Event-Based Dynamic Graph Visualisation

Paolo Simonetto, Daniel Archambault, and Stephen Kobourov

Abstract—Dynamic graph drawing algorithms take as input a series of timeslices that standard, force-directed algorithms can exploit to compute a layout. However, often dynamic graphs are expressed as a series of events where the nodes and edges have real coordinates along the time dimension that are not confined to discrete timeslices. Current techniques for dynamic graph drawing impose a set of timeslices on this event-based data in order to draw the dynamic graph, but it is unclear how many timeslices should be selected: too many timeslices slows the computation of the layout, while too few timeslices obscures important temporal features, such as causality. To address these limitations, we introduce a novel model for drawing event-based dynamic graphs and the first dynamic graph drawing algorithm, DynNoSlice, that is capable of drawing dynamic graphs in this model. DynNoSlice is an offline, force-directed algorithm that draws event-based, dynamic graphs in the space-time cube (2D+time). We also present a method to extract representative small multiples from the space-time cube. To demonstrate the advantages of our approach, DynNoSlice is compared with state-of-the-art timeslicing methods using a metrics-based experiment. Finally, we present case studies of event-based dynamic data visualised with the new model and algorithm.

Index Terms—Information Visualisation, Graph Drawing, Dynamic Graphs, Event-Based Analytics, No Timeslices.

1 Introduction

The notion of timeslicing is integral to dynamic graph visualisation, as well as information visualisation in general. Surveys of dynamic network visualisation methods [3], [13] use the timeslice as a basis to categorise techniques. For dynamic graphs, animation and small multiples techniques often rely on a timesliced definition of dynamic data. Thus, the timeslice has been part of the definition of dynamic graphs up to this point.

Dynamic graph drawing algorithms and evaluations thereof also use the timeslice [15], [16], [27], [31]. The dynamic graph drawing model for these approaches requires timeslices to be defined beforehand (Fig. 1a). In these models, the dynamic graph consists of a series of discrete snapshots of the graph in time. At any given timeslice, if a node is present in this timeslice but not the next one it is deleted. Similarly, if it is not present in this timeslice but is in the next one, it is added. Inter-timeslice edges connect a node in the current timeslice to the same node immediately before and after (if present), in order to encourage the stability of the drawing [5], allowing for easy identification of the same node across time without highlighting [4]. The algorithm then optimises this model to balance the competing goals of high quality layout of the individual graphs in each timeslice and the overall drawing stability.

Drawing dynamic graphs using timeslices makes sense for a number of reasons. First, dynamic graph drawing algorithms that use a timeslice model can be designed by simple extensions of existing static graph drawing algorithms. Second, when the input data is naturally structured into timeslices, it makes sense to think of the dynamic graph as a series of snapshots that encode graph evolution. However, often the dynamic data comes as a series of events where the nodes and edges in the graph have real-valued time coordinates. In this case, in order to use the ideas above, regular timeslices need to be imposed on the data and projected down onto the nearest timeslice. But, how many timeslices should be selected? If too many timeslices are selected computation is unnecessarily slowed due to the extra timeslices in the data set. If too few timeslices are selected, information is lost through aggregation across time. This aggregation will obscure the order of insertion of edges and nodes which obscures graph structure (Fig. 2). The Nyquist frequency states that if we want to be sure that no information is lost, we must select timeslices at a rate equal to the smallest gap between events. In most data sets, such a number of timeslices would be prohibitively large.

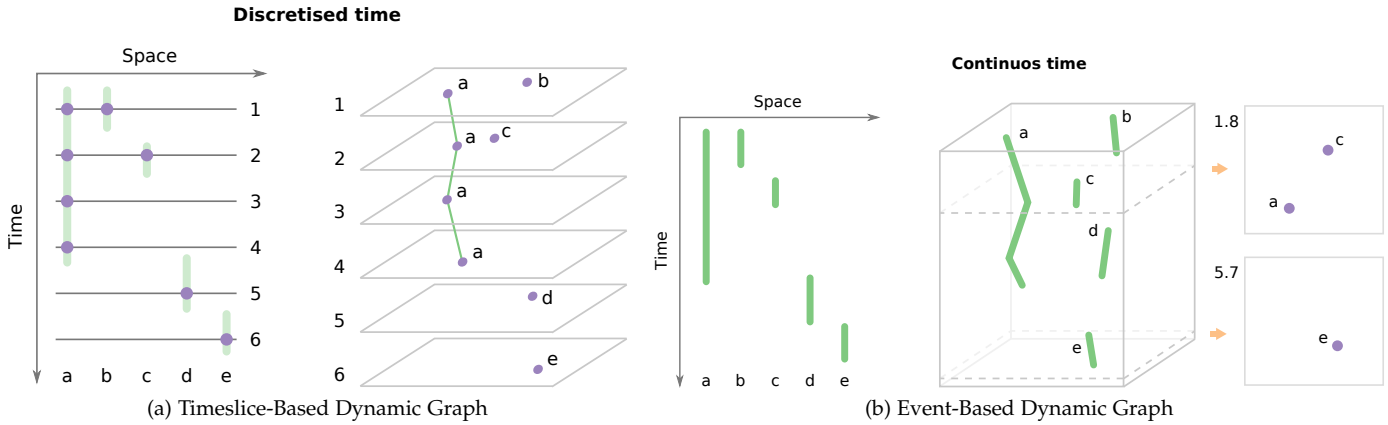
The problem of selecting an appropriate number of timeslices is further compounded by other factors. When both high and low frequency features exist in the dynamic data, a uniform set of timeslices across the entire data set may not be appropriate. Imposing timeslices on event-based data can actually induce instability into the drawing. As inter-timeslice edges are always required if a node appears in consecutive timeslices, non-interacting nodes will undergo unnecessary movement. Finally, if we want to look at graph structure between timeslices, we must re-project the graph onto timeslices and redraw the entire data set. Such limitations of timeslice-based drawing methods serve as good motivation to pursue a model for dynamic graph drawing that is not based on timeslices.

Beck *et al.* [12, p. 15] write that *"the effects of using continuous time with arbitrary fine sampling rates, rather than discretised time, are largely unexplored"*. In this paper, we introduce a new model (Fig. 1b) for drawing dynamic graphs without timeslices. We then describe the first dynamic graph drawing algorithm, DynNoSlice, that uses this model to embed the dynamic graph in the space-time cube. In this model, nodes and edges can have real-valued time coordinates (as opposed to the integer-based coordinates

- *P. Simonetto was with Swansea University. E-mail: paolo.simonetto@gmail.com*
- *D. Archambault is with Swansea University. E-mail: d.w.archambault@swansea.ac.uk*
- *S. Kobourov is with the University of Arizona. E-mail: kobourov@cs.arizona.edu*

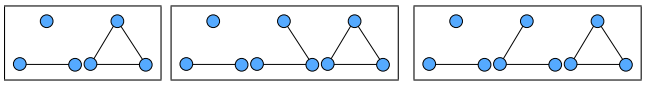
Manuscript received December 23rd, 2017.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2886901, IEEE Transactions on Visualization and Computer Graphics



(a) Timeslice-Based Dynamic Graph (b) Event-Based Dynamic Graph

Fig. 1. A dynamic graph with 5 nodes. (a) Dynamic graph represented using timeslices. Nodes are projected into each timeslice. The projection of these nodes (purple dots) causes information to be lost through aggregation across time. (b) In our approach, we do not impose timeslices on event data. Rather, nodes are defined as piecewise linear curves in the space-time cube.



(a) Graph (b) Possibility 1 (c) Possibility 2

Fig. 2. Selecting too few timeslices obscures temporal structure. A simple pattern is selected, but more complicated patterns in causality exist. **(a)** A dynamic graph where too few timeslices have been selected. Without changing the encoding, we cannot distinguish **(b)** or **(c)**.

induced by timeslicing). Nodes are polylines in the space-time cube while edges are ruled surfaces. Animation is a straightforward way to visualise the space time cube, but we also provide a method to extract and visualise a small multiples representation of its contents. To demonstrate the advantages of event-based dynamic graph drawing, we compare DynNoSlice with state-of-the-art timeslicing methods using a metrics-based experiment. Finally, we present case studies of event-based dynamic data visualised with the new model and algorithm.[1]

2 Related Work

DynNoSlice can be categorised as a offline event-based, dynamic graph drawing algorithm that does not use timeslices to lay out the dynamic graph. In order to accomplish this goal, it embeds the graph inside the space-time cube [7]. With this in mind, we review related work in dynamic graphs and event-based visualisation.

2.1 Dynamic Graph Drawing

Dynamic graph drawing and visualisation now has a long history [13]. Many algorithms have been proposed, but all of them rely on timeslices as a part of the approach.

1. An earlier version of this work was presented at the *Graph Drawing and Network Visualisation (GD '17)* conference [62]. The sections of this paper that describe the dynamic graph drawing algorithm (mainly Sections 3 and 4) are based on this content but expanded to provide more details for reproducibility. All other parts of this paper are new. Specifically, we extend our technical contribution by introducing two new methods to extract interesting small multiples, run a new metrics experiment, and provide additional case studies. Note that in the earlier version [62], we used the term *continuous dynamic graphs* for our model. After feedback at the conference, we felt that *event-based dynamic graph* was a more accurate term which we use here.

2.1.1 Offline

Offline dynamic graph drawing algorithms capture all of the dynamic data beforehand and optimise across it simultaneously. These algorithms have the advantage that they are supplied with all relevant information for the time period in question and can optimise across it.

Offline dynamic graph drawing and visualisation has taken a number of different perspectives: aggregating all time periods into a single supergraph [23], [24], linking the same node in consecutive timeslices together with inter-timeslice edges while optimising all timeslices simultaneously [27], [28], [31], providing simultaneous support for animation and small multiples [8], and evaluating the many different approaches to connect nodes across the same timeslice in terms of how well they perform [15]. Concepts in offline dynamic graph drawing have also been applied to the area of dynamic dimensionality reduction [58].

DynNoSlice is an offline dynamic graph drawing algorithm. However, all of the above described approaches require timeslices to draw and/or visualise the dynamic graph. Our approach draws event-based dynamic graphs directly without timeslices.

