DOI: 10.1111/cgf.14615

Event-based Dynamic Graph Drawing without the Agonizing Pain

A. Arleo,[1] S. Miksch[1] and D. Archambault[2]

[1]Institute of Visual Computing and Human-Centered Technology, TU Wien, Vienna, Austria
miksch@ifs.tuwien.ac.at
[2]Swansea University, Swansea, UK
d.w.archambault@swansea.ac.uk

Abstract
*Temporal networks can naturally model real-world complex phenomena such as contact networks, information dissemination and physical proximity. However, nodes and edges bear real-time coordinates, making it difficult to organize them into discrete timeslices, without a loss of temporal information due to projection. Event-based dynamic graph drawing rejects the notion of a timeslice and allows each node and edge to retain its own real-valued time coordinate. While existing work has demonstrated clear advantages for this approach, they come at a running time cost. We investigate the problem of accelerating event-based layout to make it more competitive with existing layout techniques. In this paper, we describe the design, implementation and experimental evaluation of **MultiDynNoS**, the first multi-level event-based graph layout algorithm. We consider three operators for coarsening and placement, inspired by Walshaw, GRIP and FM^3, which we couple with an event-based graph drawing algorithm. We also propose two extensions to the core algorithm:* AutoTau *and* Bend Transfer. *We perform two experiments: first, we compare* MultiDynNoS *variants to existing state-of-the-art dynamic graph layout approaches; second, we investigate the impact of each of the proposed algorithm extensions.* MultiDynNoS *proves to be competitive with existing approaches, and the proposed extensions achieve their design goals and contribute in opening new research directions.*

Keywords: Visualization, Graph Drawing, Temporal Networks

CCS Concepts: • Human-centred computing → Graph drawings; Visualization

1. Introduction

Typical graph drawing and network visualization algorithms model the temporal axis of dynamic data as discrete, or *timesliced* [BBDW17]. Timeslicing entails capturing snapshots (*Timeslices*) of the dynamic graph, usually taken at regular intervals, each one representing the state of the graph at a specific instant. Timeslice-based dynamic graph drawing has two advantages: it works well for clearly defined time intervals (e.g. yearly, monthly, etc.) and allows for existing static layout algorithms to be applied directly. However, many dynamic graphs do not fall neatly into timeslices. For example, in social media analysis, messages have precise times when they were sent. In contact tracing networks, a contact happened at a specific time with a specific duration. When nodes and edges have real-time time coordinates, projecting onto the nearest timeslice is required to draw such graphs, which inevitably results in a quantization error, potentially reducing drawing quality (see supplemental video and Section 3.2).

Event-based networks (known and described as *temporal networks* [HS12]) consider networks where each edge and node has its own time coordinate and duration. Such networks have been studied from a network analysis perspective extensively in the past, but they have not received much attention in the visualization literature until recently [SAK17, SAK20, AMA21]. Unlike timeslice-based approaches, algorithms to draw event-based networks exploit the full temporal resolution of the data by optimizing node *trajectories* in the *space-time cube* (2D + t), outperforming timeslice-based techniques in terms of drawing quality [SAK17, SAK20] albeit with significant costs in terms of running time and computational resources. These higher running times have limited the use of event-based graph drawing on networks to a thousand or so events, despite the quality improvements over timeslice-based techniques.

In the past, static graph drawing methods have been significantly improved through multi-level graph drawing techniques [BGKM10, Wal03, GK00, HJ04, AMA07, ADLM18]. In a multi-level approach, the input graph is first coarsened into a hierarchy of coarse graphs, which are smaller than the original. Afterwards, they are drawn in reverse order (from the coarsest to the finest) propagating the layout information down to finer levels of the hierarchy. This has two key advantages: first, as the coarser levels of this hierarchy are smaller in terms of the number of nodes and edges, they can be drawn quickly, increasing the overall speed of the layout algorithm. Second, the drawing of each level is used to 'place' the subsequent in the hierarchy: this gives an optimal initial layout, which in turn results in increased layout quality when compared to a random initial condition. In 2D graph drawing, there have been proposed several different nuances of this multi-level pipeline (see Section 2), some of which inspired the approaches described later in this paper.

A multi-level approach, in fact, can be applied to event-based graph drawing by coarsening node trajectories, instead of nodes themselves, and embedding them in the space-time cube. As existing event-based layout algorithms [SAK20] share their base methodology with force-directed layout (see Section 3.2), we investigate whether applying a multi-level layout pipeline to the event-based dynamic graph drawing scenario can improve the limited scalability of the existing approaches [SAK20]; we also study whether this has any effect on the final drawing quality. Within this context and motivation, we present *MultiDynNoS*: the first multi-level event-based graph drawing algorithm that brings the drawing time required for event-based networks to a level that is comparable with timeslice-based approaches. We achieve this goal by applying coarsening operators to node trajectories and drawing them directly in the space-time cube without timeslices. Similar to standard multi-level techniques for static graphs, *MultiDynNoS* follows a *coarsening-refinement* strategy. We adapt the coarsening and placement strategies of Walshaw [Wal03], GRIP [GK00] and FM^3 [HJ04] to operate on node trajectories for drawing temporal graphs in the space-time cube. A first, preliminary version of this research has been presented at EuroVis 2021 in the form of a short paper [AMA21]. Our contributions are as follows:

- An extended and revised description of the *MultiDynNoS* algorithm, enriched with new figures and more technical details, including parameter optimizations over the previous design [AMA21].
- A discussion of two extensions the core algorithm, namely *AutoTau* and *Bend Transfer*: the first improves the usability of our approach, while the second is designed to obtain a more efficient and accurate layout process.
- An extended experimental study where we compare the core algorithm to state-of-the-art timesliced techniques and *DynNoSlice* on both small and large real graphs.
- A second experimental study where we investigate the single and combined effects of the proposed extensions on the core algorithm performance.

The paper title pays tribute to the work by Shewchuk [She94]: as they attempt at relieving the 'pain' in comprehending a notoriously hard concept (the conjugate gradient method), with this paper, we are aiming at doing the same with event-based graph drawing, both in understanding and utilizing it.

2. Related Work

The visualization of dynamic graphs has been studied extensively [BBDW17]. A number of techniques have been proposed that map time-to-time (animated techniques) [APP10, AP16, FQ11, BPF14]. Other approaches map time-to-space [SA06, BVB*11, LHS*15, AB20, LAN19, LAN21]. The perceptual effectiveness of such techniques has been evaluated through human-centred studies [APP10, FQ11, AP16, LAN21] formally evaluating when such techniques work well. Our contribution lies in the drawing of dynamic graphs without timeslices using a multi-level strategy. Therefore, we present related work most closely related to this area.

Multi-level graph drawing. Force-directed layout algorithms have been extensively used in different application areas, as they are easy to implement, allow for several tuning options and provide satisfying results in terms of drawing quality. However, they suffer from high running times on larger graphs (in the order of thousands of nodes and edges) and tend to converge into sub-optimal solutions [BGKM10] (i.e. a layout where the graph does not appear properly 'unfolded'). In the 2000s, to address these issues, multi-level graph drawing algorithms [Wal03, AMA07, GK00, HJ04, BGKM10] were devised to scale graph drawing algorithms to larger datasets. These algorithms were targeted at static graphs or networks, where the nodes and edges do not change and a single layout is sought that best reflects network structure. Multi-level graph drawing approaches have been adapted to an online dynamic setting [CCM17, Vel07, Cra16] where streaming data comes in and the approach has no opportunity for lookahead and to optimise the drawing over the full time interval. Multi-*layer* networks, where several node and edge layers have different meaning [MGM*19], have also been used for visualization. Although similar, multi-layer networks consider layers and interdependent sub-systems and not the same network over time.

In this paper, we adapt the multi-level 'coarsening-refinement' scheme the offline event-based graph drawing setting, to scale up to a larger number of events while still using the temporal information in full when embedding the network in the space-time cube.

Temporal networks and event-based visualization. Temporal and event-based networks [HS12, LVM18] have been studied extensively for automatic graph analysis where algorithms try to find or count features in the network. In the field of graph drawing and visualization, all approaches for drawing the dynamic network required a series of timeslices projected timeslices and a way of encouraging a stable drawing [BBDW17]—the position of nodes and edges should change as little as possible when a change is made to the graph [CP96] in order to help with node identification as the graph evolves through time [AP12]. Algorithms have been explored to optimize the simultaneous drawing of timeslices in offline [DG02, DGK01, EHK*03, BM11] and online [MELS95, GDBG12, FT08] scenarios. Methods for non-uniform timeslicing of temporal networks, based on the network data, have also been explored [WAH*19].

Recently, event-based visualization techniques [DSP*17, MLMdO*13, MLL*13] display event sequences with real-time coordinates for each data point. In the graph drawing and visualization domain, event-based layout algorithms were devised to directly draw these event-based/temporal graphs in the space-time cube [SAK17, SAK20] allowing for the visualization of them across time. Other graph drawing techniques, such as HOTVis [PS21], draw event-based data by exploiting the temporal ordering of the edges (the *causal paths*) to shape the layout. However, this approach focuses on 2D visualizations and does not optimize the drawing across the space-time cube.

Contribution. The literature indicates a growing interest in event-based visualizations of networks for visual analytics applications. Event-based dynamic graph drawings can potentially yield improved drawing quality over timeslice-based approaches, motivating our research on more scalable techniques for embedding temporal networks in the space-time cube.

3. MultiDynNoS

In this section, we describe *MultiDynNoS* core algorithm design and two extensions aimed at simplifying its use and extending its flexibility. We first provide a set of definitions and a short reminder of the workings of *DynNoSlice*, both necessary to ease the comprehension of the remainder of the paper.

We show animated layouts produced by our algorithm in the supplemental video, and snapshots of the network evolution of four graphs, also included in our experimental evaluation (see Section 4), as drawn by *MultiDynNoS* in Tables 1 and 2.

3.1. Definitions

Consider a temporal network $D = (V, E, T)$ where each node and edge possesses a number of *attributes*, which are functions of time. The *appearance* of a node $v \in V$ is defined as $A_v : V \times T \to [true, false]$ (edge appearance is defined similarly), which maps to node/edge insertion and deletion in the event-based graphs. A_v defines a series of intervals in T (time) in which the node/edge is present. The *position* of a node in the plane over time is defined as $P_v : V \times T \to R^2$. As an example, consider a node present in the intervals [1,4) and [8,10) only. Its A_v is defined as follows:

$$A_v = \begin{cases} true & \text{if } t \in [1, 4) \\ true & \text{if } t \in [8, 10) \\ false & \text{otherwise} \end{cases} \tag{1}$$

When defined in this way, the appearance and position of the nodes are represented as a series of *trajectories* (polylines) through time that will be embedded in the three-dimensional space-time cube (2D + t). These trajectories present bends that define node movement in the two dimensional plane as time passes downwards in the cube (e.g. Figure 1). This replaces the inter-timeslice edges linking the same vertices across time in timeslice-based approaches [BM11]. We also define a *flattened* graph as the weighted static counterpart of a temporal graph where node and edge weights represent the cumulative duration of the time when a node or edge is present or, more formally, the cumulative duration of intervals in which the appearance function yields true for that node or edge. We also define $t_{max} = \max(t) \in T$ and $t_{min} = \min(t) \in T$.

