EPJ Data Science
a SpringerOpen Journal

RESEARCH   **Open Access**

Fast filtering and animation of large dynamic networks

Przemyslaw A Grabowicz[1,2*], Luca Maria Aiello[3] and Filippo Menczer[4]

*Correspondence:
pms@mpi-sws.org
[1] Max Planck Institute for Software
Systems, Saarland University,
Saarbrucken, Germany
[2] Institute for Cross-Disciplinary
Physics and Complex Systems,
University of Balearic Islands, Palma
de Mallorca, Spain
Full list of author information is
available at the end of the article

Abstract

Detecting and visualizing what are the most relevant changes in an evolving network is an open challenge in several domains. We present a fast algorithm that filters subsets of the strongest nodes and edges representing an evolving weighted graph and visualize it by either creating a movie, or by streaming it to an interactive network visualization tool. The algorithm is an approximation of exponential sliding time-window that scales linearly with the number of interactions. We compare the algorithm against rectangular and exponential sliding time-window methods. Our network filtering algorithm: (i) captures persistent trends in the structure of dynamic weighted networks, (ii) smoothens transitions between the snapshots of dynamic network, and (iii) uses limited memory and processor time. The algorithm is publicly available as open-source software.

1 Introduction

Network visualization is widely adopted to make sense of, and gain insight from, complex and large interaction data. These visualizations are typically static, and incapable to deal with quickly changing networks. Dynamic graphs, where nodes and edges churn and change over time, can be effective means of visualizing evolving networked systems such as social media, similarity graphs, or interaction networks between real world entities. The recent availability of live data streams from online social media motivated the development of interfaces to process and visualize evolving graphs. Dynamic visualization is supported by several tools [1–4]. In particular, Gephi [3] supports graph streaming with a dedicated API based on JSON events and enables the association of timestamps to each graph component.

While there is some literature on dynamic layout of graphs [5–7], not much work has been done so far about developing information filtering techniques for dynamic visualization of large and quickly changing networks. Yet, for large networks in which the rate of structural changes in time could be very high, the task of determining the nodes and edges that can represent and transmit the salient structural properties of the network at a certain time is crucial to produce meaningful visualizations of the graph evolution.

We contribute to filling this gap by presenting a new graph filtering and visualization tool called `fastviz` that processes a chronological sequence of weighted interactions between the graph nodes and dynamically filters the most relevant parts of the network to visualize. Our algorithm:

- captures persistent trends in structural properties of dynamic networks, while removing no longer relevant portions of the networks and emphasizing old nodes and links that show fresh activity;
- smoothens transitions between the snapshots of a dynamic network by leveraging short-term and long-term node activity;
- uses limited memory and processor time and is fast enough to be applied to large live data streams and visualize their representation in the form of a network.

The reminder of this paper is structured as follows. First, we introduce related studies in Section 2. Next, we introduce the `fastviz` filtering method for dynamic networks in Section 3. We compare this method against rectangular and exponential sliding time-window approaches and show what are the advantages of our method. Finally, we present visualizations created with our filtering methods for four different real datasets in Section 4, and conclude the study.

2 Related work

Graph drawing [8, 9] is a branch of information visualization that has acquired great importance in complex systems analysis. A good pictorial representation of a graph can highlight its most important structural components, logically partition its different regions, and point out the most central nodes and the edges on which the information flows more frequently or quickly. The rapid development of computer-aided visualization tools and the refinement of graph layout algorithms [10–13] allowed increasingly higher-quality visualizations of large graphs [14]. As a result, many open tools for static graph analysis and visualization have been developed in the last decade. Among the best known we mention Walrus [15], Pajek [16, 17], Visone [18], GUESS [19], Networkbench [20], NodeXL [21], and Tulip [22]. Studies about comparisons of different tools have also been published recently [23].

The interest in depicting the shape of online social networks [24, 25] and the availability of live data streams from online social media motivated the development of tools for animated visualizations of *dynamic graphs* [26], in *offline* contexts, where temporal graph evolution is known in advance, as well as in *online* scenarios, where the graph updates are received in a streaming fashion [5]. Several tools supporting dynamics visualization emerged, including GraphAEL [1] (http://graphael.cs.arizona.edu/), GleamViz (www.gleamviz.org), Gephi [3] (gephi.org), and GraphStream [4] (graphstream-project. org). Despite static visualizations based on time-windows [23], alluvial diagrams [27], or matrices [28–30] have been explored as solutions to capture the graph evolution, dynamic graph drawing remains the technique that has attracted more interest in the research community so far. Compared to static visualizations, dynamic animations present additional challenges: user studies have shown that they can be perceived as harder to parse visually, even though they have the potential to be more informative and engaging [31].

As a result, a large corpus of work about the theoretical concepts on good visualization practices, especially for dynamic graphs, has been produced in the last two decades. Besides the work done in defining efficient update operations on graphs [32, 33], several principles about good graph visualizations have been proposed and explored in different studies. Friedrich and Eades [34] defined high-level guidelines for a good visualization of graph evolution with animations, including uniform, smooth and symmetrical movement of graph elements, with minimization of edge crossings and overviewing some techniques

that make the visualization more enjoyable, such as fadeout deletion of nodes. Graph *readability* has been measured in user studies in relation to several tasks [35–37]; the experimental findings highlight the importance of visualization criteria such as minimizing bends and edge crossings and maximizing cluster separation in facilitating the viewer's interpretation and understanding of the graph. A general concept that has been studied for long in relation to the quality of dynamic graph visualization is the *mental map* [38–40] that the viewer has of the graph structure. In practical terms, the placement of existing nodes and edges should change as little as possible when a change is made to the graph [41], under the hypothesis that if the mental map is preserved the parsing of the visual information is faster and more accurate. More recent work [42] has reappraised the importance of the mental map in the comprehension of a dynamic graph series, while identifying some cases in which it may help [43, 44] (e.g., memorability of the graph evolution, following long paths, recognition of recurrent patterns, tracking a large number of moving objects).

More in general, there are several open fronts in empirical research in graph visualization to identify the impact of certain factors on the quality of the animation (e.g., speed [45], interactivity [46]). An extensive overview of this aspect has been conducted recently by Kriglstein *et al.* [47]. Methods to preserve the stability of nodes and the consistency of the network structure leveraging hierarchical organization on nodes have been proposed [48–51]. User studies have shown that hierarchical approaches that collapse several nodes in larger meta-nodes can improve graph readability in cases of high edge density [52]. The graph layout also has a significant impact on the readability of graphs [53]. Some work has been done to adapt spectral and force-directed graph layouts [54] to *incremental layouts* that recompute the position of nodes at time t based on the previous positions at time $t-1$ minimizing displacement of vertices [5, 55–57] or to propose new "stress-minimization" strategies to map the changes in the graph [7].

Although much exploration has been done in the visualization principles to achieve highly-readable animations, two aspects have been overlooked so far.

First, not many techniques to extract and visualize the most relevant information from *very large* graphs have been studied yet. Graph decomposition has been used in a static context to increase the readability of the network by splitting it into modules to be visualized separately [58], while sliding time-windows have been employed to discard older nodes and edges in visualization of graph evolution [59]. A hierarchical organization of nodes according to some authority or centrality measure allows to visualize the graph at different levels of details, eliminating the need to display all nodes and edges at once [60]. Some work has been done about interactive exploration by blending different visualization paradigms [61] and time-varying clustering [62]. Indices to measure the relevance of events in a dynamic graph at both node and community level have also been proposed [63], even if they have not been applied to any graph animation task. Yet, none of these techniques has been tested on very large data and none of the modern visualization tools provide features for the detection of the most relevant components of a graph at a given time. On the other hand, quantitative studies on the characterization of temporal networks [64–66] have been conducted, but with no direct connection with the dynamic visualization task.