2.1.2 Online

Online dynamic graph drawing algorithms do not have access to the full data over the entire period of interest. These approaches optimise the current view, given what has happened in the past, and cannot look forward to future events as the information is not available.

Early approaches for online dynamic graph drawing maintained the horizontal and vertical position of nodes [51], used node ageing methods to update positions over time [38], and adapted multilevel approaches [20] from the static graph drawing literature [42]. Online approaches have also been implemented on the GPU [34].

Although related, online approaches are substantially different from our approach. Online algorithms have no access to future information, whereas in the offline setting the entire graph evolution is known in advance. The input is fundamentally different: in the online problem, timeslices at time t are created by adding/removing nodes and edges

from $t-1$, while in the offline problem a layout for the entire cube considers all information. One can think of online approaches as aggregating all time onto a single timeslice while diminishing the influence of older data.

2.2 Time to Space Techniques

Time to space systems use one of the spatial dimensions to encode time. These visualisations look very different from those discussed in section 2.1 and have one dimension less to provide an overview of the graph structure

A number of techniques have been implemented that embed a representation of the dynamic network in two dimensional space where one of the dimensions is at least partially influenced by time. Many of these techniques explicitly use one dimension as the time dimension (1D+time) [17], [50], [60], [65]. While others use dimensionality reduction to place timeslices in similar positions in the two dimensional plane if the structure of the data is similar [10], [66].

These techniques provide effective, complimentary visualisations of dynamic graphs that are substantially different from DynNoSlice because they devote at least part of one dimension to time. Also, most of them still use the timeslice as a basis for drawing and visualisation.

Some techniques use three dimensions (2D+time). Itoh *et al.* [44] describe a technique that allows interaction with timeslices in three dimensions. Other techniques use the space-time cube directly to visualise a dynamic graph in a three dimensional setting [9] or through specific views of this cube [59]. Many techniques describe the evolution of nodes as tubes [41], layers [14], or worms [2], [26] as a metaphor where nodes are represented as polylines through the space time cube.

These techniques visualise networks in three dimensions and often nodes in this space appear as polylines. However, all of these techniques use timeslices as a basis to create the visualisation of the dynamic graph, while in this paper we do not use timeslices to draw the event-based data.

2.3 3D Graph Drawing

A number of visualisation approaches use the full three dimensions of space in order to draw and visualise a graph. These approaches are used for static graphs that do not evolve over time.

Both force-directed [33] (adapted from `Gem` [35]) and simulated annealing [21] (adapted from Davidson and Harel [22]) graph drawing approaches have been used to draw graphs directly in 3D. Multilevel algorithms [36] have also been developed in order to scale to larger data sets. Specialised graph drawing approaches, such as orthogonal [46] and compound graph [57] drawings, have been extended to 3D as well. Munzner [54] uses spanning trees to visualise graphs in 3D hyperbolic space, and Cordeil *et al.* [19] use immersive environments to visualise graphs.

DynNoSlice has forces that are computed in 3D that are similar to some of the 3D force directed algorithms above. However, our approach is not a 3D force directed algorithm. DynNoSlice has additional constraints that embed the dynamic graph in a 2D+time, space-time cube. After drawing the event-based data, we can visualise it through animation or small multiples.

2.4 Event-Based Analytics

Event-based techniques exist for directly visualising event sequences of dynamic data [25], [52], [53]. Event-based approaches do not consider timeslices, but consider individual events with real time coordinates. Often these techniques provide interactive methods for aggregating large volumes of individual events over time.

Often these events consist of scalar data across time. Our approach can be considered the first dynamic graph drawing algorithm that is event-based.

2.5 Temporal Networks in Complex Networks

Similar models for dynamic graphs have been studied in the field of complex networks and in the emerging area of temporal networks [43] with preliminary visualisations of such networks using time as the x-axis (similar to Section 2.2). Models for computing metrics on this type of data (e.g. density, clustering coefficient and others) have been formalised without timeslices [47].

These papers focus on formalising and modelling structures in stream graphs and include visualisations of event-based dynamic networks that capture dense areas of edges across time. We focus on algorithms for visualisation of such networks, and in particular we present a model, a force-directed algorithm, and visualisations from the algorithm that represent such networks in 2D+time. Our visualisations and drawings are interesting as they are better able to clarify network topology with time.

3 Event-Based Dynamic Graph Model

Let $G = (V, E)$ be a static graph defined with node set V and edge set $E \subseteq V \times V$. We define *attributes* (functions) on the nodes and edges of the graph that encode characteristics such as their positions, weights, and labels. The node position attribute ($\mathcal{P}_G : V \to \mathbb{R}^2$) maps a node v into its position in the 2D plane $\boldsymbol{p}_v$, e.g., $\mathcal{P}_G(v) = (1, 4)$. This attribute is not integral to our model of event-based dynamic graphs, but is used to compute and store the layout of these graphs.

We define an event-based dynamic graph $D = (V, E)$ as a graph whose attributes are also a function of time. Let T be the time domain defined as an interval in $\mathbb{R}$. Attributes are functions defined in the domain $V \times T$ for nodes, and $E \times T$ for edges. For example, the node position attribute $\mathcal{P}_D : V \times T \to \mathbb{R}^2$ is the function that describes the position of v for each time $t \in T$. We assume without loss of generality that the node and edge attributes can be defined piecewise, e.g.:

$$\mathcal{P}_D(v,t) = \mathcal{P}_v(t) = \begin{cases} \mathcal{P}_{v,1}(t) & \text{for } t \in T_1 \\ \vdots & \\ \mathcal{P}_{v,n}(t) & \text{for } t \in T_n \\ \boldsymbol{p}_{v,\omega} & \text{otherwise} \end{cases}$$

In other words, a dynamic attribute can be thought of as a map that links each node/edge to a sequence of functions $\mathcal{P}_{v,i}$ that describe its behaviour in disjoint intervals of time T_i, with a default value returned for $t \notin \bigcup_i T_i$. In the rest of paper, we consider only piecewise linear functions for nodes and edges.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2886901, IEEE Transactions on Visualization and Computer Graphics

Attributes specify a variety of node and edge characteristics. For data that can be meaningfully interpolated (e.g., colours, weights, positions), the functions above can be described by initial and final values. For attributes without meaningful interpolation (e.g., labels), we prefer functions that are constant in the related interval. For example, the position and label attributes for a node v of D can be:

$$\mathcal{P}_v(t) = \begin{cases} (1,0) \to (4,0) & \text{for } t \in (9,12] \\ (2,-2) \to (5,1) & \text{for } t \in (12,15] \\ (5,1) \to (4,5) & \text{for } t \in [17,19] \\ (0,0) & \text{otherwise} \end{cases}$$

$$\mathcal{L}_v(t) = \begin{cases} \text{Jane Doe} & \text{for } t \in (10,11] \\ \text{Jane Smith} & \text{for } t \in (11,16] \\ \text{unknown} & \text{otherwise} \end{cases}$$

We define the attribute *appearance* that implements the classic dynamic graph operations node/edge insertion and deletion.

$$\mathcal{A}_v(t) = \begin{cases} \text{true} & \text{for } t \in [2,7) \\ \text{true} & \text{for } t \in (9,13] \\ \text{false} & \text{otherwise} \end{cases}$$

In order to mark an edge $e = (u, v)$ as present in T_x, we need to ensure that both u and v are present for the entire T_x.

This definition supports changes in node/edge characteristics at any time, whereas timesliced dynamic graphs allow changes only to occur in timeslices. DynNoSlice implements the above model as a collection of piecewise, linear functions defined on intervals in the space-time cube. Fast access to these functions at any given time is required. In our implementation, we use interval trees. When the intervals are guaranteed to be non-overlapping, simpler structures such as binary search trees can be used.

3.1 Time to Space Conversion

In the space-time cube, the time dimension is defined in units that are not homogeneous with the space dimensions. It is therefore necessary to define a conversion factor, τ, that transforms a unit in the time coordinates to τ units of space. We also define an ideal distance between two nodes (polylines) δ. The choice of τ and δ can substantially affect the performance of the drawing algorithm, as shown in Fig. 3. Consider the points $a = (0, 0)$ and $b = (0, 5)$ at time 0, and the point $c = (1, 0)$ at time 1. Consider two conversion factors $\tau_1 = 1$ and $\tau_2 = 10$ keeping δ constant. In the first case, the points in the space-time cube will be $a = (0, 0, 0)$, $b = (0, 5, 0)$, and $c = (1, 0, 1)$. A hypothetical repulsive force exerted on a by c will be stronger than that exerted by b, since the distances $\overline{ac}$ and $\overline{ab}$ are respectively $\sqrt{2}$ and 5. In the second case, the points will be $a = (0, 0, 0)$, $b = (0, 5, 0)$, and $c = (1, 0, 10)$. In this case the situation is reversed, since $\overline{ac} \approx 10$ and $\overline{ab} = 5$.

The choice of τ depends on the original time units (milliseconds, seconds, years, etc.) and desired effect of time on the spatial dimensions. Smaller τ results in a dense cube, where time will heavily influence node position in 2D, while larger τ leads to a sparse cube, where spacial positioning is less dependent on the time dimension.

As a rule of thumb, τ should be selected such that it is δ times the average rate of events in the space-time cube. For example, if the time unit is years and events appear on average once every two months $\tau = (12/2) * \delta$. If $\delta = 5$, the value for τ is set to 30. Using this estimate, τ can be automatically selected based on a given δ.

4 DynNoSlice Implementation

An event-based dynamic graph D can be transformed into a 3D static graph D' by embedding it in the space-time cube. Algorithms for static or timesliced dynamic graphs can be extended to work with this new representation. In this section, we describe our force-directed algorithm for drawing event-based dynamic graphs. It has been implemented and the source code is available[2].

4.1 Representation in the Space-Time Cube

We can define a space-time cube transformation (STCT) that transforms an event-based dynamic graph D into a 3D static graph drawing D' in the space-time cube, as follows. In D', the presence and position of each node is represented by a sequence of trajectories. The shapes of these trajectories are defined by the position attributes. In the example above, the trajectory of v is defined by three line segments, $(1, 0, 9) \to (4, 0, 12)$, $(2, -2, 12) \to (5, 1, 15)$ and $(5, 1, 17) \to (4, 5, 19)$, and by portions of the line $(0, 0, x)$, as shown in Fig. 4a.