3.2. DynNoSlice

DynNoSlice [SAK17, SAK20] is an offline dynamic graph drawing algorithm that draws temporal networks without requiring timeslices to be imposed on the space-time cube. Previous techniques based on timeslices [BM11] would start by regularly sampling along the time dimension, creating a series of static graphs that are drawn together by linking the same node across time. When the data are not neatly divided into timeslices, the quality of the resulting layouts is reduced in timeslice-based drawings (see, e.g. the supplemental video) as the full temporal resolution of the data set cannot be exploited for layout. The main reason why timeslicing can introduce these problems when applied to event-based networks is that the frequency of events may not be the same throughout the data set. When the frequency of events is low, regular timeslicing oversamples the time dimension and can introduce movement in nodes by forcing them to reside in timeslices even though they seldom interact with other nodes in that time period. When the frequency of events is high, regular timeslicing can over-aggregate events into a small number of timeslices incurring losses in valuable information about the order of events and their frequency. In event-based data, due to this variation in frequency, it is very hard to find a regular timeslicing that satisfies both of these simultaneously. Information loss can be avoided by sampling time (i.e. creating a new timeslice) at the Nyquist frequency, that is, in this context, the smallest time gap between events. This, however, would often result in a prohibitively large number of timeslices. Also, selecting timeslices first would require recomputing the drawing of all the individual timeslices when changing time resolution, whereas in a direct drawing in the space-time cube [BDA*17], resolution can be changed without need for redrawing.

DynNoSlice exploits the full-time resolution of the data, drawing it directly in the space-time cube and thus does not sample along the time dimension before drawing. Instead, it embeds the node trajectories and edge surfaces in 2D + t. At its core, it is a force-directed drawing algorithm, iteratively improving the 3D embedding of the node trajectories in the space time cube, minimizing the energy of the layout by finding a state of equilibrium between the physical forces acting on nodes and edges. The node polyline segments will repel each other and edge surfaces will pull these closer together: in addition, there is a gravity that pulls trajectories towards the centre (to promote compact drawings). Trajectory post processing is made to ‘smooth’ node movements over time and the user mental map [AP16] is preserved by limiting the maximal horizontal movements of the trajectories.

The overall complexity of the algorithm is quadratic in the number of events [SAK17, SAK20]. Two parameters are required for the algorithm to work: the trajectories ideal distance δ (used to simplify the node repulsion forces calculation) and τ, which transforms a unit in the time coordinates in τ units of space. The choice of τ can substantially affect the performance of the algorithm: smaller values will provide dense cubes (each time unit will span less space units), and *vice versa* with larger ones. The choice of τ depends on the original time units, the *rate* of events of the dataset to draw and on the desired effect of time on the space dimension. In *DynNoSlice*, such value must be provided by the user prior to the layout generation, thus making it more difficult to use.

Table 1: *Flattened snapshots of the network evolution over time taken at regular intervals. Twenty artificial timeslices were inserted for temporal graphs.*

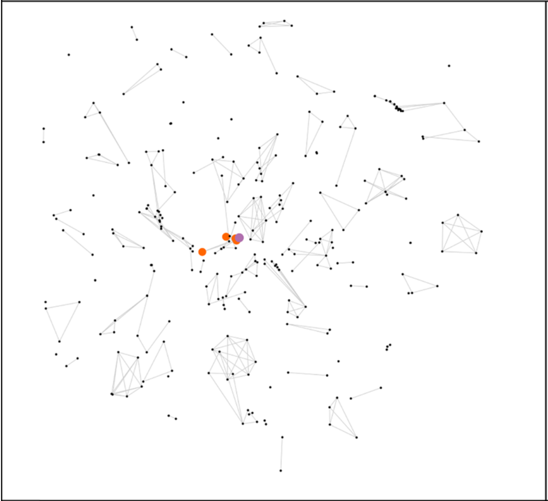
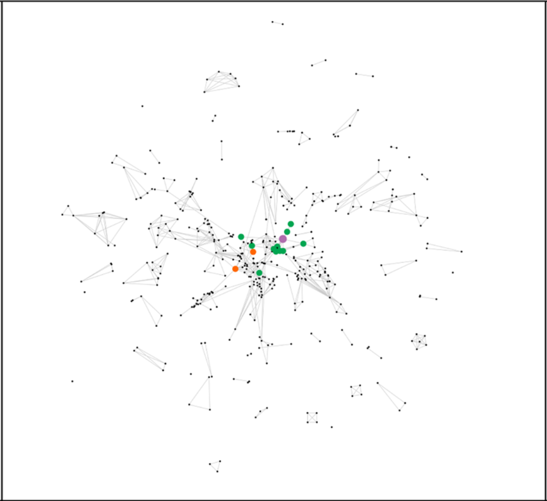
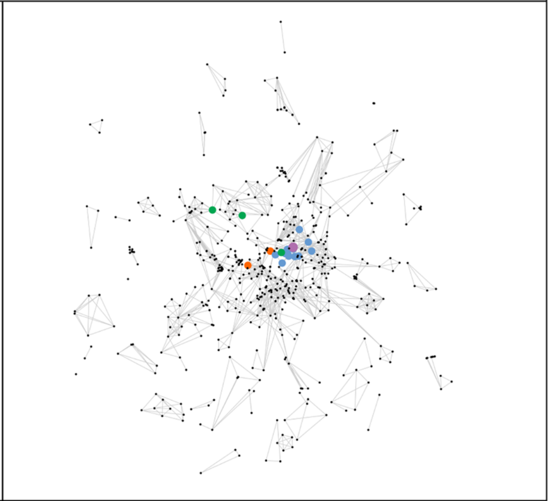
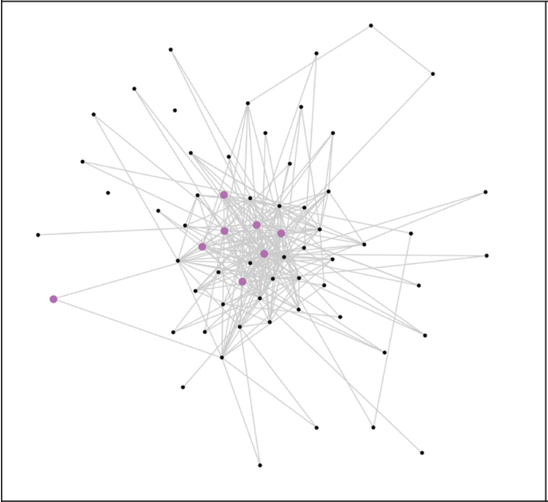
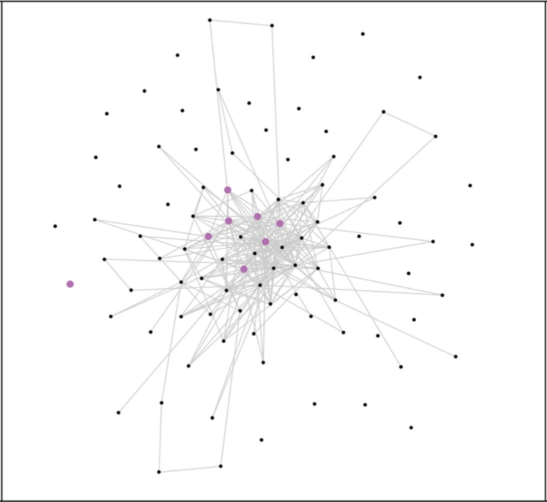
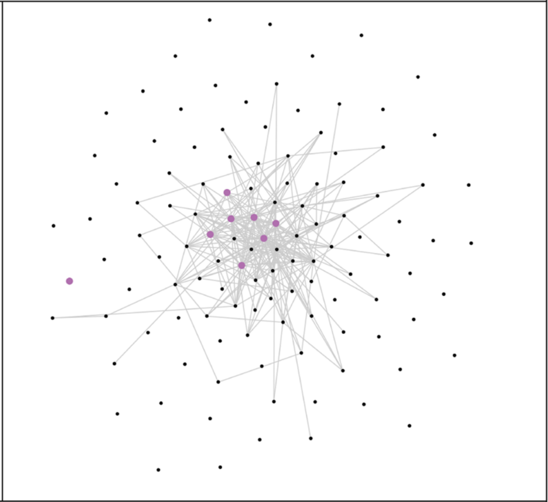
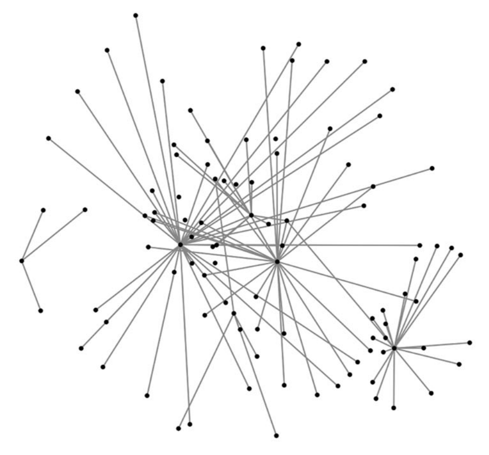
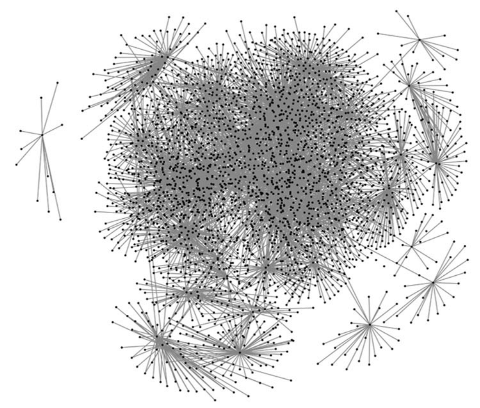
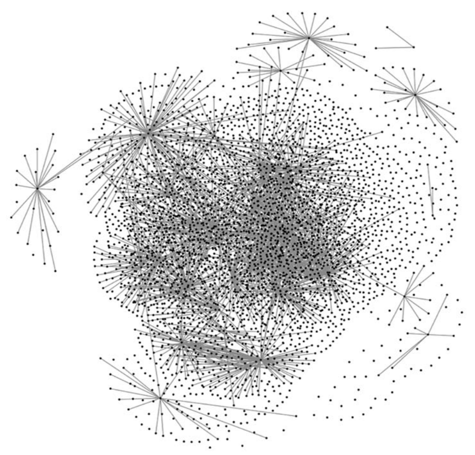
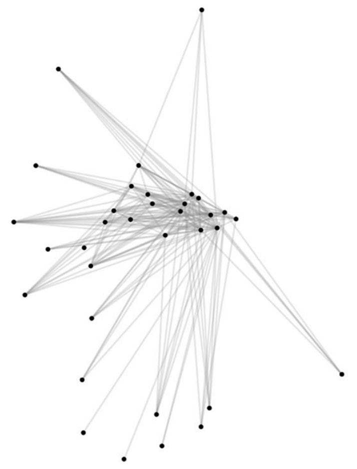
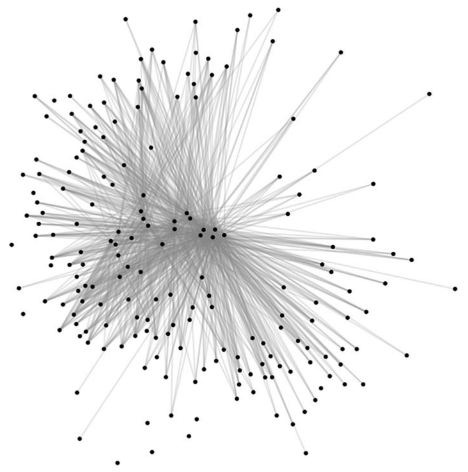
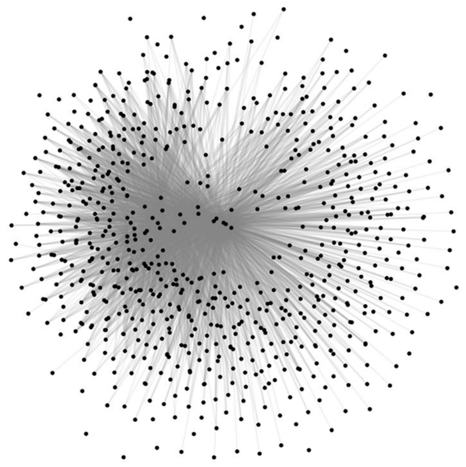
`InfoVis` (timesliced graph - 21 timeslices)		
		
Purple node is Prof. Munzner. The neighborhood is higlighted in different colors.		
`Dialogs`		
		
The nodes representing the Bennet family (the protagonists of the book) are highlighted.		