Last, the visualization of large graphs in an online scenario, where node and edge updates are received in a live stream, and the related practical implications of dynamic visualizations, have rarely been considered. In this context, just some exploratory work has been

carried out about information selection techniques for dynamic graph visualization, including solutions based on temporal decay of nodes and edges [59], node clustering [58], and centrality indices [60, 63].

3 Network filtering

We introduce the `fastviz` algorithm that takes in input a chronological stream of interactions between nodes (i.e., network edges) and converts it into a set of graph updates that account only for the most relevant part of the network. The algorithm has two stages: buffering of filtered network and generation of differential updates for the visualization (see Figure 1). The algorithm stores and visualizes the nodes with the highest strengths, i.e., the highest sum of weights of their connections.

3.1 Input

The data taken as input is an ordered chronological sequence of interactions between nodes. The interactions can be either pairwise or cliques of interacting nodes. For instance, the following input:

$$\langle t_i, n_1, \ldots, n_m, w_i \rangle$$

represents the occurrence of interactions between nodes $n_1, \ldots, n_m$ of weight w_i at epoch time t_i. Entries with more than two nodes are interpreted as interactions happening between each pair of members of the clique with the respective weight. Multiple interactions

Figure 1 Diagram of the algorithm. The diagram illustrating the main components of the algorithm.
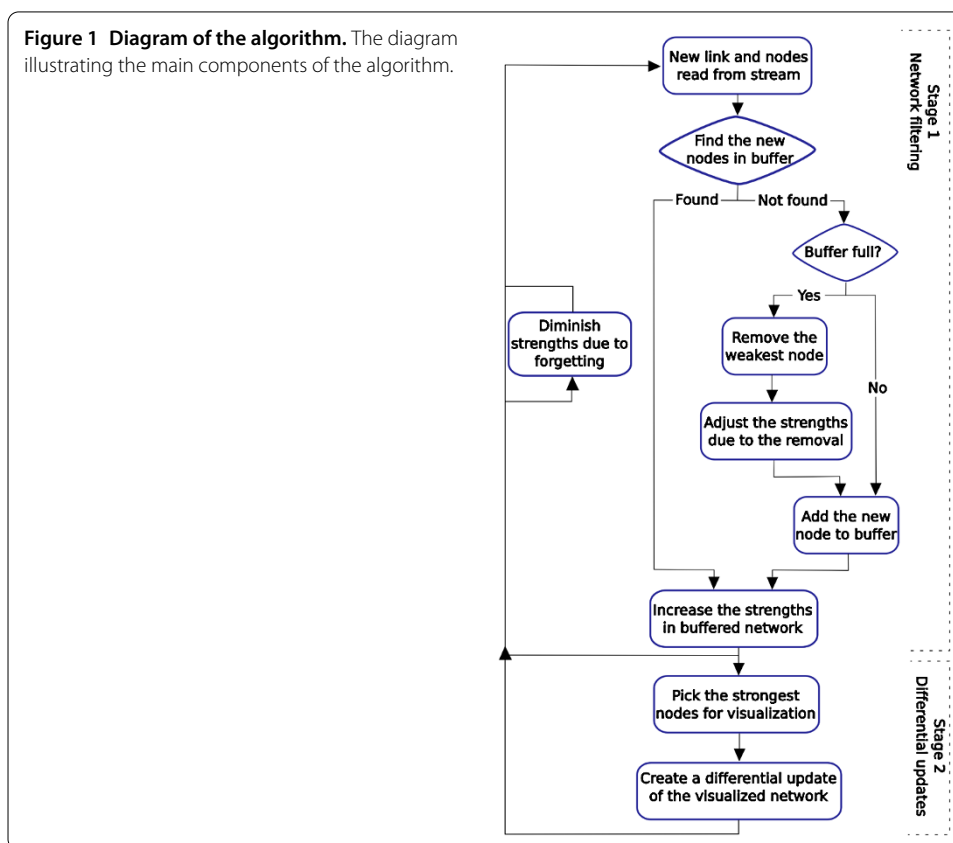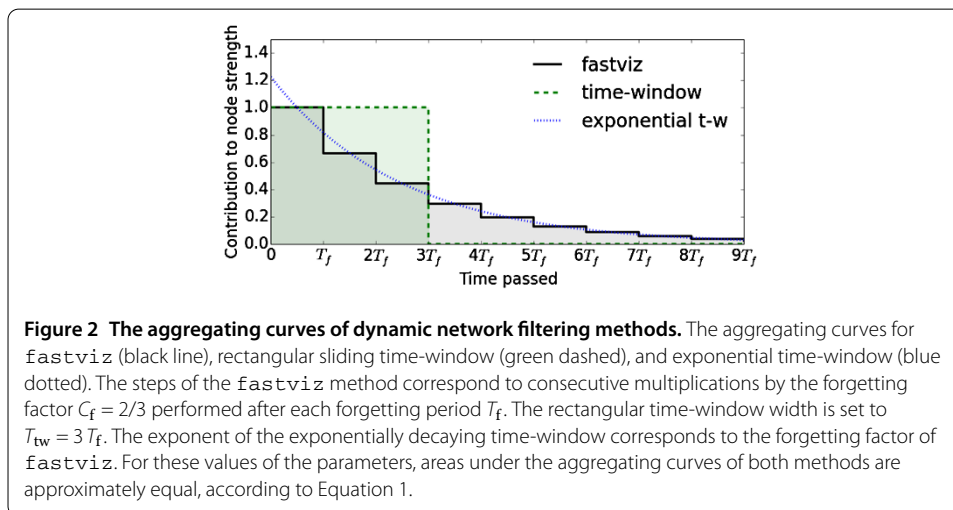
between the same pair of nodes sum up by adding up their corresponding weights. The advantage of the clique-wise format over the pairwise format is that the size of input files is smaller.

3.2 Filtering criterion

In the first stage of the algorithm, at most N_b nodes with the highest strengths are saved in the buffer together with the interactions among them. The strength S_i of a node i is a sum of weights of all connections of that node, i.e., $S_i = \sum_j w_{ij}$, where w_{ij} is the weight of an undirected connection between nodes i and j. Whenever a new node, which does not appear in the buffer yet, is read from the input, it replaces the node in the buffer with the lowest value of the strength. If an incoming input involves a node that is already in the buffer, then the strength of the node is increased by the weight of the incoming connection. To emphasize the most recent events and penalize stale ones, a forgetting mechanism that decreases the strengths of all nodes and weights of all edges is run periodically every time period T_f by multiplying their current values by a forgetting factor $0 \leq C_f < 1$. This process leads to the removal of old inactive nodes having low strength and storage of old nodes with fresh activity and high strength.