The number of these trajectories is also affected by the appearance attribute. In the example above, the node v appears two times: at $[2, 7)$ and at $(9, 13]$. Clearly, the behaviour of the node at times when it is not part of the graph is non-influential. Therefore, the node appearance and position in the space-time cube can be identified by the segments $s_{v,1} = (0, 0, 2) \to (0, 0, 7)$, $s_{v,2} = (1, 0, 9) \to (4, 0, 12)$ and $s_{v,3} = (2, -2, 12) \to (3, -1, 13)$, as shown in Fig. 4b. The trajectory given by the polyline is determined by the start and end points, as well as the bends (the junctions of consecutive segments).

The representation of edges in the space-time cube is less intuitive. By connecting two trajectories with lines in the space-time cube, we obtain a ruled surface. Therefore, an edge $e = (u, v)$ is a surface that connects trajectories u and v for a duration indicated by $\mathcal{A}_e$. If the node trajectories are not continuous as in the above example:

$$\lim_{t \to 12^-} \mathcal{P}_v(t) = (4, 0) \qquad \text{and} \qquad \lim_{t \to 12^+} \mathcal{P}_v(t) = (2, -2),$$

an edge might create two or more surfaces, as shown in Fig. 4c.

If the trajectory segments, considered as vectors, form an acute angle with respect to the positive time axis, this transformation is easily invertible. Thus, trajectory segments cannot fold back on themselves, as that would identify multiple positions for that node at a given time. We can then work on this event-based dynamic graph in 3D as if it were a static graph as follows:

$$D_a \to \text{STCT} \to D'_a \to \mathit{Op.} \to D'_b \to \text{STCT}^{-1} \to D_b$$

2. http://cs.swan.ac.uk/~dynnoslice/software.html

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2886901, IEEE Transactions on Visualization and Computer Graphics

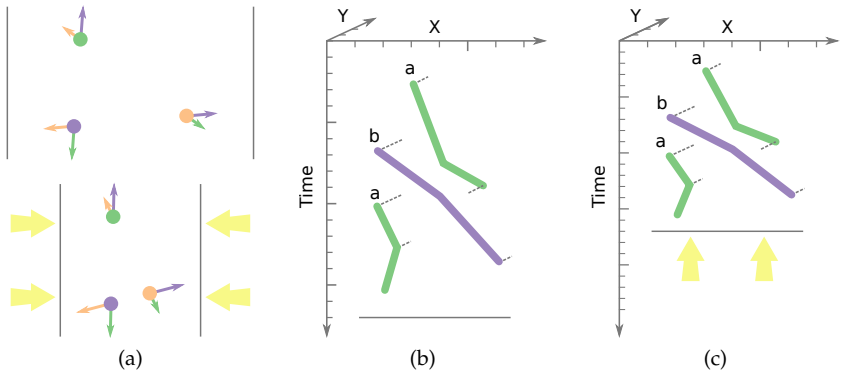


Fig. 3. Selecting an appropriate time dimension scaling factor τ. **(a)** By compressing the horizontal dimension, the direction and balance of forces changes, particularly evident for roughly horizontally aligned elements, such as the purple and orange nodes. **(b)** A starting space-time cube with two node trajectories. In this case, the time dimension is different from the two spacial dimensions; therefore a factor τ that transforms time units in space units is needed. **(c)** The space-time cube obtained using a smaller value for τ: the relative distance between node trajectories, and therefore the magnitude of the forces, is affected by the choice of τ.

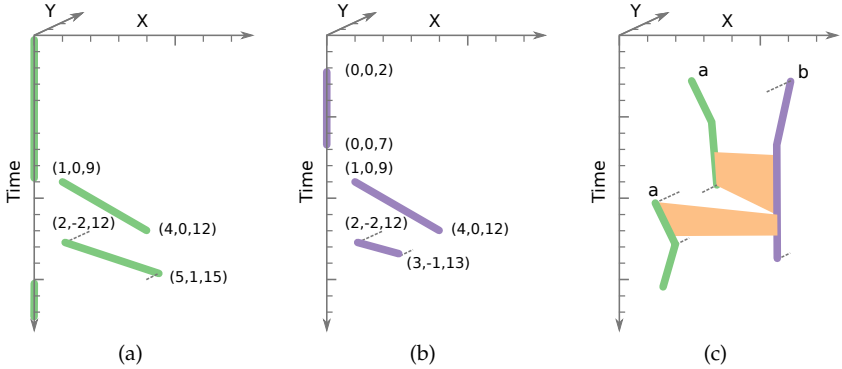


Fig. 4. A event-based dynamic graph in the space-time cube. (a) Position attribute for a node v. Piecewise linear functions encode node position across time. (b) The appearance attribute. Node v only appears over some intervals of time. (c) Edges connect node trajectories as encoded by the edge appearance attribute. The edge traces one or more ruled surfaces in the space-time cube.

4.2 Force-Directed Drawing Algorithm

Our force-directed algorithm is based on earlier variants by Simonetto *et al.* [61], [63]. As in most force-directed algorithms, after an initialisation phase, the algorithm iteratively improves the current layout of the drawing in 3D for a given number of iterations. Each iteration of DynNoSlice performs the following actions:

- Compute and sum the forces.
- Move nodes based on the forces and constraints.
- Adjust trajectory complexity in the space-time cube.

The complexity of DynNoSlice is in worst case $O(b^2 + e)$ per iteration for b trajectory bends and e edges in the dynamic graph, with the attractive and repulsive forces dominating the complexity. As the number of bends and edges is often proportional to the number of events in the event-based dynamic graph, the approach can be considered quadratic in the number of events. This worst case occurs when most of the bends are in the same dense region of the space time cube.

4.2.1 Initialisation

Each node v is randomly assigned a position (a, b) in the 2D plane. The points defining the trajectory of v are extruded linearly along the time axis (a, b, x), where x is the time coordinate.

4.2.2 Forces

DynNoSlice has five forces. The first three forces adapt standard, force-directed approaches to work with trajectories in the space-time cube. The final two are new forces needed for event-based dynamic graph drawing. In our notation, a star transforms 3D vector to 2D by dropping the time coordinate. For example, if $p = (1, 2, 3)$ then p^* is $(1, 2)$. The parameters δ and τ are as defined in section 3.1.

1. Node Repulsion. This force repels trajectories from each other. The force evenly distributes node trajectories in space and prevents crowding [32], [49].

For each segment endpoint a in the trajectory of node u and segment $s_{v,j} = c \rightarrow d$ of node v, with $u \neq v$, we compute the forces generated on the points a, c and d (see Fig. 5). If the points a, c and d are not collinear and ($p \in s_{v,j}$), they form a plane. In this case, we apply the force EdgeNodeRepulsion(δ) described in previous work [63], except node positions are in 3D. If the points are collinear or the projection p of a does not fall in the segment $s_{v,j}$ ($p \notin s_{v,j}$), we apply NodeNodeRepulsion(δ) [63] between a and c and between a and d.

Since distant segments do not interact significantly, they can be ignored to reduce the running time. A multi-level interval tree is used to identify segments that are sufficiently close ($< 5\delta$). All other pairs are ignored.

Given b bends in the node trajectories, the complexity of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2886901, IEEE Transactions on Visualization and Computer Graphics

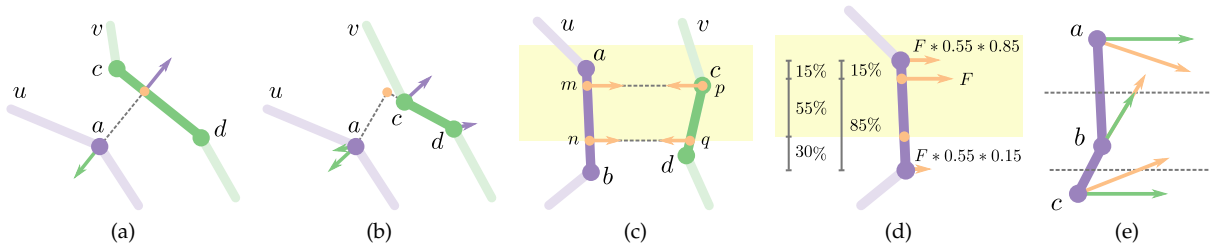


Fig. 5. Forces and constraints. **(a)** and **(b)** Node repulsion force between point a and the line segment $c \rightarrow d$ that represents the trajectories of nodes u and v. **(c)** and **(d)** Edge attraction for an edge in the interval highlighted with yellow background. **(e)** Time movement restriction. Endpoints (a and c) must keep their assigned time coordinates. Bend b cannot move past half the distance with other bends or endpoints.

the repulsive force could be as high as $O(b^2)$ if all bends are in the same area of the space-time cube. However, trajectories generally have a more uniform distribution. The number of bends is often proportional to the number of events present in the event-based dynamic graph.

2. Edge Attraction. This attractive force pulls trajectories that are linked by an edge closer to each other. The force is exerted only for the intervals where the edge is present.

Let us consider an edge $e = (u, v)$ that appears at interval (t_1, t_2) and let $I = (t_1\tau, t_2\tau)$ be the transformed time interval in the space-time cube with conversion factor τ that transforms time into the third dimension of the space time cube. For each pair of segments $s_{u,i} = a \rightarrow b$ and $s_{v,j} = c \rightarrow d$ that overlap with the interval (f, g), we compute the points m, n of segment $s_{u,i}$ and p, q of segment $s_{v,j}$ so that:

$$\min \{f \in I : \exists m \in s_{u,i},\ \exists p \in s_{v,j},\ f = m[2] = p[2]\}$$

$$\max \{g \in I : \exists n \in s_{u,i},\ \exists q \in s_{v,j},\ g = n[2] = q[2]\}$$

where $x[2]$ is the time coordinate of the point x in the space-time cube. We compute the attractive force between the points m and p, and n and q, using EdgeContraction(δ) [63] (see Fig. 5c). This force is applied to the segment endpoints once scaled by its distance from the application point and by the coverage of the edge appearance on the segment (see Fig. 5d). For example, if the force F_m attracts m to p, then F_a applied to a will be:

$$F_a = F_m * \frac{a[2] - m[2]}{a[2] - b[2]} * \frac{n[2] - m[2]}{a[2] - b[2]}.$$

Given e edges, the complexity is $O(e)$. Often, edges are the events and e is simply the event count.