Table 2: *Flattened snapshots of the network evolution over time taken at regular intervals. Twenty artificial timeslices were inserted for temporal graphs.*

RM		
		
Edges have been darkened and thickened to improve the readability of the evolution of the underlying structure.		
MOOC		
		
It is possible to see how the nodes representing the courses are kept in the middle by *MultiDynNoS*, while students' nodes "orbit" around them.		

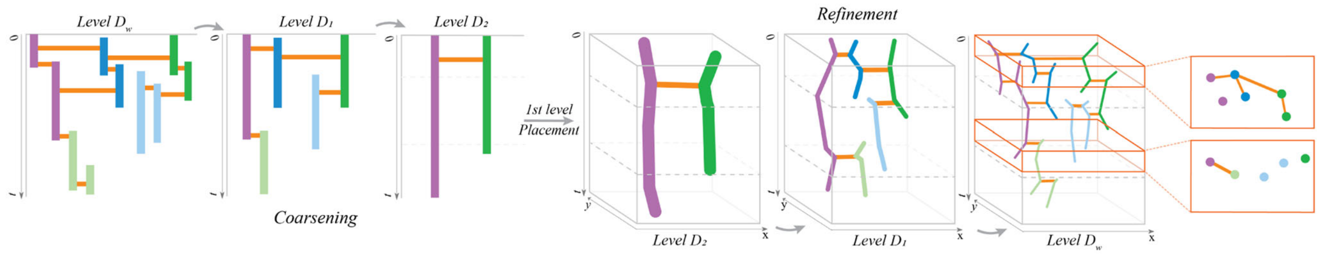


Figure 1: *The complete drawing cycle of* MultiDynNoS, *from left to right. Vertices are represented as bars/trajectories while edges are shown in orange. On the extreme right side of the picture, we show two timeslices of the space-time cube of the temporal graph final layout.*

3.3. Multi-level acceleration for *DynNoSlice*

In Sections 1 and 2, we introduced the concept of a multi-level pipeline and shortly described its success story when applied to static 2D layout algorithms. As *DynNoSlice* is, at its core, a force directed algorithm [SAK20] (see also Section 3.2), we investigate whether applying a multilevel pipeline can provide scalability and potential quality improvements. The main contribution of this paper is a *coarsening+refinement* multi-level temporal graph drawing algorithm, *MultiDynNoS*, that uses *DynNoSlice* as single level drawing technique. *MultiDynNoS* coarsening operator recursively groups trajectories into a hierarchy of coarser trajectories. Subsequently, starting from the top level (coarsest) of the hierarchy and down to the input network, for each level a layout is drawn with *DynNoSlice* and the final positions of the trajectories are transferred to the next level to provide a good initial placement for the subsequent drawing. Since the single level layout algorithm is *DynNoSlice*, the δ and τ values have to be provided as input to *MultiDynNoS*. Our approach can be summarized in three different stages, listed in execution order as follows (see also Figure 1).

▷ **Coarsening**:
1. Apply a *coarsening* operator to create a hierarchy of coarse trajectories that will be used to draw the graph in the space-time cube.

▷ **First level placement**:
1. Draw, using a static layout algorithm, a flattened version of the coarsest level of the hierarchy;
2. Place the trajectories in the space-time cube using the positions of the nodes in the flattened graph, and extrude them down through time.

▷ **Refinement**:
1. Draw the trajectories using *DynNoSlice* [SAK17, SAK20];
2. Place the trajectories in the space-time cube of the level below by using a *placement* strategy, taking into account their position at t_{max}.
3. Extrude the newly placed trajectories vertically down.
4. If the current is not the input graph, repeat refinement for the lower level.

3.4. Coarsening

Coarsening takes the list of events (node and edge appearances/disappearances along with their duration) and creates a hierarchy of coarse node trajectories that will be used to draw the temporal network. This hierarchy is created by recursively merging sets of trajectories together until only a small number of trajectories remains or the coarsening operator makes little progress on the input. Coarsening strategies have analogues to multi-level static graph drawing algorithms [Wal03, GK00, HJ04], but operate on the trajectories that will be embedded inside the space-time cube.

More formally, given an input temporal network D, we first flatten D to obtain D_f, which is the static, *flattened* (see Section 3.1) version of D. In D_f, nodes and edges are weighted, representing their cumulative appearance. We then transfer these weights from D_f to D, ultimately obtaining D_w. D_w is the original temporal graph D but with its node and edges bearing a new constant attribute representing their cumulative duration obtained from the node and edge weights of D_f. Then, we perform the coarsening, which yields a hierarchy of coarse event-based graphs $D_H = \{D_w, D_1, \dots, D_k\}$, with a depth k. The finest level is D_1 and the coarsest is D_k.

The hierarchy is obtained by selecting recursively, starting from D_w, representative vertices (trajectories) in the current local neighbourhood to form a coarser graph one level up in the hierarchy. For any level D_i with $D_i = (V_i, E_i, T)$, level D_{i+1} is created as follows. First, sort the vertices of V_i by their weight and put them on a stack. We pop the stack and get the heaviest vertex v_i: its copy v_{i+1} is then assigned to V_{i+1}. At this point, the coarsening operator is applied to the neighbourhood around v_i where a subset of its vertices (depending on the current coarsening strategy) is selected. The weights and appearance intervals of these selected vertices are merged with v_{i+1}. We refer to v_i as the *representative* of v_{i+1} in V_i (which we indicate as $\overline{v}_i$), and the vertices merged with it as the representative's *neighbourhood*. We refer to the set of representatives at level i as $\overline{V}_i$. At this point, the vertices merged with v_i are removed from the stack, and a new representative can now be chosen. This process is repeated until the stack is empty.

We generate E_{i+1} as follows. First, an edge in E_i only appears E_{i+1} if its endpoints are both selected as representatives. Therefore, for each edge $e_i = (\overline{v}_i, \overline{w}_i)$, a copy is created in E_{i+1} as $e_{i+1} = (v_{i+1}, w_{i+1})$. Each edge in E_i, whose endpoints both belong to the same neighbourhood generated by $\overline{z}_i$, contributes to the cumulative weight of $z_{i+1} \in V_{i+1}$. Finally, for each edge $e_i = (t_i, w_i)$ where t_i and w_i belong to different neighbourhoods, its duration and weight are merged with the edge between t_i and w_i representatives in E_{i+1}.

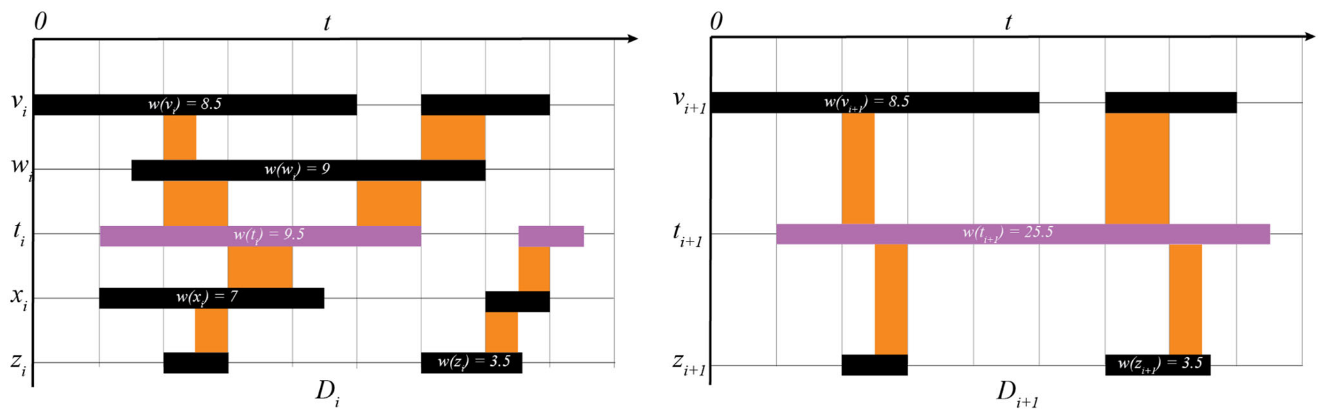


Figure 2: *Three combinations of placement and coarsening methods, as seen from the top of the space time cube, applied on the same graph (a): identity placement with Walshaw coarsening (b), barycentre with independent set coarsening (GRIP) (c) and FM^3 with galaxy partitioning trajectory placement (d). Node size represents the coarsening weight; node representatives have numbers inside showing the coarsening order. Nodes coarsened together share the same colour. Dashed lines represent the edges between the representatives (larger nodes) at the upper level. Solid orange lines represent the edges at the current level.*

At this point, level D_{i+1} has been defined. Coarsening stops at the hierarchy level D_k when the node count is $\geq 95\%$ the size of level D_{k-1}. This was introduced to avoid unnecessarily deep hierarchies.

We implemented three different coarsening strategies. First, we implemented the *Maximal Matching*, found in the multi-level approach by Walshaw [Wal03], chosen as it was the coarsening strategy of the first multi-level graph drawing algorithm published. In maximal matching, pairs of vertices connected by an edge belonging to the graph maximal matching are merged together in each level ordered by their weights: in practice, once a vertex is selected, it is merged only with its 'heaviest' neighbour (see, e.g. Figure 2b). Second, we implemented the *maximal independent set* coarsening, used by *GRIP* [GK00]. In this approach, once a vertex is selected to be part of the new level, it is merged together with all of its direct neighbours (see, e.g. Figure 3). Finally, we implemented the *galaxy partitioning* operator, used by *FM*^{3} [HJ04]. Each selected vertex is merged with its neighbours up to distance 2, creating a 'Solar System' partitioning of the graph. These last two methods were chosen as they outperformed other coarsening algorithms both in terms of number of levels and running times in the experimental evaluation conducted by Bartel *et al*. [BGKM10].

3.5. First level placement

First level placement is responsible for placing the coarsest level of the hierarchy, i.e. D_k. It is flattened into D'_k (see Section 3.1 for the definition of the flattening operator) and is drawn with a force-directed layout algorithm. As the coarsest level is usually small, a simple single level technique (such as the drawing technique by Fruchterman and Reingold [FR91]) can be used. The resulting node coordinates are used to place the trajectories of level D_k. Subsequently, they are extruded vertically downwards in the space-time cube along the time dimension according to the appearance of the nodes (see Figure 4).

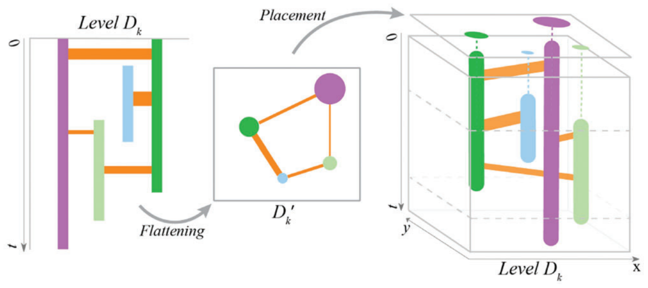


Figure 3: *Independent set coarsening example between level i and i* $+$ 1*. Horizontal black bars represent the nodes, while the orange rectangles represent the edges. Each cell of the grid represents one time unit. Function $w(\cdot)$ yields the coarsening weight of the node; t_i has the greatest weight, so it will be chosen as representative for level D_i and copied to D_{i+1}. In this coarsening strategy, all the neighbours of the chosen representative are merged with it at the upper level. v_i and z_i remain isolated, and are therefore 'passed' to the upper level. The nodes and edges merged into t_{i+1} contribute to its weight with their duration.*

3.6. Refinement

Given a drawing of trajectories in the space-time cube at level D_i, initial positions for all the nodes V_{i-1} in D_{i-1} need to be assigned based on the trajectories of V_i, that is the coarser level of the hierarchy that has already been drawn. First, for each $v_i \in V_i$, its position at t_{max} becomes the initial placement for the corresponding vertices in $\overline{V}_{i-1}$. Then, the placement operator assigns the coordinates for all of the vertices included in each representative neighbourhood. Once placed, these trajectories are then extruded down across time where their bends can be computed during the subsequent layout phase (see Figure 5). A good initial placement is expected to yield trajectories with few bends, resolving in nodes with smoother movement.

