeline plot gives a good overview on the temporal structure, but lacks details. The patterns of the edges are much better visible in the volumetric view. A drawback of this view is the high amount of occlusion that can be mitigated with the help of filtering, aggregation, and the partitioning/splitting of the volume. Making details visible on demand is possible with the interactive lens feature, the slice views, and picking/highlighting of specific edges or nodes. Comparing

sequences of the dynamic graph with one another would still be a tedious task using those data views. Here, our graph difference visualization can reveal recurring pattern and differences directly for arbitrary time sequences. In case of the flight data, for instance, the comparison view shows the connections that change between days and those that are constant.

In the case of the second application example regarding the call graph data set, the volume partitioning method is particularly effective, because the data has four clearly distinguishable temporal sections. Selecting those regions for closer inspection allows filtering of uninteresting parts. Due to the large size in all dimensions of this particular data set, aggregation is key for highlighting the coarse patterns in the graph. The slice view helps in investigating single time steps and picking individual edges, in this application example the specific method calls are recorded as weighted edges. Color mapping can be used to highlight methods that are called most often. Our graph difference visualization helps in efficiently revealing fine differences between similar looking time spans in the data. Using evolution provenance, the progress and results of the visual exploration process may be recapitulated and better understood, which can be crucial for reproducibility. During exploration, the possibility to visualize former analysis steps, compare them against the current setup, and easily toggle between them helps to ease the analysis process.

The importance of supporting all four classes of analytics methods we determined (cf. Section 4), is demonstrated with our two application examples. We mitigate the respective disadvantages of the various methods by combining and linking them. The timeline plot, the slice views, and the volumetric representation all have their own benefits and limitations by visualizing different levels, sections, or abstractions of the data. However, with growing size of dynamic graphs, data occlusion and visual clutter make it increasingly hard to find patterns even when using the different data views. Therefore, suitable aggregation and filtering methods become important. Furthermore, partitioning and splitting along the temporal domain helps in separating interesting parts for closer inspection. The comparison of temporal sequences and similar recurring patters is a complex task that we directly support with an interactive graph difference visualization. All these different types of operations and parameters make it difficult to track and recapitulate the analysis process. We support this by capturing and visualizing evolution provenance and thereby allow to navigate to significant previously performed analysis steps.

8 Expert Evaluation

Our two use cases show how trained users can successfully use our techniques and system to analyze large dynamic graphs. To gather qualitative feedback on how untrained visualization researchers can work with our techniques and system, we conducted a think-aloud user study with visualization experts. We asked five visualization researchers from our institute, three having mainly an information visualization and two a scientific visualization background, to solve

several data analysis tasks. While all of them are familiar with the general concepts of graphs and their visualization, most of them have other research topics.

We chose this type of evaluation because a fair comparison of our system to other techniques for large dynamic graph visualization is difficult since we propose an integrated approach that features different data views, comparison, as well as aggregation and filtering besides the volumetric view (see Section 4). A comparable system would need to include a similar feature set (difference views, aggregation, filtering, etc.). However, our components were designed to complement the 3D volumetric representation. Therefore they are not directly portable to other techniques or less useful in combination with them. We have refrained from performing a quantitative study with time and error measurement because of two reasons. First, statistical meaningful results need a larger amount of participants, in our case visualization experts. Second, it is difficult to find specific measurable tasks that can meaningfully represent the whole analytics process with our system.

8.1 Study Design

We started the study with a structured demonstration of the functionality and how to use the features of the system, as well as for a short learning period. We used the flight data set (Section 7.1) for this purpose. The software call graph that we discuss in Section 7.2, was then used as the data set for several analysis tasks. To begin with, the participants were given a short introduction to the data set, including the program type, the meaning of the weights and a high-level overview on the sequence (program start, user interaction, program close). However, no further details about the program execution and user interactions were provided. The study took approximately one hour on average, including the demonstration (about ten minutes) and the learning period (about five minutes). While the experts were working on the tasks, we recorded the screen and spoken comments of the participants. Furthermore, we tracked the mouse position to determine which view or widget was being used, and saved the provenance traces that were automatically generated by our system. We finally asked the participants to briefly summarize their results in a questionnaire. We asked the experts to solve six tasks:

1. Select a suitable aggregation level to get a good overview of the entire dynamic software call graph.
2. Identify time spans with different behavior.
3. Partition the graph into those time spans and investigate them.
4. Identify two equally looking time ranges and validate whether they are identical or slightly different.
5. Describe a given provenance trace.
6. Compare their own workflow against the one shown by the provenance trace.