3. Gravity. This force encourages a compact drawing of node trajectories. Let c be the centre in 2D of the initial node placement in the space-time cube. The gravity F of each segment endpoint a is $F = c^* - a^*$. Given n node trajectories, this is $O(n)$.

4. Trajectory Straightening. This force smooths node trajectories, helping with node movements over time. For trajectory bends, we use the CurveSmoothing [63] force, which pulls a bend b to the centroid of the triangle Δabc formed by using the previous and next bends or endpoints a and c. A trajectory endpoint a has no such triangle. Therefore, it is pulled in 2D toward the midpoint of the segment formed with the closest bend or endpoint b. As each bend is considered at most twice, its complexity is $O(b)$.

5. Mental Map Preservation. This force prevents trajectory segments from making large angles with respect to time, encouraging nodes in the drawing to remain stable during graph evolution [5]. When segments form a $90°$ angle with time, a node essentially "teleports" from one place to another, while segments parallel to the time axis result in no node movement. Thus, segments should form small angles with the time axis. This force pulls endpoints a and b of each trajectory towards each other in 2D with a magnitude based on the angle α the segment makes with the time axis:

$$F_a = (b^* - a^*) * \frac{\alpha}{90° - \alpha}$$

As each bend is considered at most once, its complexity is $O(b)$.

4.2.3 Constraints

Node movement constraints ensure valid drawing of the event-based dynamic graph in the space-time cube. In particular, constraints are needed to prevent undesired movements. As all of these constraints consider the movement of a bend relative to the one that comes before/after it, they can be enforced in $O(b)$ time.

Decreasing Max Movement. We insert a constraint on the maximum node movement allowed at each iteration. This allows large movements at the beginning of the computation, and smaller refinements towards the end. This constraint is similar to DecreasingMaxMovement(δ) [63].

Movement Acceleration. This constraint promotes consistent movements with previous iterations and penalises movements in the opposite direction. This constraint corresponds to MovementAcceleration(δ) [63].

Time Correctness. This constraint prevents a node from changing its time coordinate. Consider the trajectory formed by the segment $t = a \rightarrow b$. If a changes its time coordinate in the space-time cube, the time of its appearance will also change. Consider a trajectory formed by several segments, $t = a \rightarrow b \rightarrow c \rightarrow d$. The trajectory endpoints a and d, corresponding to the appearance and disappearance of the node, should have fixed time positions. However, bends b and c can move in time, so long as they do not pass each other ($a[2] < b[2] < c[2] < d[2]$) as this would result in a node being in two locations at once. Therefore, the movement M_a of node a in time is constrained to be:

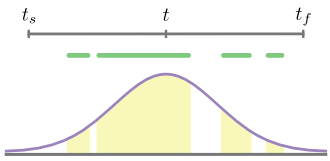


Fig. 6. Node (edge) transparency when aggregating a small multiple across time. We consider a Gaussian curve centred at t and with standard deviation equal to a sixth of the cluster interval in time. The appearance of the graph element in time is extracted (green lines). The transparency of the element is the area under the Gaussian during the appearance intervals (yellow) divided by the total area under the curve.

$M_a \leftarrow M_a^*$ if a is the endpoint of a trajectory, and

$$M_b \leftarrow M_b * \max \left\{ \begin{array}{l} r \in [0,1] : \dfrac{M_a[2] - M_b[2]}{2} < r M_b[2] \\ < \dfrac{M_c[2] - M_b[2]}{2} \end{array} \right\}$$

if b is a trajectory bend between other bends or endpoints a and c (see Fig. 5e).

4.2.4 Complexity Adjustment of Node Trajectories

As bends move freely in time and space, node trajectories can be oversampled or undersampled. Therefore, bends can be inserted or removed from the polyline representing a node trajectory. If a segment of the trajectory between bends a and b is greater than a threshold (2δ in this paper), a bend is inserted at its midpoint. Similarly, if two consecutive segments ab and bc are placed such that the distance between a and c is less than a threshold (1.5δ in this paper), the bend b is removed and the two segments are replaced by ac.

5 Small Multiple View Support

Executing DynNoSlice produces node trajectories in the space-time cube that encode the event-based dynamic graph evolution over time. Similarly, the surfaces that define the edges are also well defined in 3D. The most natural way to visualise this data set in two dimensions is through an animation where a plane passes through the space-time cube from top to bottom.

Animated visualisations of long dynamic graph series are often not effective [64]. In particular, such animations require the viewer to rely on memory to compare events that happen at the beginning of the animation to events that happen at the end of the animation. A number of studies have demonstrated that a small multiples visualisation of dynamic graphs can be more effective [6], [30] and therefore, we present a method for computing a set of small multiples to visualise the contents of the space-time cube.

The problem of selecting timeslices is a difficult one. Papers have examined how to select timeslices that preserve underlying stream properties and it is arguable that there is no way to choose relevant timeslices due to dynamics occurring at many different temporal resolutions [18], [48]. In a visualisation setting, there are perceptual advantages of considering a small multiples representation of a network and therefore we should support a small multiples representation of the drawing in the space-time cube. In our approach, we consider screen space constraints and select a small number of regions in the space-time cube where the trajectories of nodes change often.

Given a graph G with a computed drawing in the space-time cube (2D + t), we aim to select a positive number of k regions of the space time cube and based on these timestamps render a meaningful representation of the graph evolution. The value of k is determined by the number of small multiples windows that will fit on the display device. The goal is to summarise the full contents of the space-time cube by minimising the distance to all bends. In order to select these regions, we consider two measures for distance.

The first measure divides the space-time cube into k temporally consecutive slabs whereby the projection error to the k planes defining them is minimised. Let p_i with $i = 1 \ldots b$ be the position of each bend in the space-time cube with t_i as its time coordinate. Let t_{μ_j} with $j = 1 \ldots k$ the k planes in the space time cube perpendicular to the time axis. We apply k-means to these planes, assigning all bends to exactly one plane, minimising $||t_i - t_{\mu_j}||$. The extent of the slab is defined by the bend most distant to each side of the plane that has been assigned to it.

In the second, we apply classic k-means to divide the space-time cube into k regions. Let μ_j with $j = 1 \ldots k$ be the k centres placed in the space time cube. We apply classic k-means, assigning all bends to exactly one centre, minimising $||p_i - \mu_i||$. This measure divides the space time cube into k three-dimensional regions defined by the bends assigned to each centre. The projection plane is the time axis aligned plane that contains μ_j.

The small multiples are created by projecting the nodes down onto a plane t. If a polyline representing a node intersects the projection plane, its position is defined by the point of intersection with t. For for polylines that are present in the region that do not intersect t, the average position of the last appearance before t and the first appearance after t is taken. If either is appearance is missing (e.g., a node no longer appears in the data set or it appears for the first time), it is simply assigned the existing position.

For both nodes and edges, we compute a transparency value that indicates how close it is to t. Let $I = [t_s, t_f]$ be the interval of the space time cube that represents the union of all elements assigned to that cluster by k-means. We consider a Gaussian centred at t with standard deviation σ equal to a sixth of the interval $t_f - t_s$. Given a graph node or edge, we compute the sum of the area under this curve for which the element was present (see Fig. 6). This area is divided by the total area of the curve and is used as an alpha channel. This computation gives higher importance to longer appearances close to t, but factors in all other appearances. The transparency value can be efficiently computed with a numerical approximation of the cumulative distribution function (CDT), which reports the normalised area under a Gaussian for a given value. For each appearance $[t_1, t_2]$, we add a contribution equal to $\text{cdt}(t_2) - \text{cdt}(t_1)$.

An example of five clusters found by this method are Fig. 7 (3D distance used). This data set, discussed in detail in Section 6.1, consists of tweets between teams in the Guinness Pro12 rugby competition. A node is a team and an edge connects two nodes at the precise moment a tweet occurred.

6 METRIC EVALUATION

To evaluate the effectiveness of DynNoSlice, we use a metrics-based experiment that measures stress, movement, and crowding of the dynamic graph and tests both timesliced and event-based data.

6.1 Data Set Construction

In the evaluation of DynNoSlice, a number of data sets are considered, some of which are event-based data sets while others are timesliced data sets.

Infovis Co-Authorship (Timesliced). This is the co-authorship network for papers published in the InfoVis conference from 1995 to 2015 [1]. Authors that collaborated on a paper appear as a clique at the time of publication of the paper. Nodes disappear in years where the author does not publish a paper. The data is of discrete nature with exactly 21 timeslices (one per year).

VanDeBunt (Timesliced). The van De Bunt data set contains the relationships between 32 freshmen at seven different time points. At each point in time, each participant was asked to rate the friendship into one of several categories spanning "best friendship" to "troubled relation". Additional attributes such as gender, smoker/non-smoker, and program duration are also provided. We build a discrete dynamic graph using the method of Brandes and Mader [15]. An undirected edge is inserted into a timeslice if the participants reciprocally reported their friendship as either "best friendship" or "friendship" at that time.

Newcomb Fraternity Data (Timesliced). The Newcomb fraternity data set contains the sociometric preference of 17 members of a fraternity in the University of Michigan in the fall of 1956 [55]. Each participant was asked to order the other students from their closest to their least close friend. This data was collected weekly for about 15 weeks. As in previous work [15], at each timeslice, we inserted undirected edges connecting students to their three best friends.

Rugby (Event-Based). The data set consists of 3151 tweets posted between 01.09.2014 and 23.10.2015 from the teams participating in the Guinness Pro12 rugby competition. We build an event-based graph by assigning each team a node which remains constantly present in the drawing. An edge is placed between two teams at the precise moment a tweet happens. For each tweet at time t between teams a and b, we define an appearance for the edge $e = (a, b)$ at the interval $[t - 12\text{h},\ t + 12\text{h}]$. Multiple edges between the same two teams whose appearance interval overlaps or is smaller than one day ($t_2 - t_1 < 24\text{h}$) are merged into a single edge with appearance interval $[t_1 - 12\text{h},\ t_2 + 12\text{h}]$. Note that discretisation with a similar precision (daily) would create 417 slices. In the timesliced version of the data set, we flattened the cube into 20 slices ($s = 20$).