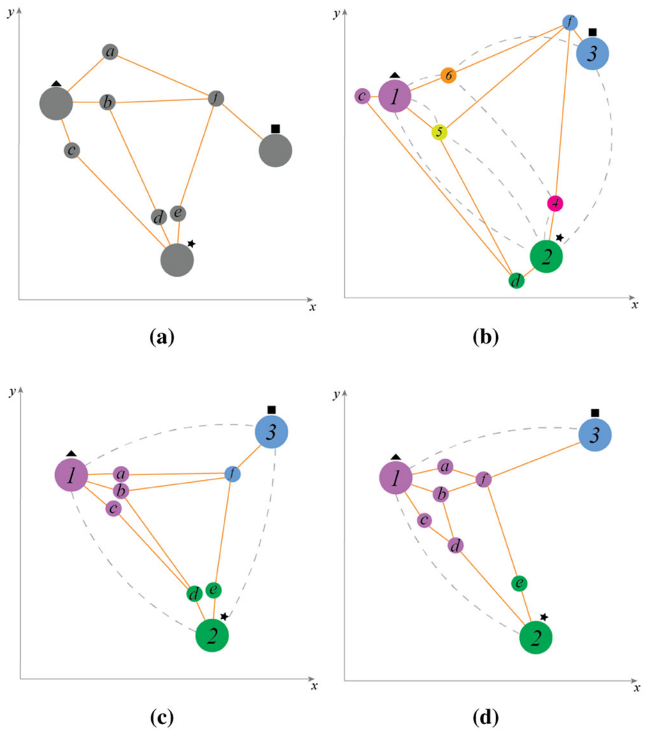


Figure 4: *First level placement example. The temporal graph is flattened into D'_k, which is then drawn using a force directed layout. The computed coordinates of the nodes in D'_k are used to place the trajectories of level D_k in the space-time cube.*

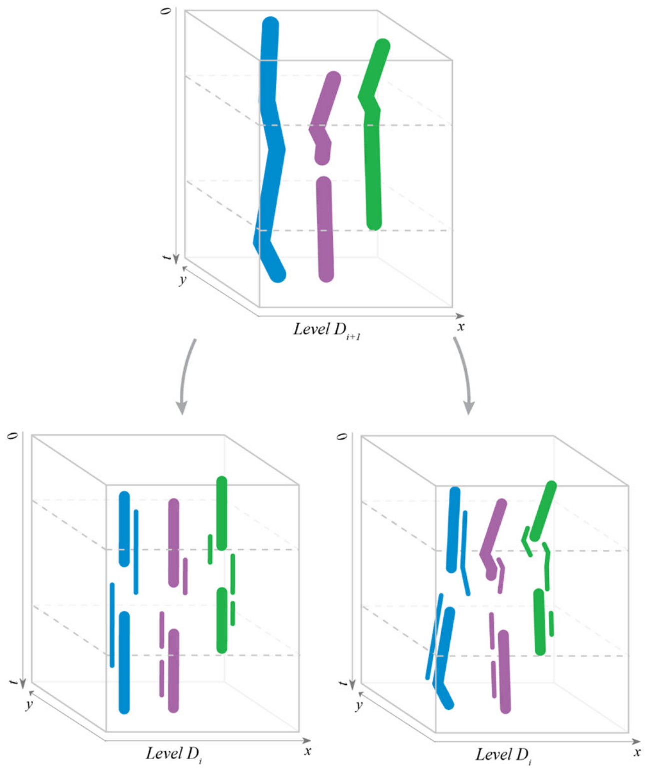


Figure 5: *Standard (left) and new placement (right). Representatives are shown with a larger width than the vertices merged with them. In standard placement, trajectories are placed at their final position at level D_{i+1} and then extruded vertically. In placement with bend transfer, the trajectory and bends of the representative are copied to the lower level and to the vertices merged with it.*

As we were inspired by three static graph drawing multi-level algorithms to create the coarsening operators used in *MultiDynNoS*, namely Walshaw [Wal03], *GRIP* [GK00] and *FM*^{3} [HJ04] (see Section 3.4), we complement them with placement methods inspired by the same techniques—as they were originally meant and designed to work together. We name each of our placement techniques as the algorithm they were inspired from for the sake of simplicity. Walshaw [Wal03] placement uses the *identity placer*: the trajectories are placed in the same position as their representative (see, e.g. Figure 2b), plus a small random perturbation to avoid being placed on the exact same coordinates. The GRIP [GK00] placement uses the *barycentre* of the coordinates (Figure 2c) of the representative's neighbours at level $i+1$ at t_{max}. The final positioning of the trajectory is skewed towards its own representative by a fixed rate. FM^3 placement is similar to barycentre but changes the attraction of the representative cluster. In the *FM*^{3} coarsening [HJ04] (see Section 3.4), all neighbours up to distance 2 from their representative are merged together: this guarantees that any pair of representatives is at most at distance 5 from each other. Therefore, it is possible to precisely reconstruct the relative position of the merged trajectories in all the paths between theirs and other representatives and place them accordingly (Figure 2d). For all approaches, randomness is added to the final coordinates to avoid possible accidental coordinate overlaps.

After placement, the trajectory bends and positions are refined using *DynNoSlice* [SAK17, SAK20]. As coarse graphs are smaller and therefore quicker to draw, more quality-oriented (time intensive) layout parameters can be used. As graph size increases with finer levels, speed becomes a priority. We tune two parameters in our approach: the maximum node mobility and the number of layout algorithm iterations. Coarser levels of the hierarchy will benefit from more flexible trajectories, while finer levels have fewer iterations and less flexible trajectories. Maximum node mobility is set as default as 2δ, with δ being the ideal distance between two trajectories. δ is set to 5. The coarsest level will go through 75 iterations. Both parameters decrease linearly by 7% at each level, with a minimum value of 3 and 20, respectively. This avoids them to decay too much on deeper hierarchies, which could make the nodes practically immovable after placing with detrimental effects on the final layout quality. We got the inspiration for this simple yet effective measure from our previous experiments [AMA21]. There, we observed how the drawings obtained with the Walshaw coarsening and placement (see Figure 2b), that had the deepest hierarchies due to the workings of the coarsening method (see Section 3.4), consistently had movement figures closer a static drawing than to a dynamic one.

All layout tuning values were obtained empirically when considering the quality/running time trade-off. Time trajectory post-processing of *DynNoSlice* [SAK17, SAK20], meant to adjust the complexity of the node trajectories, runs at level D_w at the first and then every 30 iterations, meaning that shallow hierarchies (i.e. less levels mean less time for parameters to decrease) will experience more post-processing rounds than deep ones. Deeper hierarchies, however, will have more levels to improve their trajectories over time, decreasing the loss of quality due to the reduced post-processing while maintaining competitive running times. Similarly as with the parameter tuning described above, this strategy was initially designed and then refined through empirical evaluations to find the best trade-off between layout quality and running times.

Once the layout for D_i is computed, and $D_i \neq D_w$, the final coordinates are used to *place* the node trajectories on level D_{i-1} (see above). Otherwise, the process ends.

3.7. *MultiDynNoS* extensions

In this journal paper, we experimented with two different extensions to the core algorithm, which are presented and discussed in the following.

3.7.1. *AutoTau*

We discussed in Section 3.2 that a significant barrier for users when using *DynNoSlice*, and consequently *MultiDynNoS* (see also Section 3.3), is the selection of the τ value, a parameter which controls the conversion of time-to-space when representing graphs in the space-time cube. Time, as measured in the data set, can be converted to space (the z-axis in the space-time cube) at many different scales. The value τ controls the scale of this conversion, with larger values corresponding to long and 'skinny' space-time cubes and smaller to short and 'fat' cubes. As τ decreases, the trajectories in the cube are pressed closer together increasing the strength of forces between trajectories. In previous versions of *MultiDynNoS* and *DynNoSlice*, τ was selected manually based on previous experience, estimates from a rough formula [SAK20] and experimentation. Most users would prefer an automatically computed τ that while not perfectly tuned to obtain the best layout quality, could be used right out-of-the-box. Moreover, it also represents an initial indication of the event distribution, providing a starting point for finding the best τ for the graph at hand.

We compute our *AutoTau* as shown in Equation (2). In its numerator, we calculate the average event duration as the sum of all true time intervals, across both nodes and edges, and divide by the total number of events. Then, in the denominator, we separately calculate the time difference between the latest and the first true time interval in A_v and A_e union sets, which we sum together.

$$\tau = \frac{\left(\frac{\sum t(A_v)+\sum t(A_e)}{\sum_v |A_v|+\sum_e |A_e|}\right)}{\Delta t(\bigcup_v A_v) + \Delta t(\bigcup_e A_e)} \qquad (2)$$

where $t(\cdot)$ yields the cumulative time duration of all the true intervals in A_v or A_e, $|A_v|$ being the cardinality of true time intervals in a node/edge appearance function and $\Delta t(\cdot)$ yields the time difference across the time intervals in its argument. We assume all time intervals to have finite duration. In case, infinite intervals are present (either with a left value of $-\infty$ and/or a right value of ∞), we clip the space-time cube to the existing finite intervals.

3.7.2. *Bend transfer during placement*

After placing a finer level of the hierarchy into the space-time cube, based on the coordinates of trajectories in the coarser level (see Section 3.6), the initial trajectories are extruded downwards along the space-time cube time axis (Figure 5 left). All of the bend information of coarser levels is lost and is recalculated at finer levels. The finest level, or original graph in the space time cube, computes these bends for the final time, defining the trajectories of the nodes across time. With this approach, we discard some of the information about node movement (i.e. the trajectories' bends) at the coarser levels of the hierarchy. Considering that the intuition behind the placement phase of a layout multi-level strategy (see Section 3.3) is to use the freshly computed coarse layout information to support and speed up the drawing at the following level, we investigate whether retaining bend information from the coarser levels may help define trajectories at finer levels.

Our new placement strategy (Figure 5 right), after extruding the newly placed trajectories along the time axis of the space-time cube, *retains* the previous level bend information. This is done in two steps: first, the bends from the trajectories at upper (coarser) level are copied to their corresponding representatives at the lower (finer) level. Then, at the lower level, for all the trajectories merged with each representative, its bends are copied whenever the merged node trajectory has an appearance value of true, so that it 'follows' the path of its representative.

4. Experimental Evaluation

We structured our evaluation as two separate experimental studies. In the first one (Section 4.5), we aim to assess the performance of the core algorithm of *MultiDynNoS* (see Section 3.3) in terms of drawing quality, using recognized quality metrics specific for temporal graphs, and running times. We compare *MultiDynNoS* performance with *DynNoSlice* and a state-of-the-art timeslice-based layout technique. In our second experiment (Section 4.6), we evaluate the impact of *MultiDynNoS* extensions

On the running times and drawing quality. We first discuss the strategies tested, the quality metrics and the datasets, which are common to both experiments.

4.1. Layout strategies

We test three different variants of *MultiDynNoS*, based on different combinations of coarsening and placement strategies. Specifically:

- `MultiDynNoS wi_id` is the Walshaw variant of *MultiDynNoS* with maximal matching coarsening and identity placement.
- `MultiDynNoS is_gr` is the GRIP variant of *MultiDynNoS* with maximal independent set coarsening and barycentre placement.
- `MultiDynNoS sm_sp` is the FM^3 variant of *MultiDynNoS* with the FM^3 coarsening and placement strategy.