Table 2: Answers to the question how useful the respective component was, from 1 (not helpful) to 5 (very helpful).

Component	Mean	Standard deviation
Timeline view	4.4	±0.8
Volume view	4.8	±0.4
Slice views	3.0	±1.1
Partitioning	4.2	±0.8
Difference view	4.8	±0.4
Provenance graph and views	3.6	±1.4
Overall system	4.4	±0.5

For solving these tasks, the participants were free to use any of the available views and features of the system. We chose quite specific tasks because they had a higher chance of providing comparable results with our small group of participants. Furthermore, we wanted to test if our proposed workflow depicted in Fig. 15 is suitable for visualization experts that are not familiar with the data set to get insights into the dynamic graph.

8.2 Results

We asked the experts to rate the visualization components of our system on a Likert scale from 1 (not helpful) to 5 (very helpful) with the option to give no rating. The participants could also write free-text comments and suggestions for improving the system and the visualizations. Table 2 shows the average ratings of the various components of our system. The overall system was rated very helpful, with the volume view as well as the difference view turning out to be the most helpful components for the given tasks.

Table 3 shows the results of our mouse tracking during the user study. We captured the mouse positions in half-second intervals and increased a counter for the respective widget below. The relative usage was calculated by normalizing those counters at the end of the session. As can be seen, the volume view was used the most for the tasks. The timeline view was the second most used widget that was also rated (very) helpful by most participants. In terms of usage, adjusting the settings such as the aggregation level follows next, while the slice views and the transfer function editor were used less or not at all by the experts . Only one of the participants used the transfer function editor at all (for filtering low weights). For improving this widget, one of the experts suggested easily accessible, pre-defined transfer functions and showing the editor only on demand.

Note that the user rating (Table 2) and the widget usage (Table 3) are influenced by the fact that the volumetric graph representation is the central and most prominent component of our system, whereas the additional components and features were designed to support and complement this technique. This

Table 3: Relative usage of the different widgets during the tasks (based on mouse tracking).

Widgets	Average usage (%)	Standard deviation (%)
Timeline view	16.6	±5.1
Volume view	58.5	±3.6
Slice views	7.4	±6.1
Transfer function editor	2.1	±3.6
Settings	15.0	±6.7

fact probably contributes to the high relative usage of the volume view (almost 60% on average) as well as the positive rating of the component (4.8 ±0.4).

We could derive additional suggestions for improving visual representations an usability from the free-text comments. This includes tool-tips for different terminology and sharp borders between the slice views to separate them better. Furthermore, one expert suggested improving the slice views with a separate aggregation levels and a zoom functionality to improve their helpfulness.

Regarding task 1, the participants selected aggregation levels between 2 to 4 because they yield desirable properties: "visual appeal, not too much clutter, still visible details". Several experts adjusted the aggregation level during the completion of the other tasks, as exemplified in the provenance graph (Fig. 16). This confirms the utility of our aggregation feature and its real-time adjustability. In tasks 2 and 3, the experts' selections generally match our own partitioning of the graph as suggested in Fig. 15. Overall, all experts could identify and interpret the coarse structure of the graph using our system. The main differences to our analysis in Section 7.2 were that two experts created a finer partitioning and one a coarser by combining the two sections in the middle. Furthermore, the participants could identify several details and assumptions were made about the characteristics of the program flow, although the participants were only given a high level description of the data set. For instance, several users pointed out the initialization and de-initialization phases and they were able to identify user interactions.