Pride & Prejudice (Event-Based). This data set [39] is a list of all 4033 character dialogues in the order each appears in the novel *Pride & Prejudice*. We build an event-based dynamic graph by assigning a node to each character, and by setting the appearance interval of an edge between two characters when they are involved in a dialogue. Nodes appear when they are involved in their first dialogue and remain for the duration of the data set. We assign a time to each dialogue defined by two components, $t = r + s$. The integer component r is the chapter number. The fractional component s position of the dialogue over the total number of dialogues for the chapter. For example, the 4th out of 20 dialogues in chapter 5 has $t_{5,4} = 5.2$. For each dialogue at time $t_{i,j}$, the corresponding edge is marked as present for the interval $[t_{i,j-3}, t_{i,j+3}]$, merging eventual overlapping appearances. The incident nodes are marked as present for a slightly larger interval $[t_{i,j-5}, t_{i,j+5}]$, again merging overlapping appearances into a single appearance. We create the timesliced version of the data set by creating a timeslice for each of the 61 chapters ($s = 61$).

6.2 Method

As there are no event-based dynamic graph drawing algorithms in related work, we compare our results with timesliced dynamic graph drawing algorithms. In our metric evaluation, we considered four approaches:

`VisoneA` drawings were computed using the aggregated layout with `visone` [16]. It is the simplest strategy where all nodes and edges for the timespan of the dynamic graph are collapsed down into a single graph, known as the supergraph, which is drawn once, minimising global stress.

`VisoneL` drawings were computed using the timesliced version of `visone` [16] and a linking strategy, which performed well in previous studies [15], with default link length of 200 and stability parameter $\alpha = 0.5$.

`DynNoSlice` drawings were computed using DynNoSlice and $\delta = 1$ as the desired edge length.

`DynNoSliceT` drawings were computed using a modified version of DynNoSlice that forces timeslices on the otherwise continuous DynNoSlice algorithm. Each trajectory bend coincides with a timeslice and bends cannot be inserted or removed. The rest of the algorithm is unaltered. We used $\delta = 1$ as the edge length parameter.

A number of metrics could have been selected in order to evaluate the effectiveness of event-based dynamic graph drawing. However, to allow for a comparison between algorithms and previous experiments that use timeslices, we use methods that have been previously used in experiments to evaluate such approaches.

Stress is a measurement of drawing quality used in many experiments [15], [37], [45], [56]. It measures how node position reflects the shortest path distance which is a important factor for graph readability as the Euclidean distance is often used to judge how close two nodes are in a graph [29]. As stress is defined for a static graph, we slice the space-time cube and average the stress computed on each timeslice.

Node Movement is the average 2D movements of the nodes. Intuitively, this is the average distance travelled by nodes when animating the dynamic graph. This metric has been used in important previous experiments [15] and is a measurement drawing stability – an important factor in identifying particular nodes and paths over time [4], [5].

Crowding counts the number of times nodes pass very close to each other in an animation of the dynamic graph. Crowding has adverse effects on identifying nodes [4], [5] and is know to negatively influence object tracking [32], [49]. When measuring crowding for `VisoneA`, if two nodes overlap for the duration of the dynamic graph, it is counted as a single crowding event.

To compare event-based and timesliced dynamic graphs, we define four types of stress. *StressOn(d)* is computed on the timeslices using the node and edge set of that timeslice. *StressOff(d)* is computed on and between the default timeslices using the node and edge set of the closest timeslice in time, when between two timeslices. *StressOn(c)* is computed on the timeslices using the precise node and edge appearances in continuous time. *StressOff(c)* is computed on and between the timeslices using the precise node and edge appearances in continuous time.

6.2.1 Graph Scaling

Uniformly scaling node positions changes the measure of stress even though the layout is the same [37], [45], [56]. In order to compare methods as fairly as possible, we used a strategy where scale-independent values of stress are compared as follows. Our algorithms have a parameter that indicates the desired edge length. First, we verified that `VisoneA` and `VisoneL` produce the same result (up to scale) when changing the edge length parameter. Thus, we use the default value of edge length but consider different scaling factors to compare to the output of our algorithm. For the experiment, we defined nodes to be circles of diameter 0.2 and with an ideal edge length of 1. To obtain such drawing with our approach, we run the algorithm with an ideal edge length parameter $\delta = 1$. To obtain such drawing with `VisoneA` and `VisoneL`, we run it with the default edge length of 200 and scale it down by a factor 200.

Related work in static graph drawing [37], [45], [56] searches for the best scaling factor via binary search, as a minimum is guaranteed. For our metric (average stress across all timeslices) we have no such guarantee. Thus, we evaluate scaling factors $(1.1)^i$, with $i \in \mathbb{Z} : -20 < i < 20$ for the best StressOn(d) value. This scaling factor is used to compute all metrics. After plotting the average stress for each data set, a minimum was consistently observed.

6.3 Results

Videos of *Newcomb*, *Rugby*, and *Pride & Prejudice* are available[3]. For our experiments, we run each algorithm ten times and report the average result.

Table 1 shows the results on the timesliced data sets. In the VanDeBunt and Newcomb data sets, `VisoneL` outperforms DynNoSlice in terms of stress. Crowding is comparable for all the algorithms. `VisoneA` has zero movement, and all other approaches are competitive in terms of node movement. Our event-based approach is sometimes able to improve on `VisoneL` when stress is computed off timeslices. When comparing `DynNoSliceT` to DynNoSlice, `DynNoSliceT` is often able to optimise on-timeslice stress. InfoVis is an outlier for the timesliced data sets as DynNoSlice outperforms both `visone` approaches in terms of stress. `VisoneA` is competitive or the worst in terms of stress in all cases.

Table 2 shows the results of our metric experiment on the event-based data sets. DynNoSlice has competitive or lower off-timeslice stress, average movement, crowding events, and occasionally lower on-timeslice stress. `VisoneA`, by definition, has zero movement. On *Pride & Prejudice*, all three approaches result in similar stress. `VisoneA` performs better than `VisoneL` in terms of stress, and is competitive with DynNoSlice on *Rugby*.

3. http://cs.swan.ac.uk/~dynnoslice/files/video.mp4

6.4 Discussion

Our results on the timesliced data sets were expected: `VisoneL` optimises for stress directly on every timeslice and performs the best, while it is comparable in terms of movement and crowding. As a state-of-the-art algorithm for timesliced graph drawing, it is difficult to compete with the approach when it is running on the type of data for which it was designed. However, in terms of off timeslice stress, DynNoSlice can outperform `VisoneL`. The timesliced model does not allow for stress to be optimised between timeslices and must resort to linear interpolation, leading to suboptimal stress. In our event-based dynamic graph model, we optimise for stress in continuous time, leading to this performance improvement. One exception is InfoVis where DynNoSlice is able to improve on `VisoneL` in terms of stress. This result may be due to the bursty nature of this graph (edges are only present if two authors published a joint paper that year). Therefore, large parts of the graph change drastically from year to year. Allowing node trajectories to evolve independently of timeslices may allow DynNoSlice to perform better. Although `VisoneA` has no node movement and low crowding, it performs 10% worse in the best case in terms of stress.

For the event-based data sets, DynNoSlice outperforms `VisoneL` in terms of stress and crowding and is competitive with `VisoneA`. DynNoSlice does not use timeslices to compute the layout of the dynamic graph. As a result, the approach optimises stress between timeslices. On-timeslice stress is an exception, as it is directly optimised by `VisoneL`.

DynNoSlice simultaneously improves node movement and crowding while remaining competitive or improving on stress when compared to `VisoneL`. This finding may seem counter-intuitive as low stress usually corresponds to high node movement. The result can be explained by the fact that nodes in DynNoSlice are polylines of adaptive complexity in the space-time cube. Nodes with few interactions will be long straight lines, potentially passing through many timeslices. These areas of low complexity will reduce average node movement. In a model that uses timeslices, each timeslice is forced to have the node with inter-timeslice edges. Therefore, timeslices impose additional node movement that may not be necessary.

DynNoSlice allows the polyline representing a node to adapt its complexity between timeslices if there are many interactions. When there are many changes to the graph in a short period of time, these polylines have increased complexity, allowing nodes to avoid crowding. In a timesliced model, only linear interpolation is possible, and all nodes must follow straight lines. Thus, crowding is incurred. Crowding is also avoided in DynNoSlice as our polylines have repulsive forces between them.

7 Case Studies

We present case studies showing the results of *Rugby* and *Pride & Prejudice*. The images have been created using the second small multiples technique described in Section 5.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2886901, IEEE Transactions on Visualization and Computer Graphics

TABLE 1
Results for the timesliced data sets on our metrics. Results are reported for an average of ten runs with different random seeds. All metrics are as defined in Section 6.2. As the data is timesliced, only timesliced edge sets can be measured as event-based resolution is not available.

Graph	Type	Time (s)	Scale	StressOn (d)	StressOff (d)	Movement	Crowding
VanDeBunt	VisoneA	0.04	0.75	1.75	1.76	0.00	0.00
	VisoneL	0.13	1.00	1.14	1.46	3.80	0.00
	DynNoSliceT	7.86	0.60	1.52	1.55	3.73	0.00
	DynNoSlice	7.11	0.61	1.56	1.71	3.20	0.20
Newcomb	VisoneA	0.03	0.83	20.07	19.87	0.00	0.00
	VisoneL	0.11	1.00	14.04	14.77	16.36	8.00
	DynNoSliceT	10.91	0.68	17.43	17.36	13.72	1.30
	DynNoSlice	7.87	0.75	17.93	17.89	12.41	1.10
InfoVis	VisoneA	5.90	0.32	57.00	56.05	0.00	34.00
	VisoneL	77.43	0.47	51.66	52.98	2.15	36.00
	DynNoSliceT	356.40	0.56	28.10	27.91	2.00	3.30
	DynNoSlice	379.46	0.56	29.44	30.77	1.88	4.30

TABLE 2
Results for the event-based data sets on our metrics. Results are reported for an average of ten runs with different random seeds. All metrics are as defined in Section 6.2. As the data is event-based, we measure stress on the event-based set of edges to make the algorithms comparable. # Events is the number of events in the data set. N_{SG} is the number of nodes in the supergraph. E_{SG} is the number of edges in the supergraph.