Note that the forgetting mechanism corresponds to a sliding time-window with exponential decay. The decay determines the weighting of the past interactions in the sliding time-window aggregation of a dynamic network. Standard rectangular sliding time-window aggregates all past events within the width T_{tw} of the time-window weighting them equally. In contrast, in fastviz and in the sliding time-window with an exponential decay the weighting decreases exponentially (see Figure 2). (Under a set of assumptions one can calculate how much time will a given node stay in the buffered network. Let us assume that at the time t_n the strength of a node n is $S_n(t_n)$, that this strength will not be increased after time t_n, that the next forgetting will happen in T_f time, and that the strength of the weakest buffered node $S_w < S_n(t_n)$ is constant over time. Under these assumptions, the node n will stay buffered for time $t - t_n > \frac{\log(S_w/S_n(t_n))}{\log(C_f)} T_f$.) Such exponential decay has two advantages over a standard rectangular sliding time-window approach. First, it gives more importance both to the most recent and to the oldest connections, while giving less importance to the middle-aged interactions. Second, it produces a dynamic network in

Figure 2 The aggregating curves of dynamic network filtering methods. The aggregating curves for fastviz (black line), rectangular sliding time-window (green dashed), and exponential time-window (blue dotted). The steps of the fastviz method correspond to consecutive multiplications by the forgetting factor $C_f = 2/3$ performed after each forgetting period T_f. The rectangular time-window width is set to $T_{tw} = 3 T_f$. The exponent of the exponentially decaying time-window corresponds to the forgetting factor of fastviz. For these values of the parameters, areas under the aggregating curves of both methods are approximately equal, according to Equation 1.

which changes are smoother due to the balanced weighting of old and new connections. Finally, instead of using the sliding time-window with exponential decay, we introduced the `fastviz` algorithm to limit the computational complexity of network filtering. In principle, time-window methods do not introduce such a bound. We explore and confirm these points in the following subsections using real dynamic networks.

3.3 Filtering criterion versus rectangular and exponential sliding time-windows

Comparison of structural properties of networks produced with different filtering methods is not straightforward. First, since the networks are dynamic, one needs to compare the structural properties of the static snapshots of the networks produced by the two methods at the same time. Second, parameters of the methods, i.e., forgetting factor C_f and time-window width T_{tw}, influence the algorithms, so one needs to draw an equivalency between them to compare the methods under the same conditions. A natural condition to consider is the one of equal areas under the curves from Figure 2, representing the contribution of an interaction event to the representation of a node over time. Note that under this condition a node with constant non-zero activity in time will have the same strength in networks created with each method. For `fastviz`, the area A_{fv} under the aggregation curve is equal to the sum of a geometric progression. Assuming an infinite geometric progression, we get the approximate $A_{fv} = T_f/(1 - C_f)$. The area under the aggregation curve of the rectangular time-window is simply $A_{tw} = T_{tw}$. By demanding the areas to be equal, we obtain the relation between the parameters of the two methods
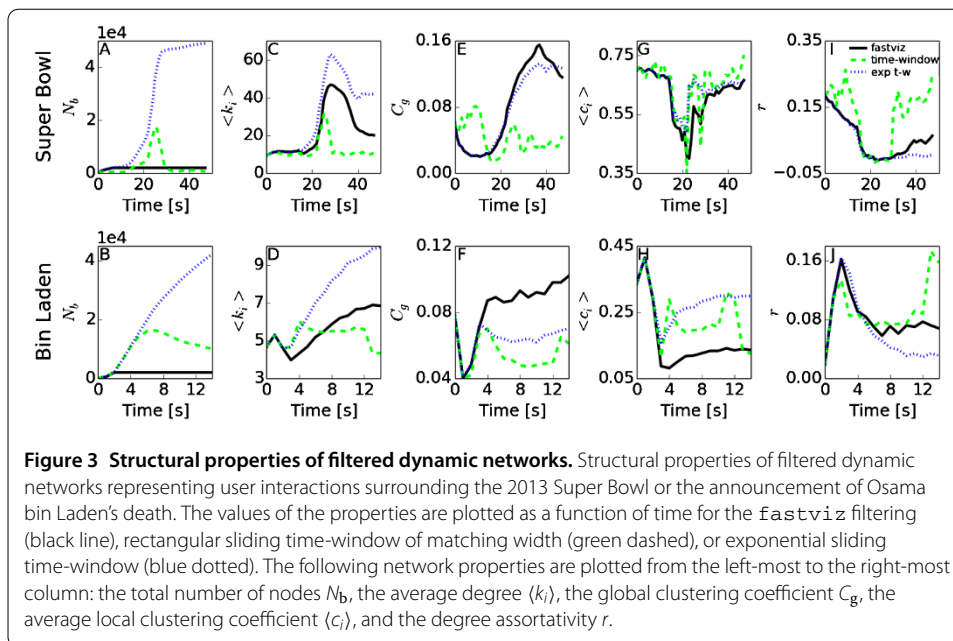
$$T_{tw} = \frac{T_f}{1 - C_f}. \tag{1}$$

In general, the forgetting period T_f is fixed, therefore there is only one free parameter controlling the filtering, e.g., the forgetting factor C_f, which we assign according to the dynamic network, i.e., the faster the network densifies in time, the more aggressive forgetting we use (see Appendix B for more details about the values of parameters). In the following paragraphs, we analyze the dynamics of several structural properties of the networks produced with `fastviz`, rectangular, and exponential sliding time-window methods having equal aggregating areas.

To highlight the differences between the three filtering methods, we apply them to two real dynamic networks from Twitter characterized by high changeability and measure the structural properties of resulting networks (Figure 3). The networks represent interactions in Twitter during two widely popular events: the 2013 Super Bowl and the announcement of Osama bin Laden's death. Further description and properties of these datasets are provided in the next section.

Due to this fact the computational complexity of sliding-time window methods increases in time, whereas it is bounded in `fastviz`. Since network structural properties such as average degree and clustering depend on the size of the network, we calculate these properties for the subgraphs of equal size, i.e., for the N_b strongest nodes of the full network produced by each of the sliding time-window methods (Figures 3C-J). For simplicity, we refer to these subgraphs of N_b nodes as the buffered networks.
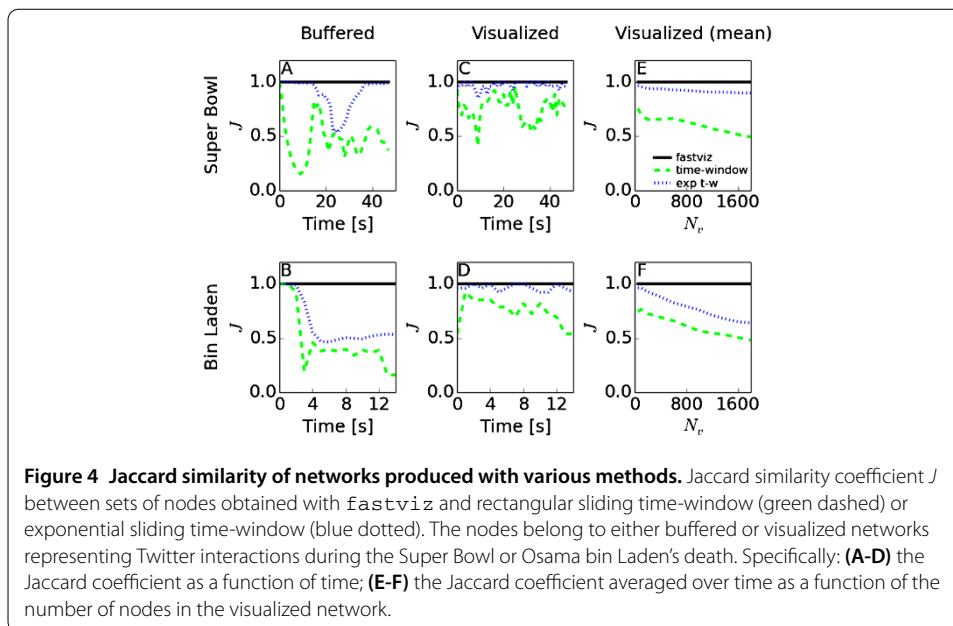
Second, we find that the networks produced with our filtering method do not experience drastic fluctuations of the global and local clustering coefficients and degree assortativity, which are especially evident for the rectangular time-window (Figures 3E, G, H,