We compare them with three state-of-the-art layout strategies, described in the following:

- `Visone` [BW04] is a timeslice-based dynamic graph drawing algorithm, using the linking strategy. It performed the best in a previous study [BM11].
- `DynNoSlice` [SAK17, SAK20] is the event-based layout algorithm used in *MultiDynNoS* as single level layout (see Section 3.6).
- `sfdp flat`: flattens the entire event-based data and draws it once as a static graph using `sfdp` [Hu05] drawing algorithm.

The `sfdp flat` is a baseline. It has zero movement, and since it operates in two dimensions we expect it to have a major advantage in terms of running times. However, this can come at the cost of higher stress on some datasets where node movement is required.

4.2. Datasets

Three of the datasets used in our experiments are naturally expressed as timeslices while the other are more naturally expressed as temporal networks.

- VanDebunt shows the relationships between 32 freshmen at seven different time points [VDBVDS99]. Timeslices and edges are selected as in the paper by Brandes and Mader [BM11].
- Newcomb contains the sociometric preference of 17 members of a fraternity in the University of Michigan in the fall of 1956 [New61]. Timeslices are selected as in previous work [BM11].
- InfoVis is a co-authorship network for papers published in the InfoVis conference from 1995 to 2015 [IHK*16]. Authors on a paper are connected in a clique at the time of publication. This is not a cumulative network as authors can appear, disappear and appear again. The dataset has 21 timeslices (one per year).

The temporal networks are the following.

- Rugby is a network derived from over 3000 tweets involving teams in the 'Guinness Pro12' rugby competition. The tweets were posted between 1 September 2014 and 23 October 2015. Each tweet contains information about the involved teams and the time of publication with a precision down to the second.
- Dialogs lists the dialogues between characters in the novel *'Pride and Prejudice'* in order [GWMG16]. The book has 61 chapters with 4000 interactions between characters. When the algorithm required timeslices as input, we divided this data into 61 of them (one for each chapter).
- MOOC represents the actions (e.g. viewing a video, submitting an answer, etc.) taken by users on a popular massive open online class platform [KZL19]. The nodes represent users and course activities (targets), and temporal edges represent the actions by users on the targets. We pick and elaborate the first 15 thousands events.
- RM is the data coming from The Reality Mining study [EP06]. It followed 94 participants using mobile phones pre-installed with several pieces of software that recorded usage data including call logs, short messages and other information. We only consider voice calls and take the first 28 thousand events.

Table 3: *Computed and original values of τ. Graphs ordered by increasing number of events.*

Graph	ManualTau	AutoTau
VanDebunt	5	0.26
Newcomb	5	0.11
RAMP	0.02	0.50
InfoVis	5	0.02
Rugby	$2.31 \cdot 10^{-6}$	0.50
Dialogs	1	$4.05 \cdot 10^{-4}$
MOOC	$5.56 \cdot 10^{-5}$	1.06
MSG	$2.31 \cdot 10^{-5}$	0.40
RM	$6.94 \cdot 10^{-5}$	$1.24 \cdot 10^{-5}$

- MSG is comprised of private messages sent on an online social network at the University of California, Irvine [POC09]. Each message is represented as a temporal edge. We consider the first 15 thousands events.
- RAMP consists of the simulated spread of COVID-19 through a population via the *Scottish COVID-19 Response Consortium* (SCRC) contact tracing model [MMB*]. Nodes are infected individuals in the population and edges occur on the day when the illness was transmitted having that duration. We consider the first 800 events.

When required (e.g. for calculation of the *StressOn* metric), temporal networks were divided into 20 regularly spaced timeslices (unless differently indicated). We remark that all the datasets came with manually defined τ values (see Sections 3.2 and 3.7.1), which were shared between both *MultiDynNoS* and *DynNoSlice* during the evaluation (except when AutoTau extension was tested) to ensure a fair comparison. Manually and AutoTau defined τ values are reported in Table 3.

4.3. Metrics

In our experiments, we quantitatively evaluate the produced layouts on their quality, readability and running times. We selected a set of quality metrics, specific for temporal graphs, listed in the following. We now provide a short description of each one and a reason for inclusion.

Movement: the average distance travelled by a node during graph evolution [BM11, SAK20]. This metric has been presented in previous studies [BM11] and is an important factor related to drawing stability, which helps support the cognitive map of the user [AP12, AP16]. In general, lower movement is desirable but can come at the cost of higher crowding and/or stress values.

Crowding: the number of times nodes pass close to each other in the animation of the dynamic graph [SAK20]. When visually similar nodes pass close to each other, it is difficult to retain their identities [AP12, AP16]. Crowding avoidance is desirable for supporting the cognitive map of the user.

Depth: the depth of the computed hierarchy (multi-level strategies only). Previous studies [BGKM10] highlighted a correlation between the running times and the number of levels generated by the corresponding coarsening strategy. An excessive number of levels might slow down the computation significantly, without any (or negligible) positive effect on the final quality of the layout.

Stress: the stress metric evaluates how well the computed distances between nodes reflect the corresponding graph theoretic distances. Ideally, a good layout algorithm should attempt at minimizing the layout stress. Average stress has been used as a metric to evaluate layout quality in previous studies [BM11, SAK17, SAK20] and we use it again here. For a fair comparison over both timesliced and event-based graphs, we provide two definitions of stress [SAK20]:

- *StressOn*: the average stress across all timeslices computed on all the nodes and edges present on the timeslice. For temporal networks, which do not naturally present timeslices, these are defined manually as described in Section 4.2.
- *StressOff*: the average stress is computed on and for all timeslices using the exact node and edge appearances in continuous time.

StressOn is a more relevant metric for timesliced graphs, while *StressOff* is to be preferred for temporal graphs.

Applying a uniform **scaling** to a graph drawing changes the stress of the layout even though node positions remain the same [GHN12, KPS14, OKB16]. To obtain a fair comparison between the different strategies, we follow the same approach as in the paper by Simonetto *et al.* [SAK20]. We evaluate the stress of each layout on 40 candidate scaling values, computed as $(1.1)^i$ with $-20 < i < 20$, and finally select the one which yields the minimum average stress. All metrics are subsequently computed at that scale. For completeness, we report the scaling factor along the results of our experiments.

4.4. Implementation details

We implemented our *MultiDynNoS* approaches in the Java programming language (version 14) and ran the experiments on a laptop equipped with an i7-8750H CPU with 16GB of RAM. The static graph drawing algorithms (`fdp` and `sfdp`) that provide the initial placement come from the 'GraphViz' library [EGK*04]. To compute the different metrics, we used the same software as in [SAK20]. *MultiDynNoS* source code with the datasets included in this experiment is https://github.com/EngAAlex/MultiDynNos.

4.5. Experiment 1

As mentioned in the beginning of Section 4, in Experiment 1 we aim to assess the performance of *MultiDynNoS* over *DynNoSlice* and other timesliced-based techniques. For this experiment, we formulated the hypotheses stated in the following.

- **H1**: ***Drawing quality***. *MultiDynNoS* delivers comparable or better performance, in terms of drawing quality, to existing state-of-the-art event-based and timesliced dynamic graph drawing algorithms.
- **H2**: ***Scalability***. The multi-level approach allows *MultiDynNoS* to scale to larger datasets than *DynNoSlice* [SAK20].
- **H3**: ***MultiDynNoSversus flattening***. *MultiDynNoS* outperforms a static drawing of the flattened network in terms of average stress. In Simonetto *et al.* [SAK20], it was observed that flattening a dynamic graph and subsequently drawing it as a static one yield drawings of surprising quality.
- **H4**: ***Initial placement***. We hypothesize that by using a multi-level approach for first level placement (such as FM^3 [HJ04]), in place of a simpler force-directed technique (see Section 3.5), the whole layout process would be quicker and occasionally provide better quality metrics. Static multi-level graph drawing algorithms are designed to overcome some intrinsic limitations of their single level counterparts (see also Section 2), thus potentially providing a better initial placement.

To evaluate **H1**, we conduct a new experiment using the same graphs and procedures as in *DynNoSlice* [SAK20] and our previous paper [AMA21]. This allows us to compare *MultiDynNoS* to state-of-the-art dynamic graph layout algorithms, using a known and established procedure. We extend the experimentation with a new small sized temporal graph (RAMP) and three other event-based graphs with tens of thousands of events, in which we put *MultiDynNoS* scalability to the test (**H2**). In this experiment, we compare *MultiDynNoS* versus *DynNoSlice* and `sfdp flat` on temporal graphs only. We expect a noticeable improvement in terms of running times over *DynNoSlice*, while retaining (or improving) its layout quality. Testing **H3** justifies the inclusion of `sfdp flat` among the tested layout strategies. To evaluate **H4**, we selected two state-of-the-art static layout algorithms for initial placement: `fdp` (based on the Fruchterman and Reingold approach [FR91]) and `sfdp` [Hu05], which are single- and multi-level, respectively.

Results. Results are reported in Tables 4 and 5. We begin our discussion focusing on the data from Table 4, which comprises of the same graphs used in *DynNoSlice* paper [SAK20].

As a preliminary consideration, for the two layout strategies (`Visone` and DynNoSlice) tested in previous event-based graph drawing strategies [SAK17, SAK20], the results on timeslice-based data have been replicated. `Visone` has the best performance, or is competitive, when compared to DynNoSliceon, this data as it is a state-of-the-art algorithm for drawing timeslice-based dynamic graphs. DynNoSlicehas similar movement and crowding. The exception is InfoVis, where DynNoSlice performs better in terms of both types of stress and crowding. As previously discussed [SAK17], InfoVisis very similar to an event-based data where there are drastic changes between timeslices as authors in the visualization community rarely publish with the exact same set of authors in consecutive years.

The *MultiDynNoS* approaches on this data perform well. In terms of running time, they are competitive with `Visone` and can be an order of magnitude faster than *DynNoSlice* while retaining many of its advantages. The *MultiDynNoS* approaches have competitive levels of stress and crowding and consistently smaller amounts of movement. This shows that the improvements in scalability did not hurt the quality of the drawings when compared to the other strategies. The `sfdp flat` approach, that is flattening and drawing the dynamic graph with `sfdp`, is not able to perform very well in terms

Table 4: *Results of the experiment conducted on the same graphs as in the original* DynNoSlice *[SAK20] and our short paper [AMA21].* $|V|$ *and* $|E|$ *columns report the number of nodes and edges in the flattened graph.* $|E_v|$ *reports the number of events. For timesliced graphs, the number of timeslices is reported by its name in brackets. The* ***Type*** *column reports the tested algorithm. The* MultiDynNoS *variant used is presented as the combination of the initial placement layout (*`fdp` *or* `sfdp`*) and the coarsening/placement technique used.* T *column reports the algorithm running time in seconds.* ***Sc.(aling)*** *column reports the scaling value. Columns* On *and* Off *show the* StressOn *and* StressOff *values. Columns* M *and* C *represent Movement and Crowding, respectively;* D *reports the depth of the coarsened hierarchy. For every graph, we highlight the lowest values for Time, Stress, Crowding, Movement, and Depth to easily compare the different strategies.*

