Universität Konstanz

Chair of Computer Graphics and Media Design
Department of Computer Science

Master Thesis

Graph Comics: Interactive Staging for Exploring Dynamic Graphs

for obtaining the academic degree
Master of Science (M.Sc.)

Field of study:	Information Engineering
Focus:	Graph Drawing

by

Josua Krause
(01/697048)

First advisor:	Prof. Dr. Oliver Deussen
Second advisor:	Assist. Prof. Dr. Enrico Bertini

Konstanz, March 23, 2014

Declaration of Authorship

The author of this work hereby declares that

- the present work is the result of his own work, without help from others and without using anything other than the named sources and aids;
- the texts, illustrations and/or ideas taken directly or indirectly from other sources (including electronic resources) have without exception been acknowledged and have been referenced in accordance with academic standards.

The Author wants to gratefully acknowledge supervision and guidance he has received from Hendrik Strobelt.

Konstanz, March 23, 2014

Josua Krause

Abstract

Analyzing dynamic graphs is complex. Using node link diagrams to display the graphs and its changes originates many challenges. This thesis introduces a novel interactive visualization technique for dynamic graphs by splitting complex graph transitions into simpler sub-transitions called stages. Transitions can be split by subsets of changing nodes, clusters in the graph, or by dividing the transition into stages of animation phases: edge removal, edge creation, and node movement. The created sub-transitions then can be reordered or merged again for the purpose of examining interesting changes in the graph. Like in a comic, the sub-transitions can be used to tell the story of interesting graph changes in a series of panels. This makes it easier to gain insight into the overall changes. Additionally, animation can be used to deepen the understanding of transitions. Furthermore, in this thesis I discuss how static node-link representations of dynamic graphs can convey the changes of a transition by using annotations. A prototype used on example graphs shows the feasibility of those approaches in the exploration of dynamic graphs.

Contents

1. Introduction

Dynamic graphs occur in many application domains. In the sciences, natural processes can be modeled as dynamic graphs that reflect changes in relationships between entities and therefore change a graph over time. For example nodes in a computer network can establish new connections to other nodes or disconnect from them. A railway infrastructure can change when tracks are closed for maintenance or a new line is opened. Likewise, a distribution network for resources like electricity or gas is dependent on construction or closing of supply lines. On international scale those networks often change due to political or economic reasons. In social networks people can become friends or terminate a friendship.

Analyzing dynamic graphs is a challenging task and techniques used for static graphs cannot be adapted for the dynamic case in a straight forward way. Applying layout algorithms for static graphs to dynamic graphs either create layouts that are incorrect and lead to false assumptions about the graph, or layouts that are correct but create strong node movements that are hard to follow. A compromise in layout quality has to be made which still leads to complex and hard to analyze transitions. In this thesis I present a novel technique to interact with dynamic graphs. By splitting complex transitions into smaller manageable sub-transitions insights about the changes in the graph can be gained.

In Chapter 2 we discuss different approaches to address the problem of analyzing dynamic graphs. After that, in Chapter 3, we further explore why properties of static graphs cannot be easily adapted for dynamic graphs. Choosing a node-link representation for start and end stages of dynamic graphs we can use edge outlines to prevent problems with edge crossings. Visual annotations in form of pin shapes show the movement of nodes due to layout changes and colors indicate emerging and removing edges. Those visual cues were inspired by a preliminary study.

The transition of a graph can be split up into three phases, removing edges, creating edges, and moving the nodes. Or, following the example of comics, complex transitions can be split into more granular steps that are easier to understand. In the described prototype, those splits can be done either automatically, using clusters in the graph as base for which changes happen in which sub-transition, or by selecting those sets manually via the user interface. In order to give a user flexibility, sub-transitions can

be reordered or merged again. Since only a subset of nodes is going to be changed per sub-transition closures around the affected nodes are used to find the relevant nodes quickly, and, with the help of colors, their end position in the next sub-transition is also indicated by closures around the nodes.

A zoom-able user interface is used to let a user navigate the created comic-strip and get either an overview over all steps or zooming in on one sub-transition in order to examine it. The created comic-strips give insight in the changes of the dynamic graph from one step to the next and help presenting those findings to others.

In addition to the static representation of the sub-transitions of the dynamic graph, the user can use animation to see the changes from one sub-transition to the next. The animated transition is automatically split into the three aforementioned phases, edge removal, edge creation, and node movement. With the help of a slider, the user always has the possibility to interact with and interrupt the animation.

Chapter 4 describes how to make the zoom-able user interface intuitively usable with zooming to the position of the mouse cursor and zooming to a given rectangle on the screen. Furthermore, in order to create a responsive user interface, performance optimizations for rendering need to be applied. When trying to animate many items at once we show how to overcome some mistakes of a quick implementation, like inconsistent positions during the transition. With the help of fork join pools we can minimize the time spent in the actual computation of the animation by utilizing multiple CPU cores and parallelism.

We further present different approaches to create iso-surfaces used to enclose sets of nodes: Bubble-sets, a widely used technique, and Kelp-like diagrams. Kelp diagrams are an aesthetic way of creating closures for sets. A simplified version is used in the prototype. Those Kelp-like diagrams are easier to create than Bubble-sets, due to less complex requirements and provide a cleaner and more coherent look.

The application of the prototype is shown in three case studies in Chapter 5. We demonstrate the prototype on three dynamic example graphs. The first graph is a small manually created graph. The second and third graphs represent social networks. The graphs show who is in face-to-face contact with whom during a day at a conference and how the friendship between girls change over time. The graphs are analyzed via the prototype by creating comic resembling stories that convey interesting changes.

In Chapter 6 we discuss which changes in a dynamic graph are potentially interesting for analyzing and how the prototype can be used to create series of sub-transitions of a dynamic graph to gain insight in those changes. We also discuss possible drawbacks of the prototype and the splitting technique, like loss of context for too granular changes, and falsely interpreting sub-transitions as valid intermediate steps. Furthermore, we present possible extensions to the prototype as future work.