Regarding task 4, which required the experts to compare two equally looking sections, three of the five participants compared parts similar to the 2^{nd} and 3^{rd} time range in Fig. 15. One validated the periodicity within the 2^{nd} time range, while another one compared the sequences right before and after those two blocks. According to their feedback, the difference feature proved very helpful for this task. In the last two tasks, all experts were able to comprehend the example workflow we provided to them and spot differences to their own provenance trace. This included differences in temporal partitioning of the graph data, aggregation levels, and the use of the transfer function for filtering.

Fig. 16 shows a sample provenance graph that was recorded for one of the participants. Identifiable is the change of the aggregation level (red curve) in the beginning as asked in the first task. Afterwards follows a detailed investigation of seven identified sections of the graph as indicated by the yellow curves (task

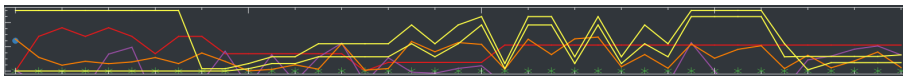


Fig. 16: Provenance graph of one of the experts that was generated during the study.

2 and 3), concluding with the investigation regarding the differences (task 4). Notably, this expert did not change the transfer function or the mapping metric during the whole time and adjusted the aggregation level several times during the analysis task. Compared to our example analysis of the same data set (Section 7.2), the participant made a more fine-grained separation of sections in the temporal domain. However, the overall separation matches ours.

Overall, visualization experts normally not working with graph data were able to successfully analyze an unknown dynamic graph data set using our system. The experts came to similar results in their analysis and were able to detect major structures in the data. While we could not directly quantify the effectiveness of our approach, the expert feedback indicates its utility for the analysis of large dynamic graphs.

9 Conclusion and Future Work

We presented an approach for the interactive visual analysis of large dynamic graphs with several thousand time steps. We base our approach on four classes of analytics methods that we consider to be central to the analysis of large temporal graph data. These four classes are data views, aggregation and filtering, comparison, and evolution provenance. We integrated methods for all classes into an interactive visual analytics system that we used to demonstrate the applicability and usefulness of our approach. We presented two use cases, one analyzing flight connections over two years and a second one analyzing a software call graph. Our expert evaluation showed that our system is well suited to get insight into dynamic graphs even for visualization experts that normally do not work with this kind of data.

Central to our analysis approach is a volumetric view of the dynamic graph based on the concept of space-time cubes. The implementation is GPU-accelerated enabling interactive exploration of large data sets. Additional views show distinct data visualizations and projections. The color and opacity mapping can be adjusted to show and hide different graph properties, while aggregation helps in identifying coarse structures. A comparison feature for temporal sequences as well as providing evolution provenance complements the data views. We base the latter on an provenance model specifically designed for the case of visually analyzing large dynamic graphs. We demonstrate the scalability of our approach to dynamic graphs with several thousand time steps with our application examples, expert evaluation, and synthetic data sets.

For future work, we want to focus on two aspects. First, scalability to even larger data sets, which is currently limited by the available GPU memory. The usage of out-of-core techniques can mitigate this limitation. Second, we want to further extend our integrated support for evolution provenance, which turned out to be very useful during data exploration. Here, we see several possibilities for improvements and extensions: We will add more features for handling the provenance information by the user, e.g., user comments or bookmarks. There is also room for improvement regarding the visualization of the analysis steps, for instance, adding more detailed representations for color mapping changes and filtering that show the actual differences between steps. Finally, we want to use a heuristic to record interesting camera states automatically and also suggest new parameter setups based on evaluating the previous captured evolution provenance.

Acknowledgements We would like to thank the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) for support within Projects A02, B01, and D03 of SFB/Transregio 161 (project number 251654672).