Figure 3 Structural properties of filtered dynamic networks. Structural properties of filtered dynamic networks representing user interactions surrounding the 2013 Super Bowl or the announcement of Osama bin Laden's death. The values of the properties are plotted as a function of time for the `fastviz` filtering (black line), rectangular sliding time-window of matching width (green dashed), or exponential sliding time-window (blue dotted). The following network properties are plotted from the left-most to the right-most column: the total number of nodes N_b, the average degree $\langle k_i \rangle$, the global clustering coefficient C_g, the average local clustering coefficient $\langle c_i \rangle$, and the degree assortativity r.

and I). We conclude that the `fastviz` filtering produces smoother transitions between network snapshots than rectangular sliding time-window. This property of our method may improve readability of visualizations of such dynamic networks.

Finally, `fastviz` captures persistent trends in the values of the properties by leveraging the short-term and long-term node activity. For instance, it captures the trends in degree, clustering coefficients, and assortativity that are less visible with the rectangular time-window, while they are well-visible with the exponential time-window (Figures 3C-F, I, and J). Note that high average degree obtained for networks produced with exponential time-window corresponds to the nodes that are active over a prolonged time-span, whose activity is aggregated over unbounded aggregation period, and the number of nodes is unbounded as well. On the contrary, rectangular sliding time-window shows the degree aggregated over a finite time-window, while `fastviz` limits the number of tracked nodes, leading to lower reported average degree.

To measure the similarity of sets of nodes filtered with different methods we calculate Jaccard similarity coefficient. Specifically, we measure the Jaccard coefficient *J* of the sets of N_b strongest nodes filtered with `fastviz` and each of the time-window methods (Figures 4A and B). The value of the coefficient varies in time and among datasets. However, the similarity between `fastviz` and exponential time-window is significantly higher than between `fastviz` and rectangular time-window. For the Super Bowl dataset, the similarity between `fastviz` and exponential time-window is close to 1 most of the time and has a drop in the middle. The drop corresponds to the period of the game characterized by the intense turnout of nodes and edges in the buffered network. Hence, the similarity is not equal to 1 for the two methods because the weakest nodes are often forgotten and interchanged with new incoming nodes in `fastviz`, while in exponential time-window method they are not forgotten and can slowly become stronger over time. In the next subsection we show that this similarity is close to 1 at all times for the subsets of strongest nodes selected for visualization.
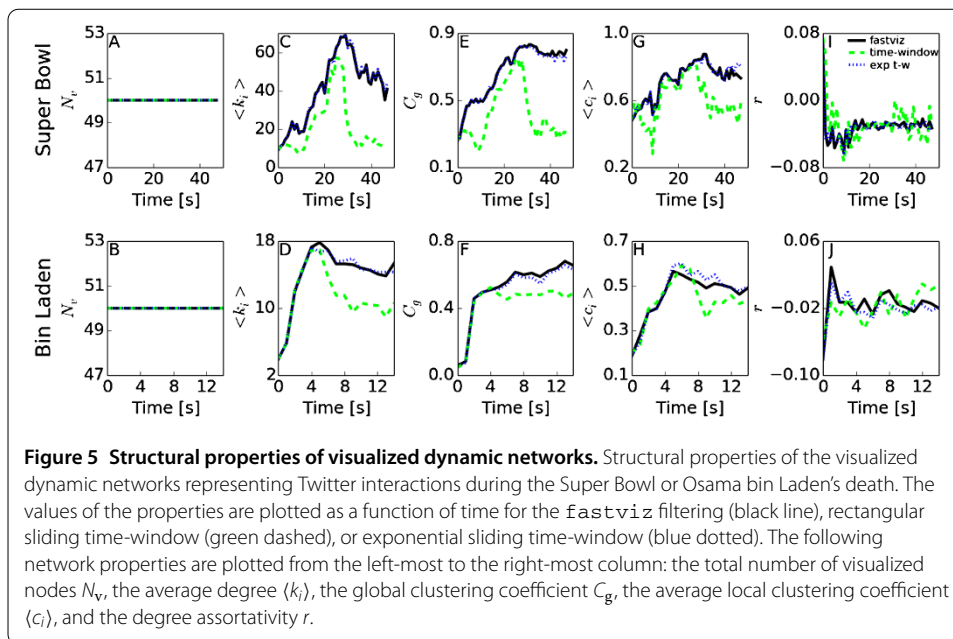
Figure 4 Jaccard similarity of networks produced with various methods. Jaccard similarity coefficient *J* between sets of nodes obtained with `fastviz` and rectangular sliding time-window (green dashed) or exponential sliding time-window (blue dotted). The nodes belong to either buffered or visualized networks representing Twitter interactions during the Super Bowl or Osama bin Laden's death. Specifically: **(A-D)** the Jaccard coefficient as a function of time; **(E-F)** the Jaccard coefficient averaged over time as a function of the number of nodes in the visualized network.

3.4 Network updates for visualization

In the second stage, for the purpose of visualization, the algorithm selects $N_v < N_b$ nodes with the highest strength and creates a differential update to the visualized network consisting of these nodes and the connections between them. Each such differential update is meant to be visualized in the resulting animation of the network, e.g., as a frame of a movie.

We compare the visualized networks generated by each of the filtering methods. Each of the visualized networks consists of $N_v = 50$ strongest nodes and all connections existing between them in the buffered network. The similarity of the nodes visualized by the `fastviz` and exponential time-window methods, measured as Jaccard coefficient *J*, is 1 or close to 1 (Figures 4C and D). The visualized networks of the two methods are almost identical. The structural properties of the networks created with the two methods yield almost the same values at each point in time (Figures 5A-J). This result is to be expected, since the forgetting mechanism of `fastviz` corresponds closely to the exponential decay of connection weights. The advantage of our method over exponential time-window consists of the limited computational complexity, which makes the `fastviz` filtering feasible even for the largest datasets of pairwise interactions. Naturally, the similarity between visualized networks created with the two methods decreases with the size of the visualized network N_v (Figures 4E and F). More specifically, the similarity decreases with the ratio N_v/N_b, as we keep in our experiments a constant value of $N_b = 2{,}000$. Hence, to visualize larger networks one can choose to buffer more nodes.

The comparison of the evolution of structural properties of the corresponding buffered and visualized networks shows that these networks differ significantly for each of the filtering methods (compare Figure 3 vs. Figure 5). This difference is the most salient in the case of rectangular time-window, which yields considerably larger fluctuations of structural properties than the other methods. In the cases of `fastviz` and exponential time-window some structural properties show evolution that is qualitatively similar for buffered and visualized networks, e.g., the average degree and the global clustering coefficient (Fig-

Figure 5 Structural properties of visualized dynamic networks. Structural properties of the visualized dynamic networks representing Twitter interactions during the Super Bowl or Osama bin Laden's death. The values of the properties are plotted as a function of time for the `fastviz` filtering (black line), rectangular sliding time-window (green dashed), or exponential sliding time-window (blue dotted). The following network properties are plotted from the left-most to the right-most column: the total number of visualized nodes N_v, the average degree $\langle k_i \rangle$, the global clustering coefficient C_g, the average local clustering coefficient $\langle c_i \rangle$, and the degree assortativity r.

ures 3C-F vs. Figures 5C-F). We conclude that the structure of visualized network differs significantly from the structure of buffered network, although this difference is smaller for `fastviz` than for rectangular sliding time-window.