2. Related Work

Optimizing layout algorithms for dynamic graphs is an abundant research focus. Brandes and Wagner [7] define the main goal as a consistent, stable, and readable dynamic graph layout. Friedrich and Eades [14] achieve this by preserving the mental map of the reader in between time steps. Reducing node movements also lead to this preservation (Lee *et al.* [23]). However, Purchase and Samra [27] states that preserving the mental map can lead to false assumptions about the graph in some cases. Also, the layout performance is decreased in later time steps which can be mitigated by using a compromise between mental map preservation and good layouts as suggested by Saffrey and Purchase [29]. An example of this is given by Brandes and Mader [6] who use a weighted combination of a global layout with local layouts.

Animation is a widely used technique for dynamic graph drawing, used for example by Brandes *et al.* [5]. Yee *et al.* [35] use interpolation in polar coordinates to transition dynamic graphs in a radial layout. To preserve the mental map during the transition, Friedrich and Eades [14] animate with non-linear interpolation. This avoids non-existing structures, and communicates the reason behind the layout change.

Principles of animated presentation are described by Zongker and Salesin [36]: In contrast to exaggeration of movements for creating vividness in traditional animation as reported by Lasseter, [22] only meaningful movements should ideally be shown. Slow-in slow-out timing should be used, as further investigated by Dragicevic *et al.* [12]. Multiple simultaneous movements are difficult to follow, therefore only one movement at a time should be shown. The principle of showing only one movement is difficult to obey, although Robertson *et al.* [28] shows that similar moving objects can still be followed correctly. Tversky *et al.* [32] argues that animation is only useful when there is a natural connection to the underlying change and the animation itself is performed slow and clear enough to be perceived easily.

Furthermore, splitting animations can be beneficial. Heer and Robertson [19] use staging of animations to keep them perceivable . DOITrees [9][18] and SpaceTree [26] both use focus+context tree exploration systems, by utilizing fish-eye view and zooming interfaces. The animation of tree changes is decomposed into trimming branches, adjusting the layout, and growing new branches. Bach *et al.* [2] with

GraphDiaries use staging on general dynamic graphs by splitting graph transitions in removing edges, moving nodes, and creating new edges. Edge changes are highlighted by color.

As alternative to animation, dynamic graphs can be represented as small multiples. Baudisch *et al.* [3] compare animation and small multiples to visualize dynamic graphs and remark that static depictions of motion perform at least as well as animation, while helping to process changes better. Archambault *et al.* [1] further state that small multiples are faster to interpret, while animation has a lower error rate. Von Landesberger *et al.* [33] find that animation is better for grasping larger changes, whereas a static representation enables understanding of more detailed changes. Beck *et al.*[4] recently proposed a hybrid system of animation and static representation by applying Rapid Serial Visual Presentation to parallel edge splatting of dynamic graphs.

While related work has addressed layout, animation, small multiples, and even RSVP for exploring dynamic graphs. To the best of the authors knowledge no approach has proposed a staging into sub transitions or the use of visual annotations as described in this thesis.

3. Design

In this chapter we discuss the design principles that were used for the prototype. After a formal definition of dynamic graphs in Section 3.1 and Section 3.1.1 we explore properties that help analyzing static graphs (see Section 3.1.2). Due to the limitations of those properties we conclude that a graphical representation of graphs as node-link diagrams is necessary and transfer this representation to dynamic graphs (see Section 3.1.3). In Section 3.2 we further discuss how to make this visual representation easier to read and interpret. To reduce complexity of a graph transition those changes can be split into its phases, removing edges, creating edges, and moving nodes, as described in Section 3.3. A preliminary study of visualizing changes in graphs in Section 3.4.1 and traditional comics splitting a complex process into smaller steps in Section 3.4.2 serve as inspiration for the idea of staging graphs by splitting the transition into less complex sub-transitions and lead to the panel design described in Section 3.4.3.

3.1 Dynamic graphs

A dynamic graph is a series of graphs $G_t = (V_t, E_t)$ where t is a discrete time step. The set of nodes V_t do not need to be same for all time steps. However, there is no difference between isolated nodes and non-existent nodes for most interpretations. Therefore the set of nodes V_t can be replaced with $V = \bigcup_t V_t$.

3.1.1 Changes in graphs

Since we ignore changes in the set nodes between time steps by isolating nodes that are otherwise removed, the only changes left to analyze in a dynamic graph are edge changes. There can be two types of edge changes, the creation of a new edge and the removal of an existing edge. This information is enough to describe a dynamic graph. Given an initial state $G_0 = (V, E_0)$ every successive step can be computed via $E_{t+1} = (E_t \cup E_t^{create}) \setminus E_t^{remove}$.

When a dynamic graph is represented as node-link diagram a third change between time steps exist. Depending on the type of layout strategy for the dynamic graph nodes may change their position which results in their movement. This can be

avoided by using fixed positions for the nodes. However, this results in reduced readability for at least some time steps of the graph in which the nodes are not laid out optimally. A more detailed discussion about layouts for dynamic graphs can be found in Section 3.1.3.

3.1.2 Interesting properties

In the analysis of static graphs there are a number of properties of nodes or sets of nodes. Centrality measures, like degree, betweenness, or closeness, describe the importance of one node within the graph. This can mean how influential a person is in a social network, or how likely traffic goes through a node of a computer network.