	$\|V\|$	$\|E\|$	$\|E_v\|$	Type	T (s)	Sc.	On	Off	M	C	D
VanDebunt(7)	39	32	104	Visone	**0.12**	1	1.14	1.46	3.79	0	-
				DynNoSlice	5.04	0.62	1.23	1.21	3.92	0	-
				sfdp flat	0.14	1.61	2.77	2.81	-	0	-
				fdp wi_id	1.10	0.68	1.23	1.26	0.94	0	5
				fdp is_gr	1.11	0.68	**1.04**	**1.07**	0.91	0	3
				fdp sm_sp	1.10	0.68	1.24	1.28	**0.89**	0	3
				sfdp wi_id	1.23	0.68	1.40	1.45	0.96	0	5
				sfdp is_gr	1.05	0.75	1.08	1.12	1.00	0	3
				sfdp sm_sp	1.04	0.75	1.38	1.43	1.01	0	3
Newcomb(15)	17	93	602	Visone	0.10	1	**14.04**	**14.76**	16.36	8	-
				DynNoSlice	7.58	0.68	16.60	16.57	13.44	1	-
				sfdp flat	**0.15**	1.33	26.54	26.52	-	**0**	-
				fdp wi_id	0.91	0.82	30.97	31.20	3.64	4	6
				fdp is_gr	1.00	0.82	20.54	20.45	2.67	1	3
				fdp sm_sp	0.95	0.82	21.11	20.94	**2.62**	2	3
				sfdp wi_id	0.96	0.82	27.68	27.66	3.38	2	6
				sfdp is_gr	0.88	0.82	21.54	21.44	2.91	3	3
				sfdp sm_sp	0.89	0.82	21.20	21.03	2.73	1	3
InfoVis(21)	1136	2506	2878	Visone	77.43	0.46	51.66	52.97	2.14	36	-
				DynNoSlice	224.93	0.56	30.14	30.19	2.03	2	-
				sfdp flat	**0.55**	1.33	105.29	102.87	-	1,253	-
				fdp wi_id	151.28	0.62	27.12	27.27	1.68	2	6
				fdp is_gr	98.36	0.62	27.51	27.42	1.65	2	4
				fdp sm_sp	116.15	0.56	29.71	29.46	**1.49**	3	3
				sfdp wi_id	153.77	0.62	**26.89**	**26.87**	1.65	**1**	7
				sfdp is_gr	96.40	0.62	27.72	27.54	1.62	2	4
				sfdp sm_sp	110.82	0.56	27.93	27.82	1.51	3	3
Rugby	12	66	3151	Visone	**0.07**	0.68	3.08	2.70	25.46	6	-
				DynNoSlice	2.84	0.51	1.86	**1.78**	6.64	0	-
				sfdp flat	0.18	0.90	2.07	2.02	-	0	-
				fdp wi_id	1.93	0.51	2.20	2.04	1.30	0	5
				fdp is_gr	5.02	0.51	2.05	1.86	1.42	0	2
				fdp sm_sp	1.23	0.56	**1.82**	1.82	**1.13**	0	2
				sfdp wi_id	1.19	0.51	2.24	2.02	1.31	0	5
				sfdp is_gr	1.20	0.51	2.06	1.92	**1.13**	0	2
				sfdp sm_sp	1.18	0.56	2.05	2.01	1.67	0	2
Dialogs	118	501	4033	Visone	3.39	0.17	**0.62**	0.87	5.44	682	-
				DynNoSlice	49.53	0.28	0.75	0.90	1.35	0	-
				sfdp flat	**0.21**	1	0.65	**0.69**	-	6	-
				fdp wi_id	8.56	0.31	0.69	0.88	0.74	**0**	15
				fdp is_gr	5.63	0.35	0.68	0.95	0.73	1	4
				fdp sm_sp	6.00	0.35	0.67	0.86	**0.68**	**0**	3
				sfdp wi_id	6.69	0.31	0.73	0.93	0.73	**0**	15
				sfdp is_gr	5.14	0.35	0.66	0.95	0.73	1	4
				sfdp sm_sp	5.94	0.35	0.74	0.97	0.72	**0**	3

of stress on these smaller datasets. However, it is a multi-level algorithm and its strengths are in terms of scalability. In static graph drawing, standard force-directed methods can be used on smaller graphs and the benefits of more scalable approaches are only realized with increased dataset size.

This behaviour can be also observed on the event-based data (Rugby and Dialogs). On Dialogs especially, running times have been vastly reduced and are now quite comparable to those of `Visone`, with competitive levels of stress. When comparing the three *MultiDynNoS* strategies, as expected, `MultiDynNoS wi_id`

Table 5: *Results of the experiment with four new graphs on the core* MultiDynNoS *algorithm.* $|V|$ *and* $|E|$ *columns report the number of nodes and edges in the flattened graph.* $|E_v|$ *reports the number of events. The* ***Type*** *column reports the tested algorithm. The* MultiDynNoS *variant used is presented as the combination of the initial placement layout (*`fdp` *or* `sfdp`*) and the coarsening/placement technique used.* T *column reports the algorithm running time in seconds.* ***Sc.(aling)*** *column reports the scaling value. Column* Off *shows the* StressOff *values. Columns* M *and* C *represent Movement and Crowding, respectively;* D *reports the depth of the coarsened hierarchy. For every graph, we highlight the lowest values for Time, Stress, Crowding, Movement, and Depth to easily compare the different strategies.*

	$\|V\|$	$\|E\|$	$\|E_v\|$	Type	T (s)	Sc.	*Off*	M	C	D
RAMP	874	802	802	`DynNoSlice`	53.51	0.16	**1.62**	**0.08**	904	-
				`sfdp flat`	**0.44**	1.94	2.81	-	3	-
				`fdp is_gr`	15.56	0.38	1.81	0.11	0	5
				`fdp sm_sp`	17.11	0.35	1.70	0.12	0	4
				`sfdp is_gr`	20.55	0.38	1.79	0.11	0	5
				`sfdp sm_sp`	26.84	0.31	1.71	0.10	0	4
MOOC	1166	8641	15k	`DynNoSlice`	497.04	0.29	1,604.23	**0.99**	4,266	-
				`sfdp flat`	**0.93**	1.46	**1,514.62**	-	**4,028**	-
				`fdp is_gr`	609.46	0.28	1,606.66	1.04	4,558	2
				`fdp sm_sp`	605.33	0.28	1,603.86	1.04	4,606	2
				`sfdp is_gr`	587.55	0.28	1,580.71	1.02	4,418	2
				`sfdp sm_sp`	494.56	0.31	1,545.25	1.14	3,759	2
MSG	882	4188	15k	`DynNoSlice`	2,350.19	0.21	**82,42**	1.50	2,143	-
				`sfdp flat`	**0.47**	1.77	83.27	-	**1,124**	-
				`fdp is_gr`	196.84	0.23	82.71	1.11	1,778	4
				`fdp sm_sp`	161,69	0.23	82.66	1.16	1,907	4
				`sfdp is_gr`	152.24	0.23	82.48	1.12	1,751	4
				`sfdp sm_sp`	154.95	0.21	84.91	**1.05**	2,314	4
RM	3822	4413	28k	`DynNoSlice`	-	-	-	-	-	-
				`sfdp flat`	**1.58**	4.17	**0.34**	-	**9,666**	-
				`fdp is_gr`	1,259.12	0.35	3.51	3.07	17,533	6
				`fdp sm_sp`	1,323.47	0.38	1.66	3.72	19,735	5
				`sfdp is_gr`	1,193.13	0.31	4.18	**2.62**	18,364	6
				`sfdp sm_sp`	1,313.38	0.28	0.66	2.80	31,532	5

(Walshaw) produces much deeper hierarchies compared to the others, with occasionally longer running times. In the previous version of the algorithm [AMA21], this strategy faced higher stress and crowding due to the lack of lower bounds for the layout parameters decay (see Section 3.6). With this issue solved in this version of the algorithm, this strategy proves to be much more competitive than before, while usually outperformed by the other variants (with the notable exception of InfoVis- at least on stress levels). The `MultiDynNoS is_gr` (GRIP) and `MultiDynNoS sm_sp` (FM^3) strategies perform similarly.

This experimental evidence supports our **H1** hypothesis: *MultiDynNoS* demonstrates comparable or better quality figures when compared to *DynNoSlice* and much more competitive running times, almost in par with `Visone` also on timesliced graphs.

We now move our discussion to Table 5. Here we test three new graphs (with up to 28 thousand events) and one small temporal graph. This last one was added since the other graphs of similar size included in the previous experiment [SAK20, AMA21] were only timesliced. Here we decided not to include `Visone` since, as we saw in Table 4, it does not perform well when drawing event-based graphs. For these graphs, computing stress metrics is a very time-consuming task, therefore, we only focused on *StressOff* (see Section 4.3), since all graphs are event-based, and excluded *MultiDynNoS* `wi_id` variant since it showed inferior performance overall on the initial set of graphs. We set a 2.5 h time limit for each algorithm to compute a drawing.

The data in Table 5 suggest that *MultiDynNoS* can be up to an order of magnitude faster than *DynNoSlice* also on these data sets with similar stress, movement and crowding. As expected, *MultiDynNoS* could always complete a drawing well ahead of the imposed time limit, while *DynNoSlice* failed to do so on one instance. On MOOC, *MultiDynNoS* and *DynNoSlice* provided very similar layouts in terms of stress, movement and crowding, as well as comparable running times: this is mostly due to the very shallow hierarchy produced by the coarsening (only two levels), greatly limiting the benefits of a multi-level strategy. On MSG and RAMP graphs, while the two drawings still present similar stress, *MultiDynNoS* proved to be faster than *DynNoSlice* by an order of magnitude on the former and by about 70% on the latter, presenting similar stress with better movement and crowding. Based on the available data, we can conclude that **H2** is confirmed.

When it comes to comparing *MultiDynNoS* to `sfdp flat`, the results in terms of average stress are less clear. `sfdp flat` has two advantages: its running times, due to the fact that the drawing is purely 2D, and therefore, not optimized in the space-time cube, and that it does not incur any movement. While it seems that flattening is a good approach in some circumstances, it provided worse stress performance than *MultiDynNoS* in all instances in Table 4 and on two out of four in Table 5. Concerning the former, we acknowledge that `sfdp flat` is a multi-level algorithm, and therefore, its strengths are in terms of scalability. In the latter, compared to *MultiDynNoS*, flattening provided better stress on MOOC and RM and better crowding in all instances except for RAMP. This suggests that to achieve less stress on RAMP and MSGnodes ‘need’ to move (which can lead to incidental overlapping) ultimately providing less stressed layouts. On MOOC, however, the least stress between the *MultiDynNoS* variants (`sfdp sm_sp`) was achieved with the highest movement, which may suggest that this layout as well might benefit from additional node motion. This behaviour is also visible on InfoVisgraph: since it is a time-dependant co-authorship network, it is a sequence of cliques representing the groups of authors that co-authored one or more papers each year. These cliques, however, change in size and composition at each timeslice, making the graph behave more like a temporal network. Therefore, we can conjecture that when node movement is expected, i.e. frequent change of neighbourhoods, temporal layout techniques might be preferable to flattening. On RM *MultiDynNoS* could achieve comparable amounts of stress, but with significantly higher crowding. Based on the available data, we can partially accept **H3**: *MultiDynNoS* can outperform flattening, but it strongly depends on the intrinsic properties of the graph.

Finally, we discuss the effect of the algorithm for first level placement. Our results do not suggest a strong correlation between the choice of the first level algorithm and the final quality of the drawing. However, in general `sfdp`- variants tend to be slightly quicker than the `fdp`-based ones, especially on the larger graphs in Table 5. Our conjecture is that when the coarsest level is so large or dense that the multi-level layout can accelerate the drawing process over a force-directed approach, then the overall running times are reduced by a significant amount, with also potential benefits on the final layout quality. Otherwise, since multi-level algorithms have their power in scalability, they do not make much difference when the coarsest graph is small, leading to having little to no impact on the final running times or potentially making the process slower compared to using `fdp`. Therefore, **H4** can only be partially accepted.

4.6. Experiment 2: Algorithm extensions

In this experiment, we investigate the effects of the proposed core algorithm extensions (see Section 3.7) on the same graphs and using the same quality metrics as in Experiment 1 (see Section 4.5).