3.5 Computational complexity

The computational complexity of the buffering stage of the algorithm is $\mathcal{O}(EN_b)$, where E is the total number of the pairwise interactions read (the cliques are made of multiple pairwise interactions). Each time when an interaction includes a node that is not yet stored in the buffered graph the adjacency matrix of the graph needs to be updated. Specifically, the weakest node is replaced with the new node, so N_b entries in the adjacency matrix are zeroed, which corresponds to $\mathcal{O}(EN_b)$. The memory usage scales as $\mathcal{O}(N_b^2)$, accounting for the adjacency matrix of the buffered graph. (For certain real dynamic networks, the buffered graph is sparse. In such cases, one can propose more optimized implementations of `fastviz`. Here, we focus on limiting the time complexity so that it scales linearly with the number of interactions and describe the generic implementation that achieves it.) The second, update-generating, stage has computational complexity of $\mathcal{O}(UN_b \log(N_b))$, where U is the total number of differential updates, which is a fraction of E and commonly it is many times smaller than E. (Typically, a large number of interactions is aggregated to create one differential update to the visualized network. In the examples that we show in the next section, one update aggregates from 400 to 2 million interactions. Therefore, U is from 400 to 2 million times smaller than E.) This term corresponds to the fact that the strengths of all buffered nodes are sorted each time an update to the visualized network is prepared. The memory trace of this stage is very low and scales as $\mathcal{O}(N_v)$. We conclude that our method has computational complexity that scales linearly with the number of interactions. It is therefore fast, that is, able to deal with extremely large dynamic networks efficiently.
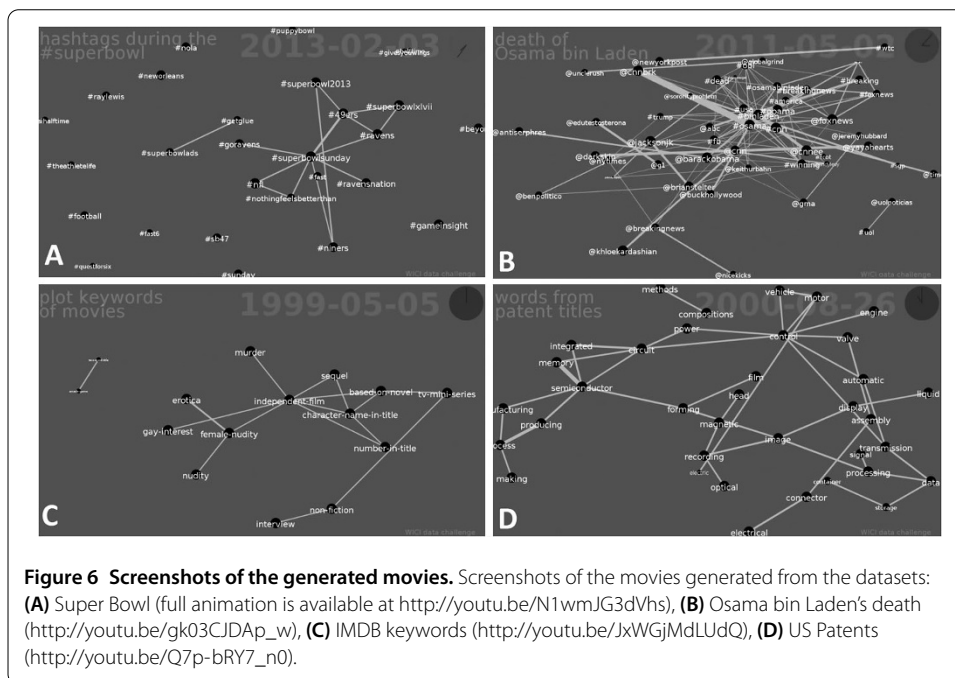
4 Visualization

In this section, we describe animations of exemplary dynamic graphs filtered with `fastviz`. Principally, the sequence of graph updates can be converted into image frames that are combined into a movie depicting the network evolution. We implement this visualizing technique and create with it the network animations described below. Alternatively, the updates can be fed directly to the Gephi Streaming API to produce an interactive visualization of the evolving network. The Gephi Streaming API allows graph streaming from a client to a server where Gephi is running. In such a case, the graphs are streamed directly from our filtering system to the Gephi server without any third-party modules. In Appendix A, we introduce implementation details of both approaches. Finally, corresponding animations can be created by other visualization tools fed with the `fastviz` updates; we highly encourage their development.

4.1 Datasets

We test the `fastviz` filtering and our visualizing technique on four datasets very different from each other in nature, size, and time span (see Table 1). The datasets and movies produced from each dataset are described in the following subsections (see Figure 6). In Appendix A, we present the source code of both tools with their documentation, four dynamic graph datasets, and instructions to recreate the visualizations introduced in this

Table 1 Statistics of the experimental datasets

Dataset	Time period	Nodes	Edges	Nodes drawn
Super Bowl	2 days	49k	1.1M	577
Osama bin Laden's death	2 hours	95k	198k	279
IMDB movie keywords	6 years	1k	220M	301
US patent title words	34 years	414k	90M	106

Figure 6 Screenshots of the generated movies. Screenshots of the movies generated from the datasets: **(A)** Super Bowl (full animation is available at http://youtu.be/N1wmJG3dVhs), **(B)** Osama bin Laden's death (http://youtu.be/gk03CJDAp_w), **(C)** IMDB keywords (http://youtu.be/JxWGjMdLUdQ), **(D)** US Patents (http://youtu.be/Q7p-bRY7_n0).

section. In Appendix B, we provide and describe the values of the parameters of the algorithm and the visualizing tool used for these datasets.

4.2 Twitter

We use data obtained through the Twitter *gardenhose* streaming API, which covers around 10% of the tweet volume. We focus on two events: the announcement of Osama bin Laden's death and the 2013 Super Bowl. We consider user mentions and hashtags as entities and their co-occurrence in the same tweet as interactions between them.

The first video (Figure 2A) shows how the anticipation for the Super Bowl steadily grows on early Sunday morning and afternoon, and how it explodes when the game is about to start. Hashtags related to #commercials and concerts (e.g., #beyonce) are evident. Later, the impact of the #blackout is clearly visible. The interest about the event drops rapidly after the game is over and stays low during the next day.

The video about the announcement of Osama bin Laden's death (Figure 2B) shows the initial burst caused by @keithurbahn and how the breaking news was spread by users @brianstelter and @jacksonjk. The video shows that the news appears later via #cnn and is announced by @obama. The breaking of this event on Twitter is described in detail by Lotan [67].

4.3 IMDB movies

We use a dataset from IMDB of all movies, their year of release and all the keywords assigned to them (from imdb.to/11SZD). We create a network of keywords that are assigned to the same movies. Our video (Figure 2C) shows interesting evolution of the keywords from "character-name-in-title" and "based-on-novel" (first half of 20th century), through "martial-arts" (70s and 80s) to "independent-film" (90s and later), "anime" and "surrealism" (2000s).