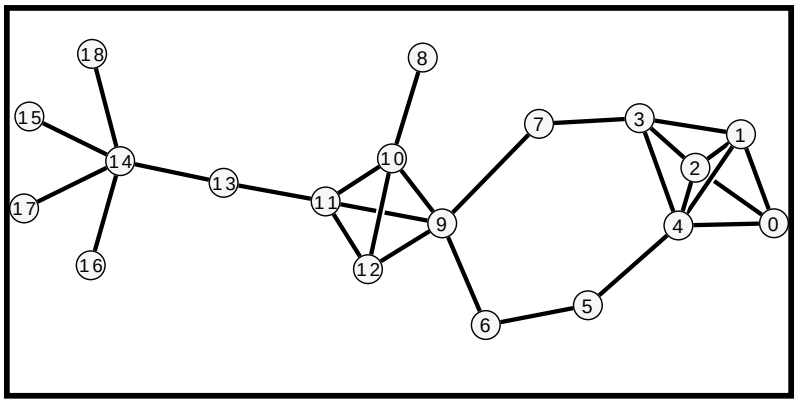


Figure 3.1: An example graph. The nodes with the highest degree (4, 9, and 14) are also nodes with a high betweenness. Betweenness indicates that in order for information to flow through the graph (i. e. diseases that spread through a social network) those nodes are important to connect parts of it. Removing them would have a great impact of how information gets distributed. The betweenness is also high for the three bridges (4–5–6, 3–7–9, and 11–13–14). Closeness of nodes is higher in the central cluster (8, 9, 10, 11, 12) which means that information originating in those nodes spreads faster than in other nodes. The nodes 14, 15, 16, 17, and 18 do not form a full cluster because they are only connected to each other via node 14.

Sets of nodes can roughly be categorized as clusters or bridges, with clusters being a group of nodes that are highly connected to each other and bridges being paths of nodes that connect two clusters. Within clusters, information is easily shared or diseases spread quicker. However, without bridges the information or disease stays in one cluster and cannot reach another. Nodes that connect multiple clusters are important to connect e. g. people of different groups in social networks. If those nodes are removed, a connection becomes more difficult. Clusters on the other hand can handle removed nodes more easily.

Those properties cannot be applied to dynamic graphs in a straight forward way. For example, given a bridge connecting two clusters. If this bridge is removed in a transition, but another bridge between the clusters is created, the relationship between the clusters stay the same. If the edge density within a cluster decreases in a transition the former cluster might not be considered a cluster anymore in the next step. Likewise, nodes can gain connections so that they can be considered as clusters in the next step. In the analysis of a dynamic graph it is therefore important to see how connections in the graph change overall, within clusters, and between clusters.

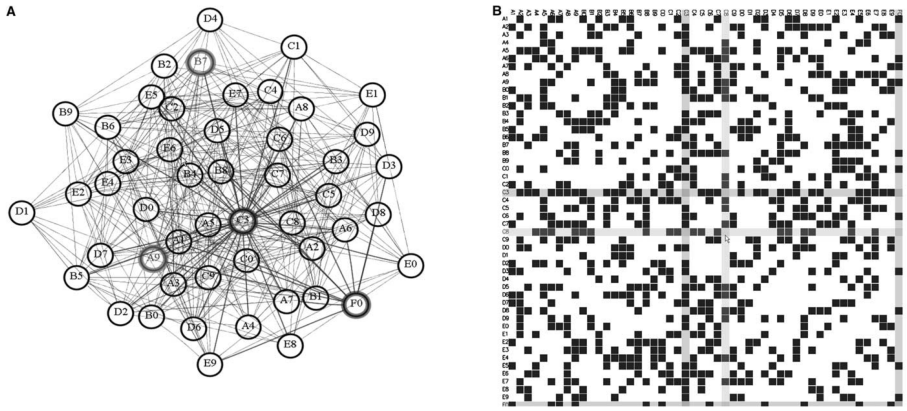


Figure 3.2: The same graph presented as node-link diagram and matrix. According to Ghoniem *et al.* [17], statistical features are easier to read in the matrix representation while following paths is easier in the node-link diagram. Note that the quality of the node-link diagram suffers because of the graph becoming a "hair-ball".

3.1.3 Visualizing graphs as node-link diagrams

A good visual representation of a graph is in form of an adjacency matrix. Many statistical features of the graph are easily visible and the matrix can be reordered to show clusters. In those areas adjacency matrices perform better than node-link diagrams (see Ghoniem *et al.* [17]). However, adjacency matrices are inferior to node-link diagrams with respect to tracing paths, scalability, and presenting additional properties of nodes. Note, that the scalability of node-link representations is limited as well, since larger graphs tend to become "hair-balls", and that showing additional properties in a node-link diagram can make the visualization of the data more difficult to read because of information overload.

For creating a node-link representation the graph nodes need a position. These positions can be generated by a layout algorithm. For general static graphs a common layout strategy is to try to map the graph distance of the nodes to spacial distance. However, the graph distance cannot generally be represented correctly. This error in distances is called stress. A popular metaphor for creating a node layout with low stress is to represent edges as springs and model nodes as electrons with a repulsive force towards each other. This leads to groups of nodes forming visual clusters while different groups get pushed away from each other. The metaphor led to the use of multidimensional scaling with the help of stress-majorization (e. g. Gansner *et al.* [16] or Khoury *et al.* [20]) for layout creation which behaves similar to simulation based approaches.

In the case of dynamic graphs, creating optimal layouts for every time step leads to strong movements of the nodes which is not desired. On the other hand creating a layout considering all time steps at once leads to a rather sub-optimal layout but no movement at all. Having no movement of nodes makes it easier to find nodes in different time steps. However, sub-optimal layouts can lead to false assumptions about the topology of a graph. For example, a group of nodes could form a visible cluster in the averaged layout, even though this group may not be connected in a given time step. Brandes and Mader [6] provide a compromise between robust layouts with little motion and optimal layouts per time step. Their findings have been used to create the layouts for the example graphs.

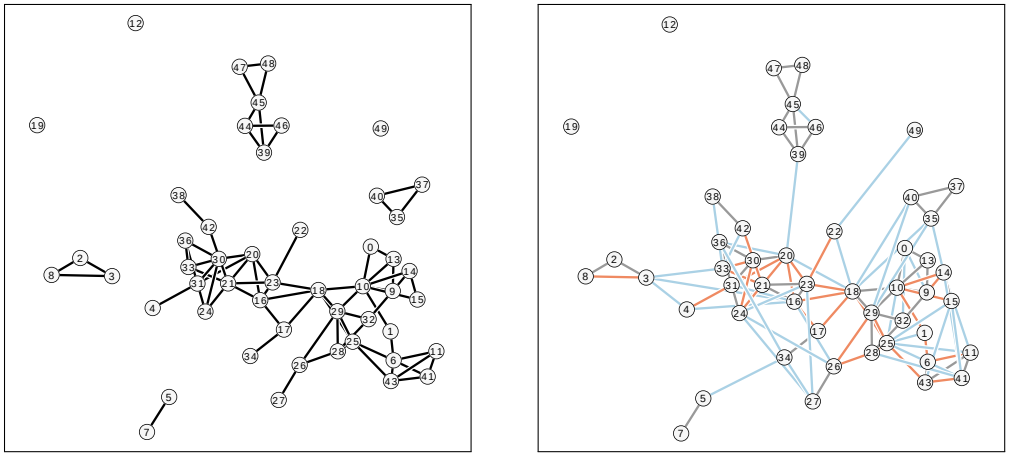


Figure 3.3: Design of the node-link representation. On the right edges are colored to indicate their change until the next step. Blue edges do not exist yet. Note the outlines of the edges which help to distinguish edges that are crossing each other.