References

1. Abdelaal, M., Hlawatsch, M., Burch, M., Weiskopf, D.: Clustering for Stacked Edge Splatting. In: F. Beck, C. Dachsbacher, F. Sadlo (eds.) Vision, Modeling and Visualization. The Eurographics Association (2018). DOI 10.2312/vmv.20181262
2. Amanatides, J., Woo, A., et al.: A fast voxel traversal algorithm for ray tracing. In: Eurographics, no. 3 in 87, pp. 3–10 (1987)
3. Archambault, D., Purchase, H.C., Pinaud, B.: Animation, small multiples, and the effect of mental map preservation in dynamic graphs. IEEE Transactions on Visualization and Computer Graphics **17**(4), 539–552 (2011)
4. Bach, B., Dragicevic, P., Archambault, D., Hurter, C., Carpendale, S.: A descriptive framework for temporal data visualizations based on generalized space-time cubes. In: Computer Graphics Forum, vol. 36, pp. 36–61. Wiley Online Library (2017)
5. Bach, B., Pietriga, E., Fekete, J.D.: Visualizing dynamic networks with matrix cubes. In: CHI Conference on Human Factors in Computing Systems, pp. 877–886 (2014)
6. Bach, B., Riche, N.H., Dwyer, T., Madhyastha, T.M., Fekete, J., Grabowski, T.J.: Small multipiles: Piling time to explore temporal patterns in dynamic networks. Computer Graphics Forum **34**(3), 31–40 (2015)
7. Balabanian, J.P., Viola, I., Möller, T., Gröller, E.: Temporal styles for time-varying volume data. In: S. Gumhold, J. Kosecka, O. Staadt (eds.) Proceedings of 3DPVT'08 - the Fourth International Symposium on 3D Data Processing, Visualization and Transmission, pp. 81–89 (2008)
8. Beck, F., Burch, M., Diehl, S., Weiskopf, D.: A taxonomy and survey of dynamic graph visualization. Computer Graphics Forum **36**(1), 133–159 (2017)
9. Beck, F., Burch, M., Vehlow, C., Diehl, S., Weiskopf, D.: Rapid serial visual presentation in dynamic graph visualization. In: 2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 185–192 (2012)
10. Behrisch, M., Bach, B., Riche, N.H., Schreck, T., Fekete, J.: Matrix reordering methods for table and network visualization. Computer Graphics Forum **35**(3), 693–716 (2016)
11. Ben Lahmar, H., Herschel, M.: Provenance-based recommendations for visual data exploration. In: Workshop on Theory and Practice of Provenance (TaPP) (2017)
12. Ben Lahmar, H., Herschel, M., Blumenschein, M., Keim, D.A.: Provenance-based visual data exploration with evlin. In: Conference on Extending Database Technology (EDBT), pp. 686–689 (2018)