4.4 Patents

We use a set of US patents issued between 1976 and 2010 [68]. We analyze the appearance of words in their titles. Whenever two or more words appear in a title of a patent we create a link between them at the moment when the patent was issued. To improve readability we filter out stopwords and the generic frequent words: "method," "device" and "apparatus." Our video (Figure 2D) shows that at the beginning of the period techniques related to "engine" and "combustion" were popular, and later start to cluster together with "motor" and "vehicle." Another cluster is sparked by patents about "magnetic" "recording" and "image" "processing." It merges with a cluster of words related to "semiconductor" and "liquid" "crystal" to form the largest cluster of connected keywords at the end of the period.

4.5 Other visualizations

Other than these experimental datasets, on-demand animations of Twitter hashtag co-occurrence and diffusion (retweet and mention) networks can be generated with our tool via the Truthy service (truthy.indiana.edu/movies). Hundreds of videos have already been generated by the users of the platform and are available to view on YouTube (youtube.com/user/truthyatindiana/videos).

4.6 Summary

The datasets in our case studies are fairly diverse in topicality, time span, and size, as shown in Table 1. Nevertheless, our method is able to narrow down the visualization to meaningful small subgraphs with less than 600 distinct nodes in all cases. The high performance of the algorithm makes it viable for real-time visualizations of live and large data streams. On a desktop machine the algorithm producing differential updates of the network took several minutes to finish for the US patents and less than two minutes for the other datasets. Given such a performance, it is possible to visualize in real-time highly popular events such as the Super Bowl, which produced up to 4,500 tweets per second.

5 Conclusions

Tools for dynamic graph visualization developed so far do not provide specialized ways to dynamically select the most important portions of large evolving graphs. We contribute to filling this gap by proposing an algorithm to filter nodes and edges that best represent the network structure at a given time. Our method captures trends and smoothens the dynamics of structural properties of weighted networks by leveraging the short-term and long-term node activity. Furthermore, our filtering method uses limited memory and processor time making it viable for large live data streams. We implemented our filtering algorithm in open source tools that take in input a stream of interaction data and output a movie of the network evolution or a live Gephi animation. As future work, we wish to improve our algorithm by means of further optimization and to enhance the tools by providing a standalone module for live visualization of graph evolution.

Appendix A: Implementation details and source code

We have implemented two independent tools described in the manuscript. The first tool is the `fastviz` algorithm. The second tool converts the sequence of updates into image frames that are combined into a movie depicting the network evolution. We release the source code of both tools (see the project website github.com/WICI/fastviz). Here, we describe the two tools in more detail.

The first tool is the `fastviz` algorithm. It takes in input a chronological stream of interactions between nodes and converts it into a set of graph updates that account only for the most relevant part of the network in the JSON format. In the network filtering stage, the algorithm stores a buffered network of size N_b, limiting the computational complexity and memory usage of the algorithm. In the second stage, for the purpose of visualization, the algorithm selects $N_v < N_b$ nodes with the highest strength and all edges between these nodes with the highest strength and all edges between these nodes that have weight above a certain threshold w^{min}. The subgraph induced by the N_v nodes is compared with the subgraph in the previous state and a differential update is created. The updates are created per every time interval that is determined with the time contraction parameter T_c. A value of 10 for this parameter means that the time will flow in the visualization 10 times faster than in the data given as the input (see Appendix B). The differential updates are written in output in the form of a JSON file formatted according to the Gephi Streaming API [69]. We choose JSON format specifically due to the compatibility with Gephi Streaming API. In short, each line of the JSON file corresponds to one update of the graph structure and contains a sequence of JSON objects that specify the addition/deletion/attribute change

of nodes and edges. We also introduced a new type of object to deal with labels on the screen, for example, to write the date and time on the screen.

The second tool converts the sequence of updates into image frames that are combined into a movie depicting the network evolution. To this end, the sequence of updates produced by the filtering algorithm is fed to a python module that builds a representation of a *dynamic graph*, namely an object that handles each of the updates and reflects the changes to its current structure. The transition between the structural states of the graph determined by the received updates is depicted by a sequence of image frames. Each differential update correspond to one visualization frame, i.e., one frame of an animation. In its initial state, the nodes in the network are arranged according to the Fruchterman Reingold graph layout algorithm [11]. The choice of the layout is arbitrary and other layouts can be used and compared. However, due to the focus of this study on the filtering method, rather than the quality of the visualization, we do not explore any other layout algorithms. For each new incoming event, a new layout is computed by running N iterations of the layout algorithm, using the previous layout as a seed. Intermediate layouts are produced at each iteration of the algorithm. Every intermediate layout is converted to a png frame that is combined through the *mencoder* tool [70] to produce a movie that shows a smooth transition between different states. The movie is encoded with the frequency of 30 frames per second. To avoid nodes and edges to appear or disappear abruptly in the movie, we use animations that smoothly collapse dying nodes and expand new ones. A configuration file allows to modify the default movie appearance (e.g, resolution, colors) and layout parameters (see the project website).

We release the source code of both tools with the documentation under the GNU General Public License (see the project website github.com/WICI/fastviz). Together with the tools we release the datasets used in this paper and instructions on how to recreate all the examples of animations presented in this manuscript. Additionally, the updates created with `fastviz` can be fed directly to the Gephi Streaming API to produce an interactive visualization of the evolving network. Respective instructions can be found at the website of the project.

Appendix B: Algorithm parameters

The exact behavior of the `fastviz` filtering depends on the parameters introduced in the manuscript. We present the values of the parameters used in the case studies and their default values in Table 2. The default values of the parameters are meant to be universal and give reasonably good visualizations for most datasets. Overall, three parameters require adjustment to the input data, namely time contraction T_c, edge width threshold $w^{\min}$, and

Table 2 Values of the parameters of the `fastviz` algorithm for the introduced case studies

Dataset	T_c	$w^{\min}$	C_f
Super Bowl	3,600	10	0.8
Osama bin Laden's death	500	0.95	0.9
IMDB movie keywords	$3,600 \times 24 \times 1,095$	10	0.75
US patent title words	$3,600 \times 24 \times 400$	20	0.65
Default	3,600	0.95	0.75

The last row contains the default values of the corresponding parameters of the algorithm. Remaining parameters of the algorithm are set to their default values, i.e., $T_f = 10$ frames, $N_b = 2,000$, $N_v = 50$.

forgetting factor C_f. We provide exemplary values of these parameters for the introduced datasets in Table 2 and describe these parameters in detail below.

The time contraction T_c corresponds to the number of seconds in data time scale that are going to be contracted to one second of the visualization. The larger the time span of the dataset, the larger should be this parameter in order to keep the length of visualization fixed. For instance, if the timespan of the network is 10 hours, and one wants to see its evolution in a 10-second-long animation, then T_c should be set to 3,600. It is crucial to provide a desired value for this parameter, because providing a value that is too large will create just a few network updates and a very short animation, while providing a value that is too small will create a large number of updates making the JSON file very big and the animation very long.

The minimal edge weight w^{min} is a threshold above which edges appear in the visualization. Low value of this parameter may results in many edges of low weight appearing in the animation, while high value of the parameter may prevent any edges from being visualized. In case a user does not have any information about the visualized network, we recommend leaving this parameter at its default value of 0.95, which will visualize all edges of standard weight 1 or higher.