Graph	Type	# Events	N_{SG}	E_{SG}	Time (s)	Scale	StressOn (c)	StressOff (c)	Movement	Crowding
Rugby	VisoneA	3151	12	66	0.01	0.56	2.21	1.96	0.00	0.00
	VisoneL				0.08	0.68	3.08	2.71	25.47	6.00
	DynNoSliceT				8.28	0.68	1.84	1.75	16.35	0.10
	DynNoSlice				4.35	0.50	1.94	1.80	6.68	0.00
Pride & Prejudice	VisoneA	4033	118	501	0.83	0.42	0.70	0.86	0.00	11.00
	VisoneL				3.39	0.18	0.62	0.88	5.44	682.00
	DynNoSliceT				1754.63	0.31	0.70	0.84	6.70	11.90
	DynNoSlice				82.28	0.28	0.71	0.86	1.28	0.00

7.1 Rugby

Section 6.1 describes how we construct this data set from the tweets between teams of the Guinness Pro12 rugby competition. The event-based dynamic graph was drawn in the space-time cube using DynNoSlice and the small multiples technique with $k = 5$ centres. Fig. 7 shows the results of this partition of the space-time cube.

Selecting a set of timeslices for this data set is hard as events occur at multiple temporal resolutions in the data set. During the season, there are many tweets at a high temporal resolution while during the off season there is a small number of tweets and a low temporal resolution. This competition consists of teams from four nations: Wales, Scotland, Ireland, and Italy. It is immediately clear from the DynNoSlice small multiples representation that geography greatly influences the layout. In particular, teams originating from Wales (Dragons, Ospreys, Blues, and Scarlets) appear on the lower left, Scotland (Glasgow and Edinburgh) in the lower right, Ireland (Leinster, Munster, Connacht, Ulster) at the top, and Italy (Benetton and Zebre) in the centre-right. In the version computed by VisoneL, this structure is not visible with node positions mixed throughout the drawing. As teams in the same region tend to tweet slightly more often about each other, this likely causes the trajectories of nodes from the same nation to evolve in the same area of the plane, making the feature appear at a high temporal resolution. A timesliced approach or one that minimises stress globally loses this temporal information through aggregation, causing high connectivity and a loss of this important temporal structure. VisoneA places the vertices better according to the nations, possibly due to the smaller size of this data set.

DynNoSlice further divides the last two timeslices into teams based in Ireland and teams based in the United Kingdom with Italian teams split over the two clusters. During this time, regional discussion likely dominates the data set as there are no fixtures between teams in the summer, causing this split along national lines. This feature in the data set is at a lower temporal frequency than the rest of the data and is not easily recovered using uniform timeslicing. As there are both high and low frequency features in this data set, it is very difficult to select uniform timeslices. The temporal structure is not visible in VisoneA and VisoneL.

7.2 Pride & Prejudice

Section 6.1 describes how we construct the dynamic graph from the character interactions in the novel *Pride & Prejudice*. The event-based dynamic graph was drawn in the space-time cube using DynNoSlice and the small multiples technique with $k = 5$ centres.

We asked an expert in literary social network analysis (and one of the co-creators of this data set [39]) with significant experience in studying novels in general and this novel in particular [40] to help interpret our results.

This data set has very high frequency features as it consists of all character dialogues in the book from its start to its end. The literary analyst noted that the position of the nodes in the DynNoSlice layout (Fig. 8) conveys important information about central characters in the novel. She noted that characters closer to the centre of the drawing are main characters. As these characters interact with

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2886901, IEEE Transactions on Visualization and Computer Graphics

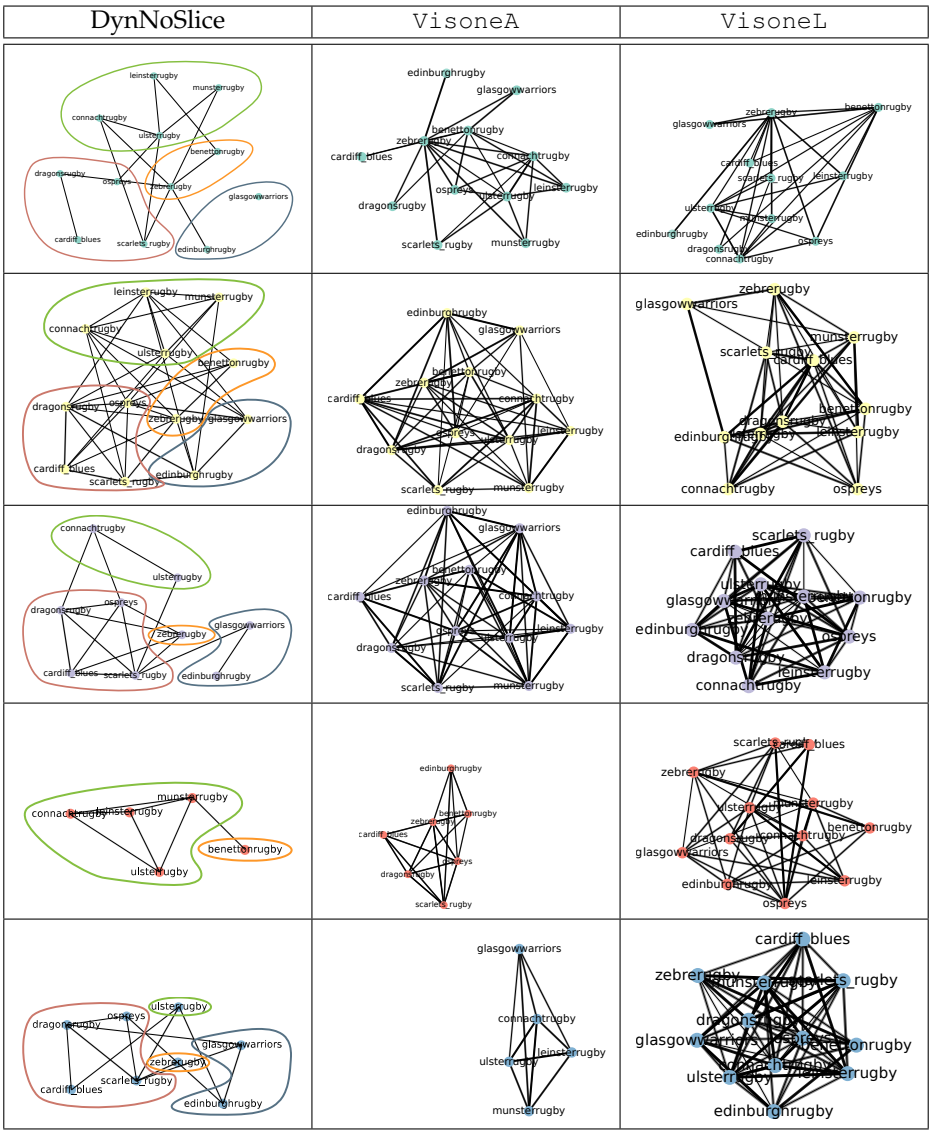


Fig. 7. *Rugby* drawn with DynNoSlice, VisoneA, and VisoneL. Five space time clusters have been subsequently extracted using the same k-means process described in Section 5. Manual scaling and alignment was used for the outputs of VisoneA in a post-processing step for a fair comparison with the remaining approaches. The four participating nations are visible in the layout as within-nation tweets occur with greater frequency (annotation in first timeslice: Wales - red, Ireland - green, Scotland - blue, and Italy - orange).

greater frequency with a large number of characters in the novel, they are pulled to the centre of the drawing leaving more minor characters at the periphery. This high frequency temporal information is lost due to temporal aggregation in the VisoneL drawings (Fig. 9). VisoneA (Fig. 10) is similar to VisoneL, but crowding makes it difficult to make sense of the network. It preserves slightly more information than VisoneL at the expense of crowding. This crowding is incurred as all temporal information is collapsed down into a single graph, making finding a global optimum for stress difficult to compute for this larger supergraph.

The first and last clusters of the data set in DynNoSlice roughly correspond to the beginning and end of the novel but are primarily composed of the central characters with strong links between protagonists. Our literary analyst found the first cluster interesting as she personally believes the beginning of *Pride & Prejudice* to be quite stylistically different when compared to the rest of the novel as the novelist wrote an early draft fifteen years earlier before completing the story.

8 Conclusions and Future Work

We present a model for event-based, dynamic graphs and the first algorithm, DynNoSlice, that is able to draw dynamic graphs in this model. Our algorithm embeds the event-based dynamic graph within the space-time cube (2D+time) to overcome the limitations of timeslicing. The most natural way to visualise these graphs is through animation. However, animations can be perceptually ineffective. Thus, we present a method for extracting timeslices from the space-time cube using two variants of k-means. Our algorithm is evaluated through metrics-based comparisons and case studies.

Currently, DynNoSlice scales to thousands of events, but many of the data sets we would like to visualise are larger. As future work, we would like to explore possible ways

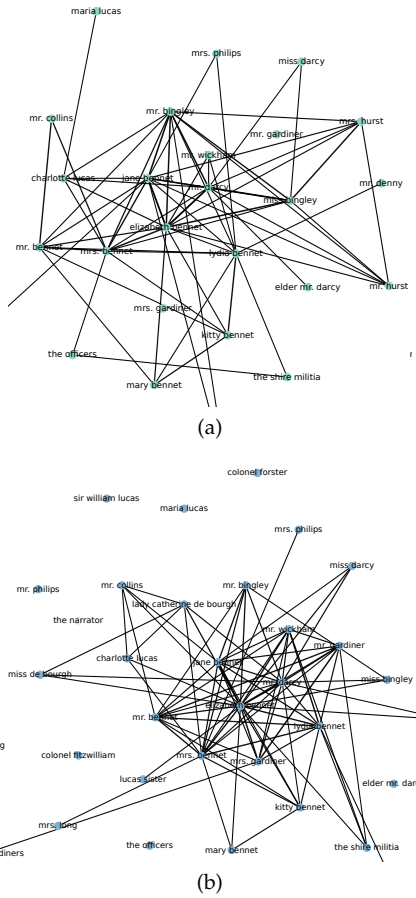


Fig. 8. *Pride & Prejudice* drawn with DynNoSlice centred around the main character *Elizabeth Bennet*. Five space time clusters have been subsequently extracted using the same k-means process described in Section 5. First and fifth cluster shown. Saturation of edges enhanced in all visualisations. Main characters of the novel appear at the centre of the drawing with minor characters at the edges. Character dialogues are at a high frequency (one edge effectively per timeslice) and DynNoSlice is able to exploit this information.