3.2 Improvements for dynamic node-link diagrams

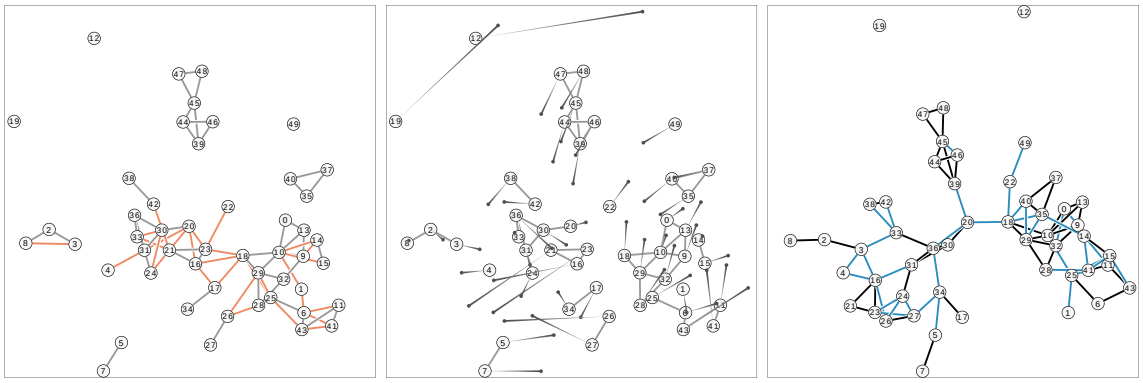
Besides laying out nodes in an intuitively meaningful way node-link diagrams have some graphical challenges. Given a non-planar graph, we have to deal with edge crossings. A large number of edge crossings can make it difficult to see which nodes are connected by an edge. Some geometrically correct approaches exist to avoid drawing edge crossings by terminating a line before it crosses another edge and starting it again before the end of the edge. If both drawn segments of an edge have the same length it is easier to connect them via closure, allowing to only hint edges (see Bruckdorfer *et al.* [8]). However, for our prototype it is sufficient to add a small outline when drawing edges (See Figure 3.3). This way we can identify both crossing edges more easily. Edges that are going through other nodes without connecting to them is a problem for the layout strategy and is much harder to solve.

Color can be used to hint at edges that are newly created or will be removed. Blue and orange for creating and removing respectively can easily be understood without having a color key in the panel. This encoding is also used by Bach *et al.* [2]. The color combination can also perceived by color blind persons without problems (See Figure 3.3). Easing the transition from a visible edge to a non existent edge (or vice versa) can be done by fading edges out or in. This also makes the color key more easily understandable.

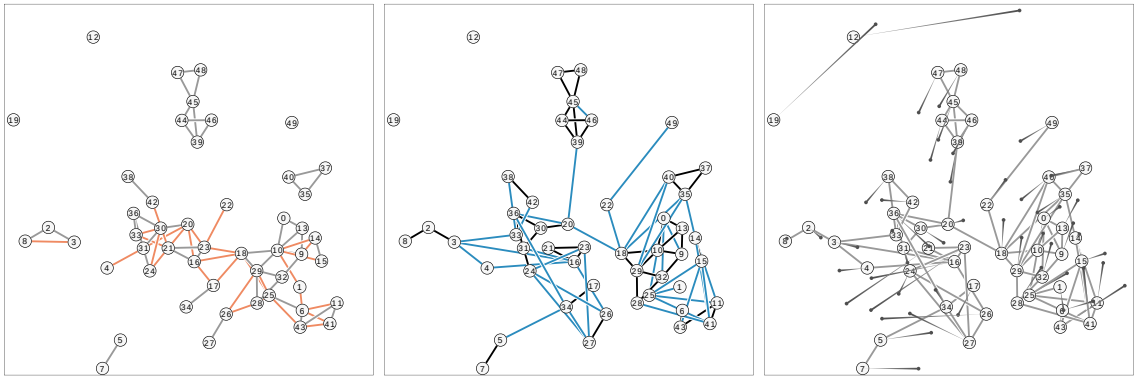
(a) Arrow Movement (b) Dashed Movement (c) Pin-shaped Movement

Figure 3.4: Three strategies of showing node movement. (a) shows the movement as arrow, (b) as dashed line with indication of the final position which is inspired by Figure 3.6a, and (c) a pin shaped design using findings of Ware *et al.* [34]. The direction of the movement is encoded by width and opacity. The end position is indicated as well.

The movement of nodes in a node-link diagram can be represented as trace between starting and end point. The most straightforward way of showing the movement of the nodes is to use an arrow with the tip pointing to the end of the movement. However, arrows only give a sense of direction and strength of a movement and don't convey very well that the tip of the arrow is the actual end position of the node. Besides that it is not clear whether the tip should be at the center of the node, at the beginning, or at the end. Changing the tip of the arrow into the shape of the moved node shows the end point of the transition clearly and defines the motion to be finite. For our prototype, we used findings of Ware *et al.* [34] stating, that opacity and width changes of lines can be used to represent direction and speed of movements resulting in a pin shape. This shape has a circle as end to depict the final position of the node and gets thinner towards the beginning of the motion (See Figure 3.4). The pin-shaped movement indicator is used in the prototype.