We experiment with three different experimental setups: AutoTau versus ManualTau (setup **1**), ManualTau versus ManualTau with bend transfer (setup **2**) and AutoTau versus AutoTau with bend transfer (setup **3**). ManualTau represents *MultiDynNoS* core algorithm, as described in Section 3, using the pre-determined τ values also used in Experiment 1. AutoTau is for *MultiDynNoS* core algorithm but using the methodology described in Section 3.7.1 for computing the τ value to use during layout. In Setups **2** and **3**, ManualTau and AutoTau are complemented with our bend transfer extension, which extends the trajectories placement as described in Section 3.7.2.

For the same reasons as in Experiment 1, we only focus on the `is_gr` and `sm_sp` variants, especially to speed up the calculation of the quality metrics for the larger graphs. For the same reason, only *StressOn* is computed for the timesliced graphs and only *StressOff* for the event-based ones (see Section 4.3).

In order to simplify the results discussion, for each of the aforementioned setups, we gather the best results for each of the quality metrics across all *MultiDynNoS* variants (e.g. best time, lowest stress, lowest crowding, etc.). Results for setups **1**, **2** and **3** are reported in Tables 6–8, respectively. Full results are available as supplemental material. In Table 3, we report the ManualTau and AutoTau values used throughout the experiment.

Setup 1. In this setup, we compare ManualTau (**M** —the same version used in Experiment 1) versus AutoTau (**A**) (see Section 3.7.1), whose results are shown in Table 6 with the following hypothesis:

- AutoTau will be competitive in terms of drawing quality and running times over a specifically and experimentally calculated τ value (ManualTau).

When applying AutoTau, we observe a growing trend on running times, with the exception of Dialogs and Rugby. This is visible especially on MOOC. On this last graph, the value of AutoTau when compared to the selected ManualTau is 5 orders of magnitude greater (see Table 3), which leads to a much more stretched cube: this makes optimizing trajectories a more time intensive operation. Since the coarse hierarchy of that graph is very shallow, because of the layout tuning more trajectory optimization rounds will be performed, noticeably increasing the final running times. However, this value of AutoTau better encapsulates the nature of the graph, in fact stress values on this graph are substantially lower than **M**, at the expense of increased movement and crowding. In general, we observe that **A** provides low movement layouts on the smaller graphs, that tend to benefit from a more stable layout – in this, the exception of InfoVis is clearly visible. On RM, the smaller AutoTau also produces a layout with reduced movement, stress and crowding than **M**. On MSG, RAMP, and Rugby, movement is higher, but **A** still achieves better stress on last two.

Overall, the available data suggest the validity of our hypothesis for this setup: the AutoTau succeeds in providing layouts with competitive quality and can be used as a starting point for the search of a specific Tau value for the graphs. This comes at the expense of potentially longer running times.

Setup 2. We now evaluate the impact of ManualTau with bend transfer (**MB**) (see Section 3.7.2) over ManualTau without bend transfer (**M**) with the following hypothesis:

- We expect that **MB** will provide graphs with more movement, since bends are transferred from one level to the other, but with improved stress—especially on networks that require high movement (e.g. InfoVis).

In Table 7, we observe that running times are slightly increased: outliers are InfoVis, where there is the largest increase (around 30%

Table 6: *Comparison between* ManualTAU *(M), reporting the results from Tables 4 and 5, and* AutoTAU *(A). The best results across all* MultiDynNoS *variants for Time, Stress, Movement and Crowding are reported here. Stress column reports* StressOn *for timesliced graphs (highlighted here with an *) and* StressOff *for temporal graphs. Graphs are ordered with increasing number of events. For every comparison, we highlight the lowest value to easily find the best performing strategy.*

	Time (s)		Stress		Movement		Crowding	
Graph	M	A	M	A	M	A	M	A
VanDebunt*	1.04	**0.47**	**1.04**	1.20	0.89	**0.31**	0	0
Newcomb*	0.88	**0.34**	**20.54**	21.02	2.62	**0.39**	1	**0**
RAMP	**15.56**	43.77	1.70	**1.56**	**0.10**	0.62	0	0
InfoVis*	96.40	**90.39**	**26.89**	35.86	1.49	**0.11**	1	**0**
Rugby	1.18	**0.51**	1.82	**1.53**	**1.13**	1.38	0	0
Dialogs	5.14	**2.25**	0.86	**0.69**	0.68	**0.01**	0	0
MOOC	**494.56**	1,364.82	1,545.25	**1,415.40**	**1.02**	14.65	**3,759**	4,858
MSG	**152.24**	211.11	**82.48**	88.36	**1.05**	2.24	**1,751**	2,427
RM	**1,193.13**	1,322.71	0.66	**0.28**	2.62	**0.90**	17,533	**11,373**

Table 7: *Comparison between* ManualTAU *(M), reporting the results from Tables 4 and 5, and* ManualTAU *with bend transfer (MB). The best results across all* MultiDynNoS *variants for Time, Stress, Movement and Crowding are reported here. Stress column reports* StressOn *for timesliced graphs (highlighted here with an *) and* StressOff *for temporal graphs. Graphs are ordered with increasing number of events. For every comparison, we highlight the lowest value to easily find the best performing strategy.*

	Time (s)		Stress		Movement		Crowding	
Graph	M	MB	M	MB	M	MB	M	MB
VanDebunt*	**1.04**	1.06	**1.04**	1.18	**0.89**	0.96	0	0
Newcomb*	0.88	**0.84**	20.54	**19.89**	2.62	**2.61**	1	**0**
RAMP	**15.56**	22.72	1.70	**1.62**	0.10	0.10	0	0
InfoVis*	**96.40**	129.80	26.89	**25.43**	**1.49**	1.60	1	**0**
Rugby	1.18	**1.11**	1.82	**1.71**	**1.13**	1.31	0	0
Dialogs	5.14	**4.88**	**0.86**	0.88	0.68	**0.67**	0	0
MOOC	494.56	**440.68**	**1,545.25**	1,557.12	1.02	**1.01**	**3,759**	4,100
MSG	**152.24**	170.67	82.48	**82.09**	1.05	**1.03**	**1,751**	1,762
RM	**1,193.13**	1,222.15	0.66	**0.55**	2.62	**1.17**	17,533	**8,524**

more), and MOOC, where **MB** is faster than **M** by 50 s (10% of total running time).

A first partial confirmation to our hypothesis comes by the movement values on the smaller graphs, where **MB** also outperforms **M** in terms of stress on four instances out of six. On larger graphs the effect on movement is mitigated, this is visible especially on RM and possibly caused by the deeper hierarchies that slow down vertices max movement as layout gets to the finer levels. There are also benefits on drawing quality, slightly better stress on two instances out of three, and much better crowding on RM. We conclude that we can partially accept our hypotheses: bend placement increases movement and quality in some instances, but its effect is mitigated with deeper hierarchies due to the layout tuning.

Setup 3. In this setup, we evaluate the impact of AutoTau with bend transfer (**AB**) over AutoTau without (**A**) with the following hypothesis:

- As AutoTau tends to reduce movement, we expect bend placement will partially counteract this effect and improving stress.

In contrast to the results in Setup **2**, the bend transfer effect is mostly negligible (Table 8). Running times are comparable or slightly lower, with exception of InfoVis. In terms of stress, movement and crowding, all results of **AB** overlap closely with **A** with the exception of the crowding on RMgraph. This last result, however, comes with very high stress and movement (12.70 and 3.10, respectively—full results available in supplemental material). Therefore, we reject the hypothesis for this setup: the effect of AutoTau seems to greatly reduce the efficacy of bend placement, but slightly reduces running times on the larger instances.

Discussion. Our experiments tested the improvements brought by τ, the time to space conversion parameter, which maps time to the depth of the space-time cube, and bend transfer to lower levels of the hierarchy.

Our first setup tested automatically computed τ against a manually tuned value. AutoTau was introduced to increase the accessibility of our approach to wider audiences. In *DynNoSlice* [SAK17, SAK20], in fact, the user would need to estimate a τ value using a 'rule-of-thumb' and then experiment with that value to get a good layout. The purpose of AutoTau is to compute and assign a reasonable value for τ that allow users to simply use our implementation without having to manually tune this parameter, and/or to provide a starting point to engineer their own ManualTau for that specific dataset. In general, our experiments suggest that our computed AutoTau is 'stiff', meaning that provides layouts with low movement, for small data sets and, on the other hand, more flexible, i.e. for larger data sets.

In our second setup, we compared ManualTau against ManualTau with bend transfer. Bend transfer would normally increase the mobility of vertices in the dynamic graph by allowing higher levels of the hierarchy to influence node movement at lower levels. We observe that, generally, bend transfer improves stress at the expense of movement.

In our third setup, we compared automatically computed τ against automatically computed τ with bend transfer. Given that

Table 8: *Comparison between* AutoTAU *(A), reporting the results from Table 6, and* AutoTAU *with Bend Transfer (AB). The best results across all* MultiDynNoS *variants for Time, Stress, Movement and Crowding are reported here. Stress column reports* StressOn *for timesliced graphs (highlighted tup, we compare ManualT here with an *) and* StressOff *for temporal graphs. Graphs are ordered with increasing number of events. For every comparison, we highlight the lowest value to easily find the best performing strategy.*

Graph	Time (s)		Stress		Movement		Crowding	
	A	AB	A	AB	A	AB	A	AB
VanDebunt*	**0.47**	0.58	1.20	**1.05**	**0.31**	0.86	0	0
Newcomb*	**0.34**	0.40	21.02	**20.88**	**0.39**	2.47	0	0
RAMP	43.77	**43.45**	1.56	1.56	0.62	0.62	0	0
InfoVis*	**90.39**	209.08	**35.86**	36.78	**0.11**	0.12	**0**	1
Rugby	**0.51**	1.19	**1.53**	1.54	**1.38**	1.79	0	0
Dialogs	**2.25**	2.64	**0.69**	0.75	**0.01**	0.05	0	0
MOOC	1,364.82	**1,168.76**	**1,415.40**	1,417.96	14.65	**14.63**	4,858	**4,443**
MSG	211.11	**198.04**	**88.36**	89.24	2.24	**2.20**	**2,427**	2,486
RM	1,322.71	**1,181.37**	**0.28**	0.31	**0.90**	1.59	11,373	**1,081**

Auto Tau is stiff on small data sets, we thought that bend transfer would help improve this situation. Our experiment did not see this result. Therefore, we conclude that bend transfer would help improving the quality of the layout in combination with a manually defined τ rather than with AutoTau, as the latter mitigates the bend transfer effects to lower levels of the hierarchy.

Finally, it remains to be seen if these results generalise to other temporal network data sets.

5. Conclusion and Future Work

In this paper, we present *MultiDynNoS*, the first multi-level event-based graph layout algorithm. We expand on our previous preliminary work [AMA21] by providing a much more detailed algorithm description and an extended experimental evaluation with new and larger graphs. We also describe, implement and evaluate two extensions (AutoTau and Bend Transfer) to the core algorithm, which also uncover new potential research questions. Our experiments show that *MultiDynNoS* meets its design requirement of making event-based graph layout more competitive compared to existing timesliced [BW04] and event-based techniques [SAK20], while retaining its advantages in terms of drawing quality especially on temporal graphs. Overall, *MultiDynNoS* can be up to an order of magnitude faster than *DynNoSlice*, with the scalability improvements particularly visible on our experiments on large graphs, where *DynNoSlice* could not complete all layouts or was considerably slower with little to no differences in drawing quality.

Of our tested algorithm extensions, our new automatic Tau selection makes *MultiDynNoS* much more accessible. One of the most relevant obstacles when using *DynNoSlice* in fact, was the necessity of choosing an appropriate τ value when drawing a graph. Our experiments show that our AutoTau provides a reliable and reasonable compromise between drawing quality and running times, and also represent a good starting point for power users that would like to find an optimized τ value to enhance the final layout. Transferring bends from one level to the other also increased the influence that the coarse hierarchy has on the final layout. It increases the movement figures as we expected, and improves the quality figures on some graphs with a small cost in terms of running times.