13. Bruder, V., Hlawatsch, M., Frey, S., Burch, M., Weiskopf, D., Ertl, T.: Volume-based large dynamic graph analytics. In: Proceedings of the 22nd International Conference on Information Visualization, IV, pp. 210–219 (2018)
14. Burch, M., Hlawatsch, M., Weiskopf, D.: Visualizing a sequence of a thousand graphs (or even more). Computer Graphics Forum **36**(3), 261–271 (2017)
15. Burch, M., Schmidt, B., Weiskopf, D.: A matrix-based visualization for exploring dynamic compound digraphs. In: Proceedings of the 17th International Conference on Information Visualisation, IV, pp. 66–73 (2013)
16. Burch, M., Vehlow, C., Beck, F., Diehl, S., Weiskopf, D.: Parallel edge splatting for scalable dynamic graph visualization. IEEE Transactions on Visualization and Computer Graphics **17**(12), 2344–2353 (2011)
17. Callahan, S.P., Freire, J., Santos, E., Scheidegger, C.E., Vo, T., Silva, H.T.: VisTrails : Visualization meets Data Management. In: SIGMOD (2006)
18. Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the 1969 24th national conference, pp. 157–172. ACM (1969)
19. Ellkvist, T., Koop, D., Anderson, E.W., Freire, J., Silva, C.T.: Using provenance to support real-time collaborative design of workflows. pp. 266–279 (2008)
20. Frey, S., Sadlo, F., Ertl, T.: Visualization of temporal similarity in field data. IEEE Vis. Comput. Gr. **18**, 2023–2032 (2012)
21. Ghoniem, M., Fekete, J., Castagliola, P.: On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis. Information Visualization **4**(2), 114–135 (2005)
22. Gratzl, S., Lex, A., Gehlenborg, N., Cosgrove, N., Streit, M.: From visual exploration to storytelling and back again. Computer Graphics Forum (EuroVis '16) **35**(3), 491–500 (2016)
23. Hadwiger, M., Ljung, P., Salama, C.R., Ropinski, T.: Advanced illumination techniques for gpu volume raycasting. In: ACM SIGGRAPH ASIA 2008 Courses, SIGGRAPH Asia '08, pp. 1:1–1:166. ACM, New York, NY, USA (2008)
24. Herschel, M., Diestelkämper, R., Ben Lahmar, H.: A survey on provenance: What for? what form? what from? VLDB Journal **26**(6), 881–906 (2017)
25. Hlawatsch, M., Burch, M., Weiskopf, D.: Visual adjacency lists for dynamic graphs. IEEE Transactions on Visualization and Computer Graphics **20**(11), 1590–1603 (2014)
26. Kaufman, L., Rousseeuw, P.J.: Finding groups in data: an introduction to cluster analysis, vol. 344. John Wiley & Sons (2009)
27. King, I.P.: An automatic reordering scheme for simultaneous equations derived from network systems. International Journal for Numerical Methods in Engineering **2**(4), 523–533 (1970)
28. von Landesberger, T., Kuijper, A., Schreck, T., Kohlhammer, J., van Wijk, J.J., Fekete, J., Fellner, D.W.: Visual analysis of large graphs: State-of-the-art and future research challenges. Computer Graphics Forum **30**(6), 1719–1749 (2011)
29. Milo, T., Somech, A.: React: Context-sensitive recommendations for data analysis. In: ACM SIG Conference on the Management of Data (SIGMOD), pp. 2137–2140 (2016)
30. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. Journal of Visual Languages and Computing **6**(2), 183–210 (1995)
31. Perer, A., Sun, J.: Matrixflow: temporal network visual analytics to track symptom evolution during disease progression. In: AMIA annual symposium proceedings, vol. 2012, p. 716. American Medical Informatics Association (2012)
32. Schneider, T., Tymchuk, Y., Salgado, R., Bergel, A.: Cuboidmatrix: Exploring dynamic structural connections in software components using space-time cube. In: Software Visualization (VISSOFT), 2016 IEEE Working Conference on, pp. 116–125. IEEE (2016)
33. Siek, J.G., Lee, L.Q., Lumsdaine, A.: The Boost Graph Library: User Guide and Reference Manual, Portable Documents. Pearson Education (2001)
34. Sloan, S.: An algorithm for profile and wavefront reduction of sparse matrices. International Journal for Numerical Methods in Engineering **23**(2), 239–251 (1986)
35. Stegmaier, S., Strengert, M., Klein, T., Ertl, T.: A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In: Proceedings of the Fourth Eurographics / IEEE VGTC Conference on Volume Graphics, VG'05, pp. 187–195. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2005)

36. Tversky, B., Morrison, J.B., Betrancourt, M.: Animation: can it facilitate? International Journal of Human-Computer Studies **57**(4), 247–262 (2002)
37. van den Elzen, S., Holten, D., Blaas, J., van Wijk, J.J.: Dynamic network visualization withextended massive sequence views. IEEE Transactions on Visualization and Computer Graphics **20**(8), 1087–1099 (2014)
38. van den Elzen, S., Holten, D., Blaas, J., van Wijk, J.J.: Reducing snapshots to points: A visual analytics approach to dynamic network exploration. IEEE Transactions on Visualization and Computer Graphics **22**(1), 1–10 (2016)
39. Woodring, J., Shen, H.W.: Chronovolumes: a direct rendering technique for visualizing time-varying data. In: Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume Graphics, VG '03, pp. 27–34. ACM, New York, NY, USA (2003)
40. Woodring, J., Shen, H.W.: Multi-variate, time varying, and comparative visualization with contextual cues. IEEE transactions on visualization and computer graphics **12**(5), 909–916 (2006)
41. Yi, J.S., Elmqvist, N., Lee, S.: Timematrix: Analyzing temporal social networks using interactive matrix-based visualizations. International Journal of Human-Computer Interaction **26**(11&12), 1031–1051 (2010)