(a) Remove edges, move nodes, and create edges.



(b) Remove edges, create edges, and move nodes.

Figure 3.5: Two ways of splitting a transition into stages of animation phases. (a) moves the nodes before creating the new edges. In (b) new edges are created before the movement and therefore make the movement predictable when interpreting edges as springs. The movement of the two nodes at the top that are not connected to anything are an artifact of the layout algorithm which tries to position them in the largest free areas.

3.3 Staged graph changes

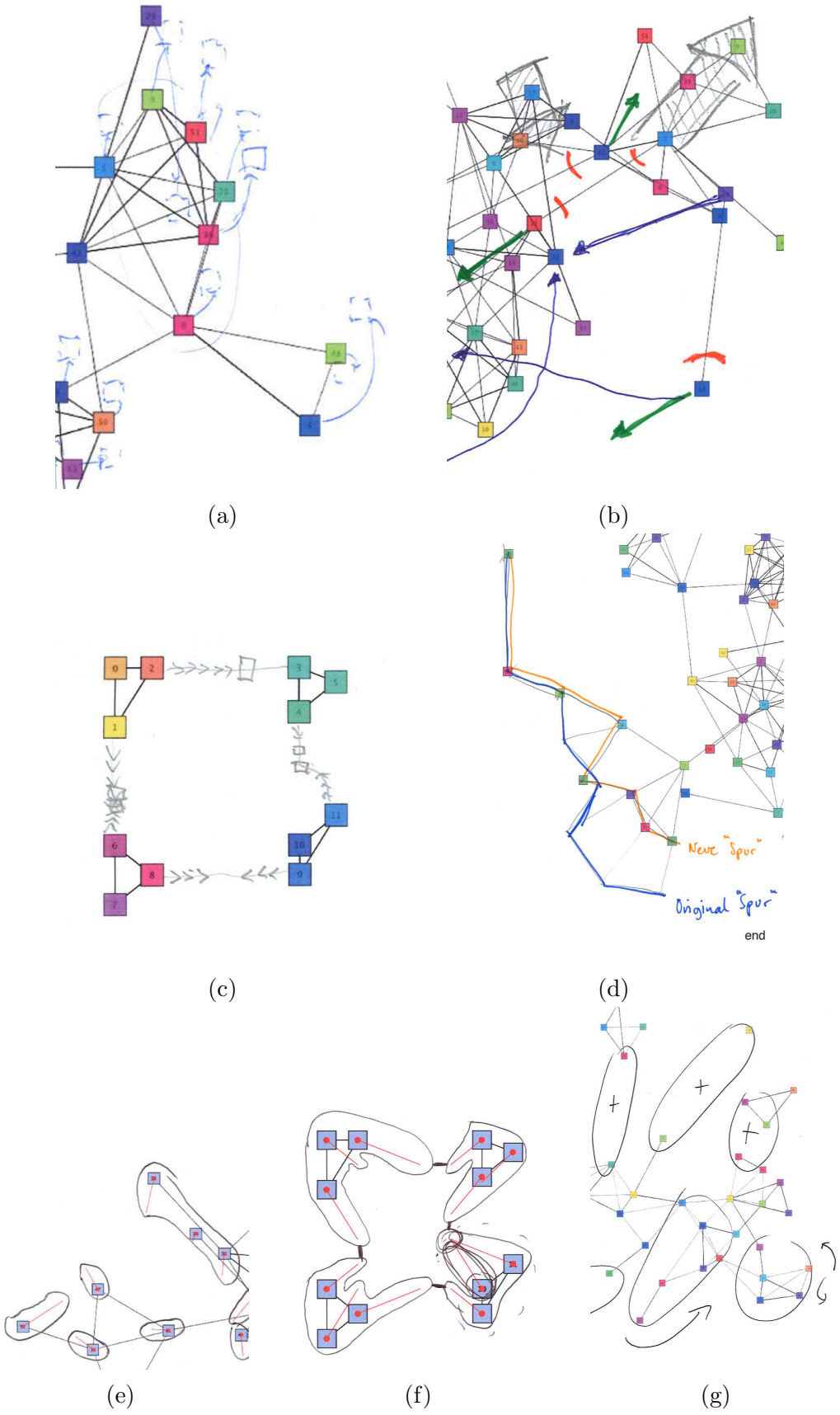
A popular way of showing changes in a dynamic graph is by animating the transition from one time step to the next (see Section 3.5.6 for the discussion of animation). This creates the problem that, when done naïvely, edge changes and node changes are shown at once, making it hard to follow the animation. A common technique to keep animation understandable is by splitting it into transition stages (see Heer and Robertson [19]). This can be applied for a graph transition by splitting it into three phases for edge and node changes. An example for this technique is used by Bach *et al.* [2], first removing the obsolete edges, then moving the nodes, and finally creating new edges. This technique, first used by Friedrich and Eades [14], can be improved by reordering the stages into edge removal, edge creation, and then node movement which is illustrated in Figure 3.5. The reordering introduces temporarily more edge crossings and a worse layout due to nodes having the position of the previous time step but the graph having edges of the next time step, which was what Friedrich and Eades tried to avoid. However, the spring metaphor of the layout strategy becomes more apparent when reordering the stages. With this the newly created edges pull the nodes to their correct position in the final movement phase. This makes the actual movement of nodes seem more logical, since a user can see why nodes are moving and can, to a certain extend, predict where the nodes are going to. The prototype uses the order: edge removal, edge creation, and node movement.

3.4 Design inspiration

The following preliminary study (Section 3.4.1) and inspiration from traditional comics splitting complex actions into manageable steps (Section 3.4.2) introduce the ideas of staging graphs by splitting the overall transition into sub-transitions and lead to the design of transition panels described in Section 3.4.3.