This paper also provides some more evidence on the research question about when is more convenient to stick to timeslicing (or flattening) and when using event-based layout strategies would provide a significant advantage. We believe that graphs which share the same properties (i.e. rapidly changing neighbourhoods, see Section 4.5) of InfoVis to be of significant interest when investigating the graph properties that impact the final quality of the different algorithms, since that is a timesliced graph that performs better when drawn with event-based techniques.

Finally, we believe that this paper opens, amongst others, two interesting research directions for future work. First, the improved scalability and ease of use of *MultiDynNoS* over previous event-based layout methods might open the way to user studies aimed at uncovering the impact of the use these continuous time graph drawing techniques on user perception and performance when dealing with typical dynamic network visualization tasks. Second, integrating graph simplification and filtering techniques would allow *MultiDynNoS* and generally event-based layouts to scale even more by removing unnecessary noise and clutter. This would naturally pave the way of event-based layouts into visual analytics systems for decision making: temporal networks, in fact, can naturally model complex phenomena including contact-to-contact interaction, information dissemination over social networks, and physical proximity [HS12].

Acknowledgements

The authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme.

References

[AB20] Agarwal S., Beck F.: Set streams: Visual exploration of dynamic overlapping sets. *Computer Graphics Forum 39, 3* (2020), 383–391.

[ADLM18] Arleo A., Didimo W., Liotta G., Montecchiani F.: A distributed multilevel force-directed algorithm. *IEEE Transactions on Parallel and Distributed Systems 30, 4* (2018), 754–765.

[AMA07] Archambault D., Munzner T., Auber D.: TopoLayout: Multilevel graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics 13, 2* (2007), 305–317.

[AMA21] Arleo A., Miksch S., Archambault D.: A multilevel approach for event-based dynamic graph drawing. In *EuroVis 2021—Short Papers*. M. Agus, C. Garth and A. Kerren (Eds.). The Eurographics Association.

[AP12] Archambault D., Purchase H. C.: Mental map preservation helps user orientation in dynamic graphs. In *Proceedings of the International Symposium on Graph Drawing* (2012), Springer, pp. 475–486.

[AP16] Archambault D., Purchase H. C.: Can animation support the visualization of dynamic graphs?. *Information Sciences 330* (2016), 495–509.

[APP10] Archambault D., Purchase H., Pinaud B.: Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics 17, 4* (2010), 539–552.

[Arl] Arleo A.: MultiDynNoS code repository on GitHub. https://github.com/EngAAlex/MultiDynNos. Accessed: 24th June 2022.

[BBDW17] Beck F., Burch M., Diehl S., Weiskopf D.: A taxonomy and survey of dynamic graph visualization. *Computer Graphics Forum 36, 1* (2017), 133–159.

[BDA*17] Bach B., Dragicevic P., Archambault D., Hurter C., Carpendale S.: A descriptive framework for temporal data visualizations based on generalized space-time cubes. *Computer Graphics Forum 36, 6* (2017), 36–61.

[BGKM10] Bartel G., Gutwenger C., Klein K., Mutzel P.: An experimental evaluation of multilevel layout methods. In *Proceedings of the International Symposium on Graph Drawing* (2010), Springer, pp. 80–91.

[BM11] Brandes U., Mader M.: A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In *Proceedings of the International Symposium on Graph Drawing* (2011), Springer, pp. 99–110.

[BPF14] Bach B., Pietriga E., Fekete J. D.: GraphDiaries: Animated transitions and temporal navigation for dynamic networks. *IEEE Transactions on Visualization and Computer Graphics 20*, *5* (2014), 740–754.

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

[BW04] Brandes U., Wagner D.: *Analysis and Visualization of Social Networks* (pp. 321–340). Springer, Berlin, Heidelberg, 2004.

[CCM17] Crnovrsanin T., Chu J., Ma K.-L.: An incremental layout method for visualizing online dynamic graphs. *Journal of Graph Algorithms and Applications 21*, *1* (2017), 55–80.

[CP96] Coleman M. K., Parker D. S.: Aesthetics-based graph layout for human consumption. *Software: Practice and Experience 26*, *12* (1996), 1415–1438.

[Cra16] Crawford C. J.: Dynamic Multilevel Graph Layout and Visualisation. PhD thesis, University of Greenwich, 2016.

[DG02] Diehl S., Görg C.: Graphs, they are changing. In *Proceedings of the International Symposium on Graph Drawing* (2002), Springer, pp. 23–31.

[DGK01] Diehl S., Görg C., Kerren A.: Preserving the mental map using foresighted layout. In *Data Visualization 2001*. Springer, Vienna (2001), pp. 175–184.

[DSP*17] Du F., Shneiderman B., Plaisant C., Malik S., Perer A.: Coping with volume and variety in temporal event sequences: Strategies for sharpening analytic focus. *IEEE Transactions on Visualization and Computer Graphics 23*, *6* (2017), 1636–1649.

[EGK*04] Ellson J., Gansner E. R., Koutsofios E., North S. C., Woodhull G.: Graphviz and dynagraph–static and dynamic graph drawing tools. In *Graph Drawing Software*. Springer, Berling, Heidelberg (2004), pp. 127–148.

[EHK*03] Erten C., Harding P. J., Kobourov S. G., Wampler K., Yee G.: Graphael: Graph animations with evolving layouts. In *Proceedings of the International Symposium on Graph Drawing* (2003), Springer, pp. 98–110.

[EP06] Eagle N., Pentland A. S.: Reality mining: Sensing complex social systems. *Personal and Ubiquitous Computing 10*, *4* (2006), 255–268.

[FQ11] Farrugia M., Quigley A.: Effective temporal graph layout: A comparative study of animation versus static display methods. *Journal of Information Visualization 10*, *1* (2011), 47–64.

[FR91] Fruchterman T. M., Reingold E. M.: Graph drawing by force-directed placement. *Software: Practice and Experience 21*, *11* (1991), 1129–1164.

[FT08] Frishman Y., Tal A.: Online dynamic graph drawing. *IEEE Transactions on Visualization and Computer Graphics 14*, *4* (2008), 727–740.

[GDBG12] Gorochowski T. E., Di Bernardo M., Grierson C. S.: Using aging to visually uncover evolutionary processes on networks. *IEEE Transactions on Visualization and Computer Graphics 18*, *8* (2012), 1343–1352.

[GHN12] Gansner E. R., Hu Y., North S.: A maxent-stress model for graph layout. *IEEE Transactions on Visualization and Computer Graphics 19*, *6* (2012), 927–940.

[GK00] Gajer P., Kobourov S. G.: Grip: Graph drawing with intelligent placement. In *Proceedings of the International Symposium on Graph Drawing* (2000), Springer, pp. 222–228.

[GWMG16] Grayson S., Wade K., Meaney G., Greene D.: The sense and sensibility of different sliding windows in constructing co-occurrence networks from literature. In *Proceedings of the International Workshop on Computational History and Data-Driven Humanities* (2016), Springer, pp. 65–77.

[HJ04] Hachul S., Jünger M.: Drawing large graphs with a potential-field-based multilevel algorithm. In *Proceedings of the International Symposium on Graph Drawing* (2004), Springer, pp. 285–295.

[HS12] Holme P., Saramäki J.: Temporal networks. *Physics Reports 519*, *3* (2012), 97–125.

[Hu05] Hu Y.: Efficient, high-quality force-directed graph drawing. *Mathematica Journal 10*, *1* (2005), 37–71.

[IHK*16] Isenberg P., Heimerl F., Koch S., Isenberg T., Xu P., Stolper C. D., Sedlmair M., Chen J., Möller T., Stasko J.: vispubdata. org: A metadata collection about IEEE visualization (vis) publications. *IEEE Transactions on Visualization and Computer Graphics 23*, *9* (2016), 2199–2206.

[KPS14] Kobourov S. G., Pupyrev S., Saket B.: Are crossings important for drawing large graphs?. In *Proceedings of the International Symposium on Graph Drawing* (2014), Springer, pp. 234–245.

[KZL19] Kumar S., Zhang X., Leskovec J.: Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), ACM, pp. 1269–1278.

[LAN19] Lee A., Archambault D., Nacenta M.: Dynamic network plaid: A tool for the analysis of dynamic networks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (2019), pp. 1–14.

[LAN21] Lee A., Archambault D., Nacenta M. A.: The effectiveness of interactive visualization techniques for time navigation of dynamic graphs on large displays. *IEEE Transactions on Visualization and Computer Graphics 27*, *2* (2021), 528–538.

[LHS*15] Liu Q., Hu Y., Shi L., Mu X., Zhang Y., Tang J.: Egonetcloud: Event-based egocentric dynamic network visualization. In *Proceedings of the 2015 IEEE Conference on Visual Analytics Science and Technology (VAST)* (2015), IEEE, pp. 65–72.

[LVM18] Latapy M., Viard T., Magnien C.: Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining 8*, *1* (2018), 61.

[MELS95] Misue K., Eades P., Lai W., Sugiyama K.: Layout adjustment and the mental map. *Journal of Visual Languages & Computing 6*, *2* (1995), 183–210.

[MGM*19] McGee F., Ghoniem M., Melançon G., Otjacques B., Pinaud B.: The state of the art in multilayer network visualization. *Computer Graphics Forum 38*, *6* (2019), 125–149.

[MLL*13] Monroe M., Lan R., Lee H., Plaisant C., Shneiderman B.: Temporal event sequence simplification. *IEEE Transactions on Visualization and Computer Graphics 19*, *12* (2013), 2227–2236.

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

[MMB*] Mohr S., Matthews L., Brett S., Mano V., Nonweiler J., Takahashi R., Townsend E.: Contact Tracing Model. https://github.com/ScottishCovidResponse/Contact-Tracing-Model. Accessed: 18th September 2020.

[New61] Newcomb T. M.: The acquaintance process as a prototype of human interaction. In *The Acquaintance Process*. Holt, Rinehart & Winston (1961). pp. 259–261.

[OKB16] Ortmann M., Klimenta M., Brandes U.: A sparse stress model. In *Proceedings of the International Symposium on Graph Drawing and Network Visualization* (2016), Springer, pp. 18–32.

[POC09] Panzarasa P., Opsahl T., Carley K. M.: Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology 60*, *5* (2009), 911–932.

[PS21] Perri V., Scholtes I.: HOTVis: Higher-order time-aware visualisation of dynamic graphs. In *Proceedings of the International Symposium on Graph Drawing and Network Visualization* (2021).

[SA06] Shneiderman B., Aris A.: Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics 12*, *5* (2006), 733–740.

[SAK17] Simonetto P., Archambault D., Kobourov S.: Drawing dynamic graphs without timeslices. In *Proceedings of the International Symposium on Graph Drawing and Network Visualization* (2017), Springer, pp. 394–409.

[SAK20] Simonetto P., Archambault D., Kobourov S.: Event-based dynamic graph visualisation. *IEEE Transactions on Visualization and Computer Graphics 26*, *7* (2020), 2373–2386.

[She94] Shewchuk J. R.: An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Tech. Rep., 1994.

[VDBVDS99] Van De Bunt G., Van Duijn M., Snijders T.: Friendship networks through time: An actor-oriented dynamic statistical network model. *Computational & Mathematical Organization Theory 5* (1999), 167–192.

[Vel07] Veldhuizen T. L.: Dynamic multilevel graph visualization. *arXiv preprint arXiv:0712.1549 [cs.GR]* (2007).

[WAH*19] Wang Y., Archambault D., Haleem H., Moeller T., Wu Y., Qu H.: Nonuniform timeslicing of dynamic graphs based on visual complexity. In *Proceedings of the 2019 IEEE Visualization Conference (VIS)* (2019), IEEE, pp. 1–5.

[Wal03] Walshaw C.: A multilevel algorithm for force-directed graph-drawing. *Journal of Graph Algorithms and Applications 7*, *3* (2003), 253–285.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Supporting Data S1

Supporting Data Video S1