The forgetting factor C_f decides how fast older interactions among nodes are forgotten in comparison with more recent interactions. This parameter can be tuned individually for the purpose of the visualization. In general, the faster the network densifies in time, the more aggressive should be the forgetting, i.e., the lower should be the forgetting factor C_f. In general, keeping the default value of this parameter is safe, although its adjustment will improve the quality of visualization.

Authors' contributions
All authors designed the research. PAG wrote the source code of the algorithm and LMA wrote the source code of the visualization tool. All authors deployed the tools. PAG and LMA analyzed the data. All authors wrote, reviewed and approved the manuscript.

Author details
[1]Max Planck Institute for Software Systems, Saarland University, Saarbrucken, Germany. [2]Institute for Cross-Disciplinary Physics and Complex Systems, University of Balearic Islands, Palma de Mallorca, Spain. [3]Yahoo! Research, Barcelona, Spain. [4]Center for Complex Networks and Systems Research, Indiana University, Bloomington, USA.

References
1. Erten C, Harding P, Kobourov S, Wampler K, Yee G (2004) GraphAEL: graph animations with evolving layouts. In: Liotta G (ed) Graph drawing. Lecture notes in computer science, vol 2912. Springer, Berlin
2. Broeck W, Gioannini C, Goncalves B, Quaggiotto M, Colizza V, Vespignani A (2011) The GLEaMviz computational tool, a publicly available software to explore realistic epidemic spreading scenarios at the global scale. BMC Infect Dis 11:37
3. Bastian M, Heymann S, Jacomy M (2009) Gephi: an open source software for exploring and manipulating networks. In: ICWSM'09: proceedings of the international AAAI conference on weblogs and social media. AAAI Press, Menlo Park
4. Dutot A, Guinand F, Olivier D, Pigné Y (2007) GraphStream: a tool for bridging the gap between complex systems and dynamic graphs. In: EPNACS: emergent properties in natural and artificial complex systems
5. Brandes U, Fleischer D, Puppe T (2005) Dynamic spectral layout of small worlds. In: GD'05: proceedings of the 13th international symposium on graph drawing. Springer, Berlin

6. Brandes U, Fleischer D, Puppe T (2007) Dynamic spectral layout with an application to small worlds. J Graph Algorithms Appl 11(2):325-343
7. Brandes U, Indlekofer N, Mader M (2012) Visualization methods for longitudinal social networks and stochastic actor-oriented modeling. Soc Netw 34(3):291-308
8. Kamada T (1989) Visualizing abstract objects and relations. World Scientific, Singapore
9. Tollis IG, Di Battista G, Eades P, Tamassia R (1999) Graph drawing: algorithms for the visualization of graphs. Prentice Hall, New York
10. Kamada T, Kawai S (1989) An algorithm for drawing general undirected graphs. Inf Process Lett 31:7-15
11. Fruchterman TMJ, Reingold EM (1991) Graph drawing by force-directed placement. Softw Pract Exp 21(11):1129-1164. doi:10.1002/spe.4380211102
12. Gansner ER, North SC (1998) Improved force-directed layouts. In: GD'98: proceedings of the 6th international symposium on graph drawing. Springer, London
13. Hu YF (2005) Efficient and high quality force-directed graph drawing. The Mathematica J 10:37-71
14. Herman I, Melançon G, Marshall MS (2000) Graph visualization and navigation in information visualization: a survey. IEEE Trans Vis Comput Graph 6:24-43. doi:10.1109/2945.841119
15. Munzner TM (2000) Interactive visualization of large graphs and networks. PhD thesis, Stanford University, Stanford
16. Batagelj V, Mrvar A (2002) Pajek—analysis and visualization of large networks. In: Mutzel P, Junger M, Leipert S (eds) Graph drawing. Lecture notes in computer science, vol 2265. Springer, Berlin
17. De Nooy W, Mrvar A, Batagelj V (2005) Exploratory social network analysis with Pajek. Cambridge University Press, Cambridge
18. Brandes U, Wagner D (2003) Visone—analysis and visualization of social networks. In: Graph drawing software. Springer, Berlin
19. Adar E (2006) GUESS: a language and interface for graph exploration. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'06. ACM, New York
20. Network workbench tool, Indiana University, Northeastern University, and University of Michigan. http://nwb.cns.iu.edu. Accessed 13 Mar 2014
21. Smith MA, Shneiderman B, Milic-Frayling N, Mendes Rodrigues E, Barash V, Dunne C, Capone T, Perer A, Gleave E (2009) Analyzing (social media) networks with NodeXL. In: C&T'09: proceedings of the fourth international conference on communities and technologies. ACM, New York
22. Auber D, Archambault D, Lambert RBA, Mathiaut M, Mary P, Delest M, Dubois J, Melancon G (2012) The tulip 3 framework: a scalable software library for information visualization applications based on relational data. Technical report 7860, INRIA
23. Ahn JW, Taieb-Maimon M, Sopan A, Plaisant C, Shneiderman B (2011) Temporal visualization of social network dynamics: prototypes for nation of neighbors. In: SBP'11: proceedings of the 4th international conference on social computing, behavioral-cultural modeling and prediction. Springer, Berlin
24. Heer J, Vizster BD (2005) Visualizing online social networks. In: InfoVis'05: proceedings of the IEEE symposium on information visualization. IEEE Computer Society, Washington
25. Falkowski T, Bartelheimer J, Spiliopoulou M (2006) Mining and visualizing the evolution of subgroups in social networks. In: WI'06: proceedings of the 2006 IEEE/WIC/ACM international conference on web intelligence. IEEE Computer Society, Washington
26. Demetrescu C, Eppstein D, Galil Z, Italiano GF (2010) Dynamic graph algorithms. In: Atallah MJ, Blanton M (eds) Algorithms and theory of computation handbook. Chapman & Hall/CRC, Boca Raton
27. Rosvall M, Bergstrom CT (2010) Mapping change in large networks. PLoS ONE 5:e8694. doi:10.1371/journal.pone.0008694
28. Yi JS, Elmqvist N Lee S (2010) TimeMatrix: analyzing temporal social networks using interactive matrix-based visualizations. Int J Hum-Comput Interact 26:11-12
29. Stein K, Wegener R, Schlieder C (2010) Pixel-oriented visualization of change in social networks. In: ASONAM'10: proceedings of the international conference on advances in social networks analysis and mining. IEEE Computer Society, Washington
30. Gove R, Gramsky N, Kirby R, Sefer E, Sopan A, Dunne C, Shneiderman B, Taieb-Maimon M (2011) NetVisia: heat map & matrix visualization of dynamic social network statistics & content. In: SocialCom'11: proceedings of the 3rd IEEE international conference on social computing. IEEE Computer Society, Washington
31. Farrugia M, Quigley A (2011) Effective temporal graph layout: a comparative study of animation versus static display methods. Inf Vis 10:47-64. doi:10.1057/ivs.2010.10
32. Ramalingam G, Reps T (1996) On the computational complexity of dynamic graph problems. Theor Comput Sci 158:233-277
33. Henzinger MR, King V (1999) Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J ACM 46:502-516
34. Friedrich C, Houle ME (2002) Graph drawing in motion II. In: Mutzel P, Junger M, Leipert S (eds) Graph drawing. Lecture notes in computer science, vol 2265. Springer, Berlin, pp 220-231. doi:10.1007/3-540-45848-4_18
35. Purchase HC (1997) Which aesthetic has the greatest effect on human understanding? In: GD'97: proceedings of the 5th international symposium on graph drawing. Springer, London, pp 248-261. http://dl.acm.org/citation.cfm?id=647549.728779
36. Huang W, Hong SH, Eades P (2006) How people read sociograms: a questionnaire study. In: APVis'06: proceedings of the 2006 Asia-Pacific symposium on information visualisation—volume 60. Australian Computer Society, Darlinghurst, pp 199-206. http://dl.acm.org/citation.cfm?id=1151903.1151932
37. Huang W, Eades P, Hong SH (2008) Beyond time and error: a cognitive approach to the evaluation of graph drawings. In: Proceedings of the 2008 workshop on BEyond time and errors: novel evaLuation methods for Information Visualization, BELIV'08, vol 3. ACM, New York, pp 1-8. http://doi.acm.org/10.1145/1377966.1377970
38. Eades PWL, Misue K, Sugiyama K (1991) Preserving the mental map of a diagram. In: Compugraphics, pp 24-33
39. Misue K, Eades P, Lai W, Sugiyama K (1995) Layout adjustment and the mental map. J Vis Lang Comput 6(2):183-210
40. Freire M, Rodríguez P (2006) Preserving the mental map in interactive graph interfaces. In: AVI'06: proceedings of the working conference on advanced visual interfaces. ACM, New York