3.4.1 Preliminary study

Prior to designing a prototype for dynamic graph analysis we conducted a qualitative study in order to find out what humans consider to be helpful when exploring a dynamic graph. We asked participants to enrich a given transition of a dynamic graph, given in the form of small multiples, to show its changes. Four graph-drawing experts and four non-experts received the graphs printed on paper and were asked to make visual annotations with pen on the printouts. Each participant got a simpler and a complex graph randomly chosen from two variants. The simpler graphs were two variants containing twelve nodes arranged in four groups which changed from being unconnected to being connected. The larger graphs where taken from the SIENA data set described in Section 5.3. The transitions were 2 variations consisting of two time steps. For each task we provided two images: the graph before and after the transition. For easier navigation the nodes were labeled with numbers and colored the same in both time steps. Additionally, half of the participants got an image depicting the graph before the transition enriched with small red lines depicting the node movement towards after the transition. No restrictions or rules where given for the execution of the task, neither in use of material nor in use of time.



(a) (b) (c) (d) (e) (f) (g)

Figure 3.6: Manually annotated graph changes as result of our pen and paper study. Annotations in Figures (a) – (d) mostly use line variations, while those in Figures (e) – (g) rely mostly on closures. Further details are given in Section 3.4.1

In addition to the results we retrieved oral feedback explaining the implemented ideas. We use those comments to present the following summary. Figure 3.6 gives an overview of the results. Node movements in (a) are shown by connecting the node position to its new position, given as shadowed rectangle, with a curved directed line. The creator of (b) goes further and includes notable node movements (green arrows), newly created edges (blue arrows), vanishing edges (red strike out lines), and global movements (big grey arrows) in the result. This visual encoding might be confusing, since arrows are used for movements (green) and edge changes (blue). On a simple graph, the combination of edge creation and movement is depicted as contraction patterns in (c). This method cannot easily be transferred to more complex graphs since it is limited to contraction and expansion. A different idea is given in (d), where traces ("spur") of close nodes are created and their transformation from the transition is observed. This visual annotation combines very precise node changes and global patterns, but additional traces might lead to false assumptions about connections. (e) and (f) are similar. In both, nodes and their new positions are surrounded by outlines. Additionally, (f) indicates new connections between clusters with small black lines. In (g) closures are used to annotate movement of sub-graphs (indicated with arrows) and creation of new edges (with + as label).

The used metaphors can be categorized into two groups: in (a) – (d) line variations highlight changes, while (e) – (g) uses closures. In all examples node movement is indicated using different granularity ranging from a very detailed indication in (a) to high level patterns in (g), or a combination of both, e.g. (b) and (d). This granularity of changes are revisited in Section 3.4.2 when discussing comics. In the prototype the user can choose the subjectively best granularity.

Recurring graphical primitives in the examples are arrows which are used to depict node position changes and the use of closures to summarize changes in sub-graphs. Those primitives are used in the prototype. The feedback of the study was inspiring and offered many design variants. However, the large variance also implies, that the small sample set cannot define a common trend or correlations between user groups.

3.4.2 Comic book metaphor

Comic books served as another source of inspiration. In his book, Scott McCloud [24] shows how comics can dilate time and explain complex changes. Time can be stretched in a single frame showing multiple actions that happen sequentially in the same panel. One the other hand one complex action can be split into multiple panels. This leads each of those panels showing one part of the action, which are easier to understand. The overall action can then be inferred from all panels.

In order for a reader to understand the connection of the single actions happening in panels they have to be arranged intuitively. Although, reading directions are culture dependent (Japanese comics are read from top-right to bottom-left) all comics connect frame sequences via proximity and horizontal line-first. For our needs, panels arranged in a single line from left to right are sufficient. Multiple lines can be used to present an overview over all panels, though.

Following the ideas behind comics, our prototype allows users to divide a complex transition of a dynamic graph into smaller, easier to understand sub-transitions. The results, depicted as comic strip, can be used to gain insights in the changes happening in the transition and to show those findings to others.

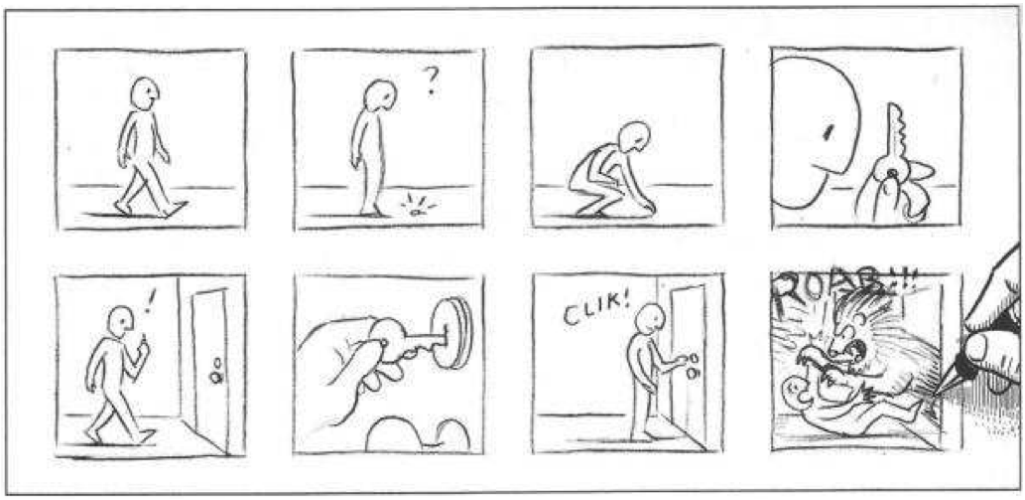


Figure 3.7: Splitting a process into smaller steps. Every panel shows the minimal amount of change necessary to understand the story. Note that removing a panel leaves the story incomplete. (© Scott McCloud – Making Comics: Storytelling Secrets of Comics, Manga and Graphic Novels [24])

3.4.3 Panel design

In the prototype, the user can use both annotations and the comic metaphor to explore a dynamic graph. While annotations simply can be turned on and off, the comic metaphor is always present. The prototype shows one transition from a state to the next state of the dynamic graph. All stages are shown as comic like panels on the canvas, the first and the last panel being the beginning and final state of the transition. The user can interact with all panels in between, starting with the complete transition from start to end. As explained later in more detail (see Section 3.5.1) a transition can be split into one or more sub-transitions allowing for a finer analysis of the primitives of the transition. Due to this, the graph is shown in an inconsistent stage in some panels. In order to emphasize this and give a feeling for which parts of the node-link diagram can be trusted, the size and translucency of vertices are changed depending on whether they are at their final position or not. Nodes that are at their final position or about to be at the end of the current transition, are shown bigger and completely opaque whereas nodes staying at their initial position are small and semi-transparent. The same is true for edges.