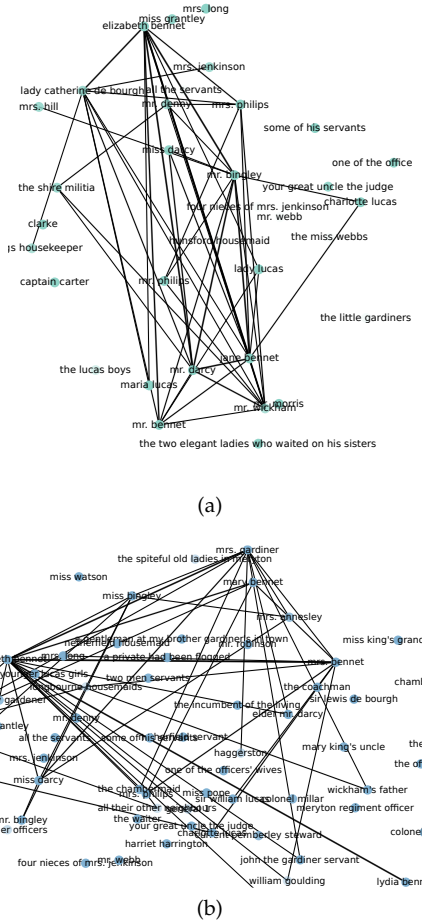


Fig. 9. *Pride & Prejudice* drawn with `VisoneL` near *Elizabeth Bennet*. Five space time clusters have been subsequently extracted using the same k-means process described in Section 5. First and fifth cluster shown. Saturation of edges enhanced in all visualisations. As the data is divided into timeslices and aggregated across time, temporal information is lost. Main characters do not appear central in the layout.

to make this approach more scalable with respect to the number of events in the dynamic graph. Multilevel techniques [11], [36], [42], [67] are a promising way to increase the scalability of event-based graph drawing. However, the design of effective coarsening operators needs to be explored and remains future work.

Small multiples often have significant advantages over animation in terms of user response time [6], [30]. The main advantage of static representations is that many periods of time can be visible onscreen simultaneously and do not rely on user memory for task completion. Small multiples are only one static representation of dynamic data. Other techniques can be adapted to visualising event-based dynamic graphs. Further study of effective representations of event-based dynamic graphs in the space time cube are necessary.

Selecting timeslices for dynamic data shares many things in common with sampling. To guarantee that no temporal features are missed through aggregation, we would need to sample at the Nyquist frequency, meaning that timeslices should be spaced at a distance of the smallest possible interval between two events. Sampling real, event-based data sets at this distance is usually prohibitively expensive. Future work should explore sampling methods for dynamic data that enable lower sampling rates in order to produce scalable visualisations of event-based data.

Acknowledgments

We would like to thank Peter Eades for suggesting *Event-Based Graph Drawing* as a more accurate name for this research at GD 2017. We would also like to thank Derek Greene and Karen Wade for providing the *Rugby* and *Pride & Prejudice* data sets. We would like to thank Karen Wade for her insights into the *Pride & Prejudice* network. Work on this project was supported by EPSRC First Grant EP/N005724/1 and NSF grants CCF-1423411 and CCF-1712119.

References

[1] Citevis citation datafile. http://www.cc.gatech.edu/gvu/ii/citevis/infovis-citation-data.txt. Accessed: 2016-11-19.

[2] A. Ahmed, T. Dwyer, C. Murray Le, S. Ying, and X. Wu. Infovis 2004 contest: Wilmascope graph visualisation. 2004. https://www.cs.umd.edu/hcil/InfovisRepository/contest-2004/1/unzip/standardform2004.html.

[3] D. Archambault, J. Abello, J. Kennedy, S. Kobourov, K.-L. Ma, S. Miksch, C. Muelder, and A. C. Telea. Temporal multivariate networks. In A. Kerren, H. C. Purchase, and M. O. Ward, editors, *Multivariate Network Visualization*, volume 8380, pages 151–174. Springer, 2014.

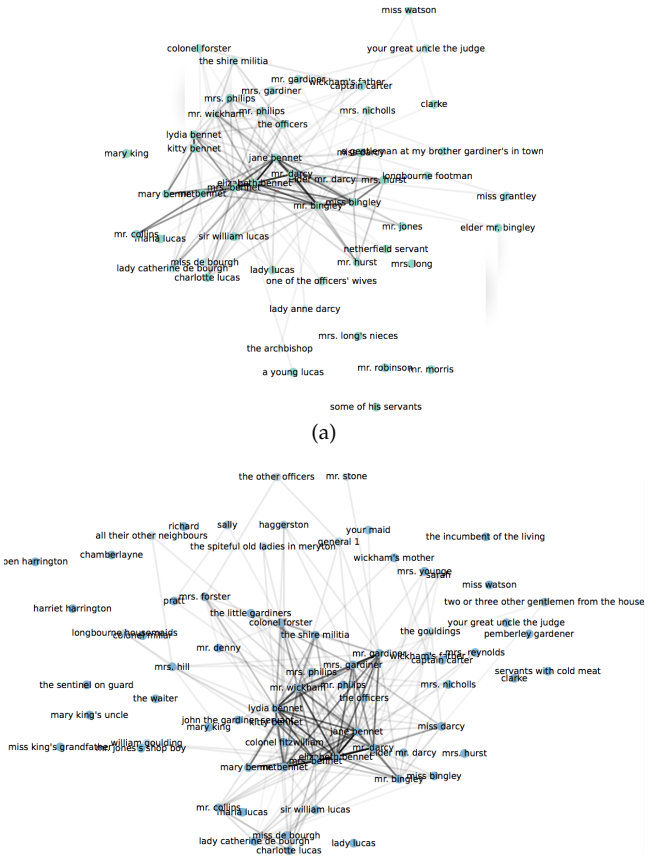


Fig. 10. *Pride & Prejudice* drawn with `VisoneA`, near *Elizabeth Bennet*. Five space time clusters have been subsequently extracted using the same k-means process described in Section 5. First and fifth cluster shown. As all information is aggregated into a single network, temporal information is lost and it is hard to find a global optimum for stress.

[4] D. Archambault and H. C. Purchase. Mental map preservation helps user orientation in dynamic graphs. In *Proc. of Graph Drawing (GD '12)*, pages 475–486, 2012.

[5] D. Archambault and H. C. Purchase. Can animation support the visualization of dynamic graphs? *Information Sciences*, 330:495–509, 2016.

[6] D. Archambault, H. C. Purchase, and B. Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Trans. on Visualization and Computer Graphics*, 17(4):539–552, 2011.

[7] B. Bach, P. Dragicevic, D. Archambault, C. Hurter, and S. Carpendale. A descriptive framework for temporal data visualizations based on generalized space-time cubes. *Computer Graphics Forum*, 36(6):36–61, 2017.

[8] B. Bach, E. Pietriga, and J. D. Fekete. GraphDiaries: Animated transitions and temporal navigation for dynamic networks. *IEEE Trans. on Visualization and Computer Graphics*, 20(5):740–754, 2014.

[9] B. Bach, E. Pietriga, and J.-D. Fekete. Visualizing dynamic networks with matrix cubes. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*, pages 877–886, 2014.

[10] B. Bach, C. Shi, N. Heulot, T. Madhyastha, T. Grabowski, and P. Dragicevic. Time curves: Folding time to visualize patterns of temporal evolution in data. *IEEE Trans. on Visualization and Computer Graphics*, 22(1):559–568, 2016.

[11] G. Bartel, C. Gutwenger, K. Klein, and P. Mutzel. An experimental evaluation of multilevel layout methods. In *Proc. of Graph Drawing (GD '10)*, volume 6502, pages 80–91. Springer, 2011.

[12] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. The state of the art in visualizing dynamic graphs. In R. Borgo, R. Maciejewski, and I. Viola, editors, *EuroVis - STARSs*. The Eurographics Association, 2014.

[13] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. A taxonomy and survey of dynamic graph visualization. *Computer Graphics Forum*, 36(1):133–159, 2017.

[14] U. Brandes and S. R. Corman. Visual unrolling of network evolution and the analysis of dynamic discourse. *Information Visualization*, 2(1):40–50, Mar. 2003.

[15] U. Brandes and M. Mader. A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In *Proc. of Graph Drawing (GD '11)*, volume 7034, pages 99–110. Springer, 2012.

[16] U. Brandes and D. Wagner. *Analysis and Visualization of Social Networks*, pages 321–340. Springer Berlin Heidelberg, 2004.

[17] M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf. Parallel edge splatting for scalable dynamic graph visualization. *IEEE Trans. on Visualization and Computer Graphics (InfoVis '11)*, 17(12):2344–2353, 2011.

[18] R. S. Caceres and T. Berger-Wolf. *Temporal Scale of Dynamic Networks*, pages 65–94. 2013.

[19] M. Cordeil, T. Dwyer, K. Klein, B. Laha, K. Marriott, and B. H. Thomas. Immersive collaborative analysis of network connectivity: CAVE-style or head-mounted display? *IEEE Trans. on Visualization and Computer Graphics (InfoVis '16)*, 23(1):441–450, 2017.

[20] T. Crnovrsanin, J. Chu, and K.-L. Ma. An incremental layout method for visualizing online dynamic graphs. In *Proc. of the International Symposium on Graph Drawing and Network Visualization (GD '15)*, volume 9411, pages 16–29. Springer, 2015.

[21] I. F. Cruz and J. P. Twarog. 3D graph drawing with simulated annealing. In *Proc. of Graph Drawing (GD95)*, volume 1027, pages 162–165. Springer, 1996.

[22] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Trans. on Graphics*, 15(4):301–331, Oct. 1996.