41. Coleman MK, Parker DS (1996) Aesthetics-based graph layout for human consumption. Softw Pract Exp 26(12):1415-1438. doi:10.1002/(SICI)1097-024X(199612)26:12<1415::AID-SPE69>3.0.CO;2-P

42. Archambault D, Purchase HC (2013) The "Map" in the mental map: experimental results in dynamic graph drawing. Int J Hum-Comput Stud 71(11):1044-1055. http://www.sciencedirect.com/science/article/pii/S107158191300102X

43. Archambault D, Purchase H (2012) The mental map and memorability in dynamic graphs. In: 2012 IEEE Pacific visualization symposium (PacificVis), pp 89-96

44. Archambault D, Purchase H (2013) Mental map preservation helps user orientation in dynamic graphs. In: Didimo W, Patrignani M (eds) Graph drawing. Lecture notes in computer science, vol 7704. Springer, Berlin, pp 475-486. doi:10.1007/978-3-642-36763-2_42

45. Ghani S, Elmqvist N, Yi JS (2012) Perception of animated node-link diagrams for dynamic graphs. Comput Graph Forum 31(3):1205-1214. doi:10.1111/j.1467-8659.2012.03113.x

46. Archambault D, Munzner T, Auber D (2008) GrouseFlocks: steerable exploration of graph hierarchy space. IEEE Trans Vis Comput Graph 14(4):900-913. doi:10.1109/TVCG.2008.34

47. Kriglstein S, Pohl M, Stachl C (2012) Animation for time-oriented data: an overview of empirical research. In: 16th international conference on information visualisation, pp 30-35

48. North SC (1996) Incremental layout in DynaDAG. In: GD'95: proceedings of the symposium on graph drawing. Springer, London

49. North SC, Woodhull G (2002) Online hierarchical graph drawing. In: GD'01: revised papers from the 9th international symposium on graph drawing. Springer, London

50. Archambault D, Munzner T, Auber D (2007) TopoLayout: multilevel graph layout by topological features. IEEE Trans Vis Comput Graph 13(2):305-317

51. Archambault D (2009) Structural differences between two graphs through hierarchies. In: GI'09: proceedings of graphics interface. Canadian Information Processing Society, Toronto

52. Archambault D, Purchase H, Pinaud B (2010) The readability of path-preserving clusterings of graphs. Comput Graph Forum 29(3):1173-1182. http://hal.inria.fr/inria-00471432

53. Blythe J, McGrath C, Krackhardt D (1996) The effect of graph layout on inference from social network data. In: GD'95: proceedings of the symposium on graph drawing. Springer, London, pp 40-51. http://dl.acm.org/citation.cfm?id=647547.728581

54. Brandes U (2001) Drawing on physical analogies. In: Kaufmann M, Wagner D (eds) Drawing graphs. Springer, London

55. Branke J (2001) Dynamic graph drawing. In: Kaufmann M, Wagner D (eds) Drawing graphs. Springer, London

56. Diehl S, Gorg C (2002) Graphs, they are changing. In: Goodrich M, Kobourov S (eds) Graph drawing. Lecture notes in computer science, vol 2528. Springer, Berlin

57. Frishman Y, Tal A (2008) Online dynamic graph drawing. IEEE Trans Vis Comput Graph 14:727-740

58. Rodrigues EM, Milic-Frayling N, Smith M, Shneiderman B, Hansen D (2011) Group-in-a-box layout for multi-faceted analysis of communities. In: SocialCom'11: proceedings of the 3rd IEEE international conference on social computing. IEEE Computer Society, Washington

59. Dynes SBC, Gloor PA, Gloor PA, Gloor PA, Laubacher R, Laubacher R, Zhao Y, Zhao Y, Dynes S (2004) Temporal visualization and analysis of social networks. In: NAACSOS'04: conference of North American Association for Computational Social and Organizational Science

60. Kumar G, Garland M (2006) Visual exploration of complex time-varying graphs. IEEE Trans Vis Comput Graph 12:805-812

61. Hadlak S, Schulz HJ, Schumann H (2011) In situ exploration of large dynamic networks. IEEE Trans Vis Comput Graph 17(12):2334-2343. http://dblp.uni-trier.de/db/journals/tvcg/tvcg17.html#HadlakSS11

62. Sallaberry A, Muelder C, Ma KL (2013) Clustering, visualizing, and navigating for large dynamic graphs. In: GD'12: proceedings of the 20th international conference on graph drawing. Springer, Berlin, pp 487-498. http://dx.doi.org/10.1007/978-3-642-36763-2_43

63. Asur S, Parthasarathy S, Ucar D (2007) An event-based framework for characterizing the evolutionary behavior of interaction graphs. In: KDD'07: proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York

64. Clauset AEN (2007) Persistence and periodicity in a dynamic proximity network. In: DIMACS workshop on computational methods for dynamic interaction networks

65. Cattuto C, Van den Broeck W, Barrat A, Colizza V, Pinton JF, Vespignani A (2010) Dynamics of person-to-person interactions from distributed RFID sensor networks. PLoS ONE 5(7):e11596. doi:10.1371/journal.pone.0011596

66. Krings G, Karsai M, Bernhardsson S, Blondel V, Saramaki J (2012) Effects of time window size and placement on the structure of an aggregated communication network. EPJ Data Sci 1:4. http://www.epjdatascience.com/content/1/1/4

67. Lotan G (2011) Breaking bin Laden: a closer look. http://blog.socialflow.com/post/5454638896/breaking-bin-laden-a-closer-look. Accessed 13 Mar 2014

68. LaRowe G, Ambre S, Burgoon J, Ke W, Börner K (2009) The scholarly database and its utility for scientometrics research. Scientometrics 79(2):219-234

69. Graph Streaming API documentation. http://wiki.gephi.org/index.php/Specification_-_GSoC_Graph_Streaming_API. Accessed 13 Mar 2014

70. Mencoder tool documentation. http://www.mplayerhq.hu/design7/documentation.html. Accessed 13 Mar 2014