For sub-transitions, the set of changing, called active, nodes can be highlighted. This is done by enclosing the nodes in iso-surfaces (see Section 4.3 for a more detailed explanation). Those closures are also used in the next sub-transition to show the final position of their original nodes, making all panels readable without animation. Therefore, color is used to identify the same closure in the next frame. The color is cycled for each panel so that the actual closure of this panel can always be distinguished. In order to show the current active color, the bottom part of the panel is dyed as well. For panels only showing a certain set of changes (e. g. only creating edges) the top part of the panel is dyed in the color representing the changes (e. g. blue for creating edges – see Figure 5.2).

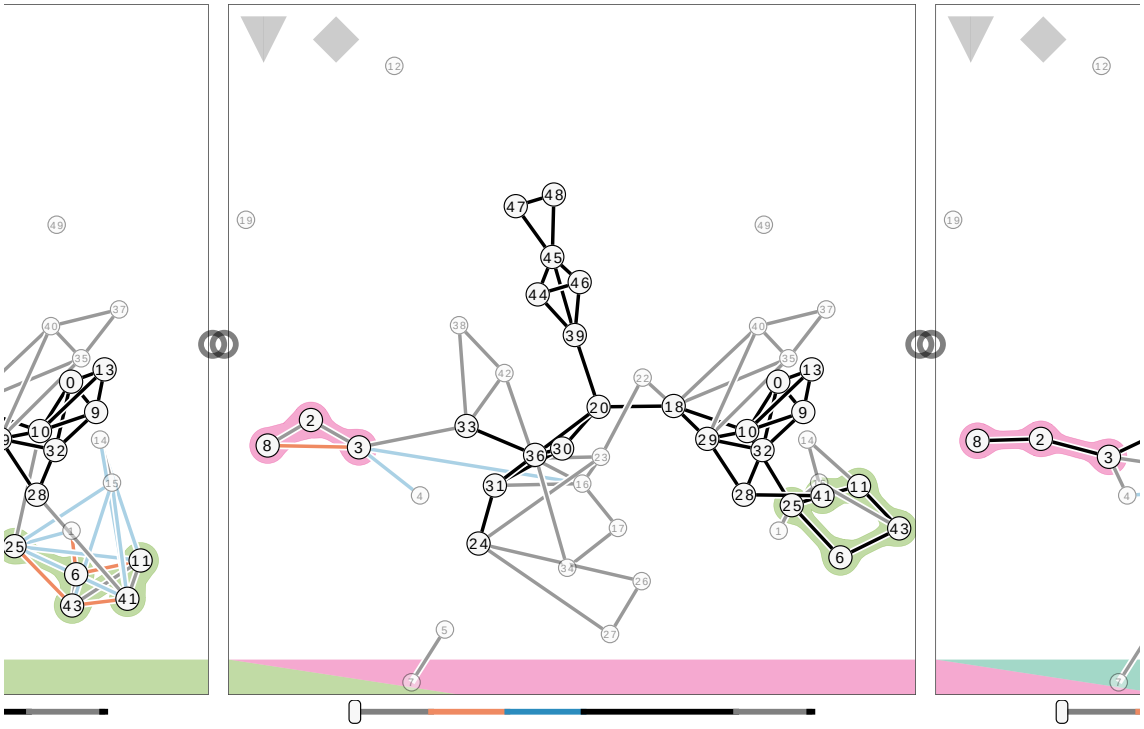


Figure 3.8: The design of transition panels. The symbols at the top are used for panel operations. The triangle splits the transition into its phases (see Figure 3.10), the diamond shape symbol is used to move steps around (see Figure 3.12)). The circle symbol seen in Figure 3.11 is not visible because the panel was created by an automatic split. The chain symbol between panels is used to merge panels as described in Section 3.5.3. The pink closure of the nodes show the set of active nodes matching the bottom color. The green closure is the set of active nodes from the previous panel showing their final position. The slider at the bottom is used to interact with animation.

3.5 Interaction

When exploring a dynamic graph interaction is an important part of an application. Interaction gives the user the possibility to focus on details or get an overview. Continuing with the comic metaphor interaction is used to build a personalized comic showing the changes of the graph for either understanding them or presenting them to others. Comic panels are created by splitting a transition into stages representing partial transitions. Those stages can either be defined manually by the user (see Section 3.5.1) or be determined by using heuristics (see Section 3.5.2). Splitting a transition into its phases, edge removal, edge creation, and node movement is shown in Section 3.5.2.

To personalize the resulting graph comic even more and undo mistakes done by the user, panels can also be merged again (see Section 3.5.3). For convenience the panels can also be reordered (see Section 3.5.4).

The prototype uses a zoom-able user interface to ease navigation through the shown small multiples of the given dynamic graph (see Section 3.5.5). The first and the last of those panels show the actual states of the graph at the beginning and at the end. The border of those panels are slightly thicker than the other borders in order to make them distinguishable (see Figure 5.1 and Figure 5.2). Panels in between show partial transitions of the overall step. The panels can be arranged as a long strip which makes it easier to interact with them or as a comic-like grid which makes for a better overview of the panels. Although unlike in a comic the last panel of a row in the grid gets copied to the beginning of the next row to allow pairwise comparison of panels consistently (see Figure 5.3).

Focusing on a single node over all panels can be achieved by clicking on the node. This highlights the node in all panels. Multiple nodes can be highlighted at the same time.

Animation provides additional interaction with the graph comic (see Section 3.5.6). Having a higher precision than a small multiples representation the time it takes to watch a transition lowers the amount of information that can be processed. Therefore, it is vital to allow a user to always stop an animation and change the progress of the transition manually. This is achieved by using a slider as shown in Section 3.5.6. Furthermore, the timing of the animation has to be chosen carefully as seen in Section 3.5.6. Slowing down the animation at the beginning and the end of the transition helps anticipating the movement of nodes.