[23] S. Diehl and C. Görg. Graphs, they are changing — dynamic graph drawing for a sequence of graphs. In *Proc. of Graph Drawing (GD' 02)*, volume 2528, pages 23–31. Springer, 2002.

[24] S. Diehl, C. Görg, and A. Kerren. Preserving the mental map using foresighted layout. In *Eurographics/IEEE VGTC Symposium on Visualization (VisSym01)*, pages 175–184, 2001.

[25] F. Du, B. Shneiderman, C. Plaisant, S. Malik, and A. Perer. Coping with volume and variety in temporal event sequences: Strategies for sharpening analytic focus. *IEEE Trans. on Visualization and Computer Graphics*, 23(6):1636–1649, 2017.

[26] T. Dwyer and P. Eades. Visualising a fund manager flow graph with columns and worms. In *Proc. of the International Conference on Information Visualisation (IV '02)*, pages 147–152, 2002.

[27] C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. Yee. GraphAEL: Graph animations with evolving layouts. In *Proc. of Graph Drawing (GD '03)*, volume 2912, pages 98–110. Springer, 2004.

[28] C. Erten, S. G. Kobourov, V. Le, and A. Navabi. Simultaneous graph drawing: Layout algorithms and visualization schemes. In *Proc. of Graph Drawing (GD '03)*, volume 2912, pages 437–449. Springer, 2004.

[29] S. I. Fabrikant and D. R. Montello. The effect of instructions on distance and similarity judgements in information spatializations. *International Journal of Geographical Information Science*, 22(4):463–478, 2008.

[30] M. Farrugia and A. Quigley. Effective temporal graph layout: A comparative study of animation versus static display methods. *Journal of Information Visualization*, 10(1):47–64, 2011.

[31] D. Forrester, S. G. Kobourov, A. Navabi, K. Wampler, and G. Yee. Graphael: A system for generalized force-directed layouts. In *Proc. of Graph Drawing (GD '04)*, volume 3383, pages 454–464. Springer, 2004.

[32] S. L. Franconeri, J. Y. Lin, Z. W. Pylyshyn, B. Fisher, and J. T. Enns. Evidence against a speed limit in multiple-object tracking. *Psychonomic Bulletin & Review*, 15(4), 2008.

[33] A. Frick, A. Ludwig, and H. Mehldau. Fast interactive 3-D graph visualization. In *Proc. of Graph Drawing (GD '94)*, volume 894, pages 388–403. Springer, 1995.

[34] Y. Frishman and A. Tal. Online dynamic graph drawing. *IEEE Trans. on Visualization and Computer Graphics*, 14(4):727–740, July 2008.

[35] P. Gajer and S. G. Kobourov. A fast adaptive layout algorithm for undirected graphs. In *Proc. of Graph Drawing (GD '00)*, volume 1984, pages 222–228. Springer, 2001.

[36] P. Gajer and S. G. Kobourov. Grip: Graph drawing with intelligent placement. In *Proc. of Graph Drawing (GD '00)*, volume 1984, pages 222–228. Springer, 2001.

[37] E. R. Gansner, Y. Hu, and S. North. A maxent-stress model for graph layout. *IEEE Trans. on Visualization and Computer Graphics*, 19(6):927–940, 2013.

[38] T. E. Gorochowski, M. Di Bernardo, and C. S. Grierson. Using aging to visually uncover evolutionary processes on networks. *IEEE Trans. on Visualization and Computer Graphics*, 18(8):1343–1352, Aug. 2012.

[39] S. Grayson, K. Wade, G. Meaney, and D. Greene. The sense and sensibility of different sliding windows in constructing co-occurrence networks from literature. In *Computational History and Data-Driven Humanities (CHDDH16)*, pages 65–77, 2016.

[40] S. Grayson, K. Wade, G. Meaney, J. Rothwell, M. Mulvany, and D. Greene. Discovering structure in social networks of 19th century fiction. In *Proc. of the 8th ACM Conference on Web Science (WebSci '16)*, pages 325–326, 2016.

[41] G. Groh, H. Hanstein, and W. Wörndl. Interactively visualizing dynamic social networks with DySoN. In *Workshop on Visual Interfaces to the Social and the Semantic Web (VISSW2009)*, Feb. 2009.

[42] S. Hachul and M. Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Proc. of Graph Drawing (GD '04)*, volume 3383, pages 285–295. Springer, 2004.

[43] P. Holme and J. Saramki. Temporal networks. *Physics Reports*, 519(3):97 – 125, 2012.

[44] M. Itoh, N. Yoshinaga, M. Toyoda, and M. Kitsuregawa. Analysis and visualization of temporal changes in bloggers' activities and interests. In *IEEE Pacific Visualization Symposium (PacificVis '12)*, pages 57–64, 2012.

[45] S. G. Kobourov, S. Pupyrev, and B. Saket. Are crossings important for drawing large graphs? In *International Symposium on Graph Drawing (GD '14)*, pages 234–245. Springer Berlin Heidelberg, 2014.

[46] B. Landgraf. 3D graph drawing. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs*, volume 2025 of *Lecture Notes in Computer Science*, pages 172–192. Springer, Apr. 2001.

[47] M. Latapy, T. Viard, and C. Magnien. Stream graphs and link streams for the modeling of interactions over time, 2017. arXiv:1710.04073.

[48] Y. Léo, C. Crespelle, and E. Fleury. Non-altering time scales for aggregation of dynamic networks into series of graphs. In *Proc. of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, pages 29:1–29:7, 2015.

[49] G. Liu, E. L. Austen, K. S. Booth, B. D. Fisher, R. Argue, M. I. Rempel, and J. T. Enns. Multiple-object tracking is based on scene, not retinal, coordinates. *Journal of Experimental Psychology: Human Perception and Performance*, 31(2):235–247, 2005.

[50] Q. Liu, Y. Hu, L. Shi, X. Mu, Y. Zhang, and J. Tang. EgoNetCloud: Event-based egocentric dynamic network visualization. In *Proc. of the IEEE Conference on Visual Analytics Science and Technology (VAST15)*, pages 65–72. IEEE Computer Society, 2015.

[51] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, 1995.

[52] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman. Temporal event sequence simplification. *IEEE Trans. on Visualization and Computer Graphics*, 19(12):2227–2236, 2013.

[53] M. Monroe, R. Lan, J. Morales del Olmo, B. Shneiderman, C. Plaisant, and J. Millstein. The challenges of specifying intervals and absences in temporal queries: A graphical language approach. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*, pages 2349–2358, 2013.

[54] T. Munzner. H3: Laying out large directed graphs in 3D hyperbolic space. In *Proc of the IEEE Symposium on Information Visualization (InfoVis '97)*, pages 2–10, 1997.

[55] T. M. Newcomb. *The Acquaintance Process*. Holt, Reinhard & Winston, 1961.

[56] M. Ortmann, M. Klimenta, and U. Brandes. A sparse stress model. In *Proc. of the International Symposium on Graph Drawing and Network Visualization (GD '16)*, pages 18–32, 2016.

[57] G. Parker, G. Franck, and C. Ware. Visualization of large nested graphs in 3D: Navigation and interaction. *Journal of Visual Languages & Computing*, 9(3):299–317, June 1998.

[58] P. E. Rauber, A. X. Falcão, and A. C. Telea. Visualizing time-dependent data using dynamic t-SNE. In E. Bertini, N. Elmqvist, and T. Wischgoll, editors, *EuroVis 2016, Short Papers*. Eurographics Association, 2016.

[59] A. Sallaberry, C. Muelder, and K.-L. Ma. Clustering, visualizing, and navigating for large dynamic graphs. In *Proc. of Graph Drawing (GD '12)*, volume 7704, pages 487–498. Springer, 2013.

[60] B. Shneiderman and A. Aris. Network visualization by semantic substrates. *IEEE Trans. on Visualization and Computer Graphics (InfoVis '06)*, 12(5):733–740, 2006.

[61] P. Simonetto, D. Archambault, D. Auber, and R. Bourqui. ImPrEd: An improved force-directed algorithm that prevents nodes from crossing edges. *Computer Graphics Forum (EuroVis '11)*, 30(3):1071–1080, 2011.

[62] P. Simonetto, D. Archambault, and S. Kobourov. Drawing dynamic graphs without timeslices. In *Proc. of Graph Drawing and Network Visualization(GD '17)*, pages 394–409, 2018. https://arxiv.org/abs/1709.00372.

[63] P. Simonetto, D. Archambault, and C. Scheidegger. A simple approach for boundary improvement of Euler diagrams. *IEEE Trans. on Visualization and Computer Graphics (InfoVis '15)*, 22(1):678–687, 2016.

[64] B. Tversky, J. Morrison, and M. Betrancourt. Animation: Can it facilitate? *Int. Journal of Human-Computer Studies*, 57(4):247–262, 2002.

[65] S. van den Elzen, D. Holten, J. Blaas, and J. J. van Wijk. Dynamic network visualization with extended massive sequence views. *IEEE Trans. on Visualization and Computer Graphics*, 20(8):1087–1099, 2014.

[66] S. van den Elzen, D. Holten, J. Blaas, and J. J. van Wijk. Reducing snapshots to points: A visual analytics approach to dynamic network exploration. *IEEE Trans. on Visualization and Computer Graphics*, 22(1):1–10, 2016.

[67] C. Walshaw. A multilevel algorithm for force-directed graph-drawing. *Journal of Graph Algorithms and Applications*, 7(3):253–285, 2003.

Paolo Simonetto received his masters degree in Computer Engineering from the University of Padua and a PhD degree from the University of Bordeaux. His principal research interests include the visualisation of overlapping sets and dynamic networks.

Daniel Archambault received his PhD from the University of British Columbia in 2008 and is a Senior Lecturer at Swansea University. His principal area of research is network visualisation and evaluating the perceptual effectiveness of such approaches. In particular, he has focused on the development and evaluation of techniques for visualising dynamic graphs and scalable graph visualisations.

Stephen Kobourov is a Professor at the Department of Computer Science at the University of Arizona. He received a BS degree in Mathematics and Computer Science from Dartmouth College and MS and PhD degrees from Johns Hopkins University. His research interests include information visualisation, graph theory, and geometric algorithms.