3.5.1 Manually splitting stages

A powerful interaction technique with dynamic graphs is to divide a complex transition into multiple steps. While this splitting may lead to temporary inaccuracies, it helps to gain insights by separating the changes into manageable chunks. In the application this can be done by splitting a transition panel into two or more new panels. Splits can be user guided or automatically created.

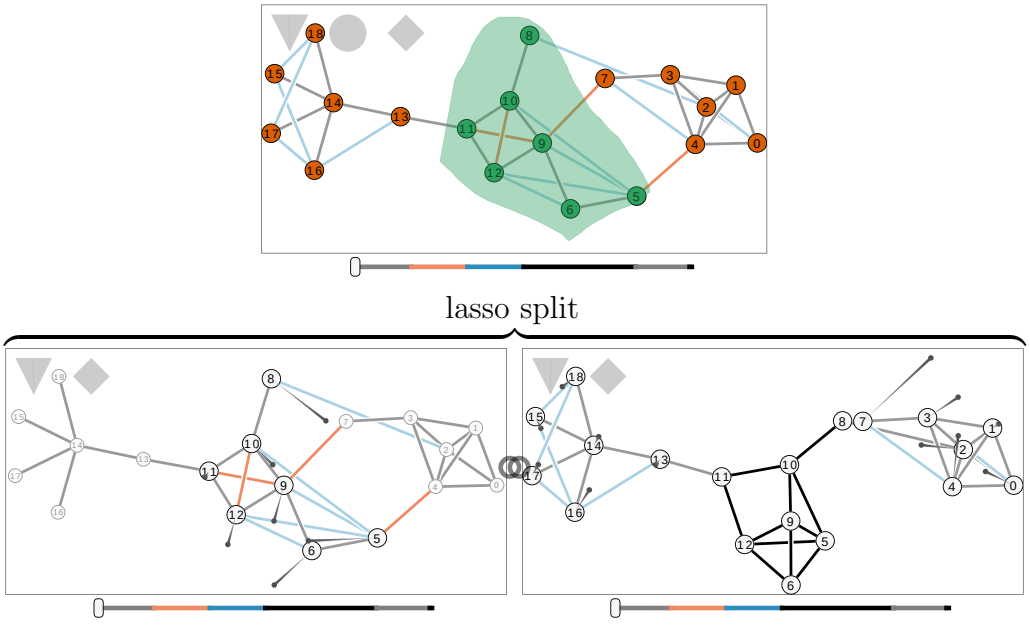


Figure 3.9: Splitting a transition with a lasso selection.

Selecting subsets

Splits can be performed on stages of a transition (see Section 3.5.2) or based on node subsets. We define every node that changes during a transition as active. When doing a subset based split, we create a transition panel with all nodes remaining in the beginning state of the original transition except those defined by the subset. The nodes defined by the subset become active in this panel. In the second panel the nodes from the subset remain in the end state of the original transition and the other nodes become active. A node can become active only in one panel. After that nodes remain in their end state.

The state of a node in this context means its position in the node-link diagram and the edge configuration. When a node becomes active its edges transition as well. This means that nodes that have not been active yet may have edges that are not consistent with the beginning state of the graph.

For user guided node subset splits the user can use lasso selection to define subsets. The split is then performed on this subset. Only nodes that are active in the given transition panel are considered by the selection. Subsets containing less than two nodes or more than $n - 2$ nodes do not lead to a split.

3.5.2 Automatically splitting stages

In order to speed up the progress of creating an insightful series of transition panels to explore a dynamic graph automatic splits can be performed. Those work with a heuristic to create a number of transition panels.

Splitting into phases

A useful split for static images is to split a transition panel into the different stages of a graph transition: edge removal, edge creation, and node movement (see Section 3.3). The split creates three new panels that show edge removal in the first, edge creation in the second, and the node movement in the last panel. This kind of split is useful when the panels are meant to be printed and therefore animation is not available.

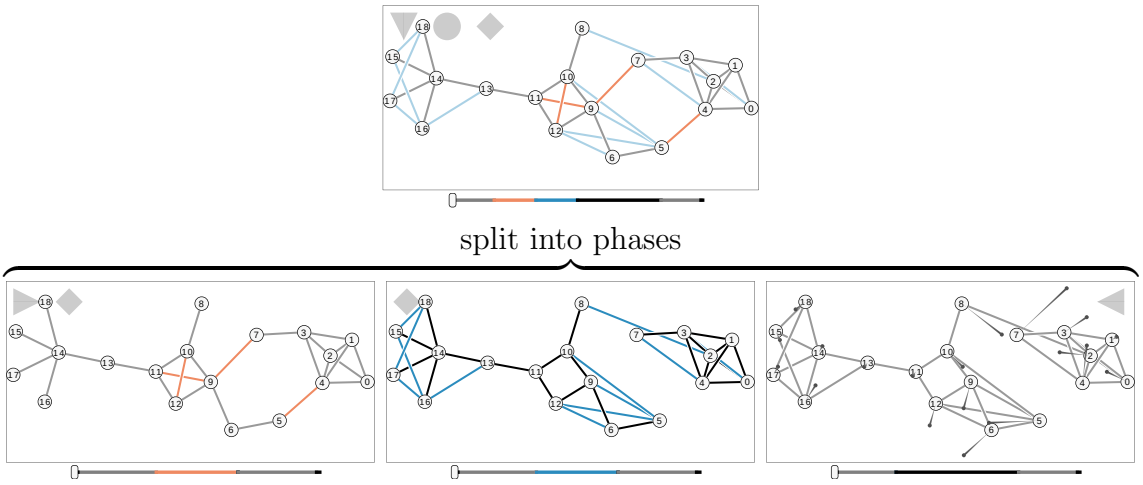


Figure 3.10: Splitting a transition into its phases by clicking on the triangle symbol. Clicking on either sideways triangle reverses the action.

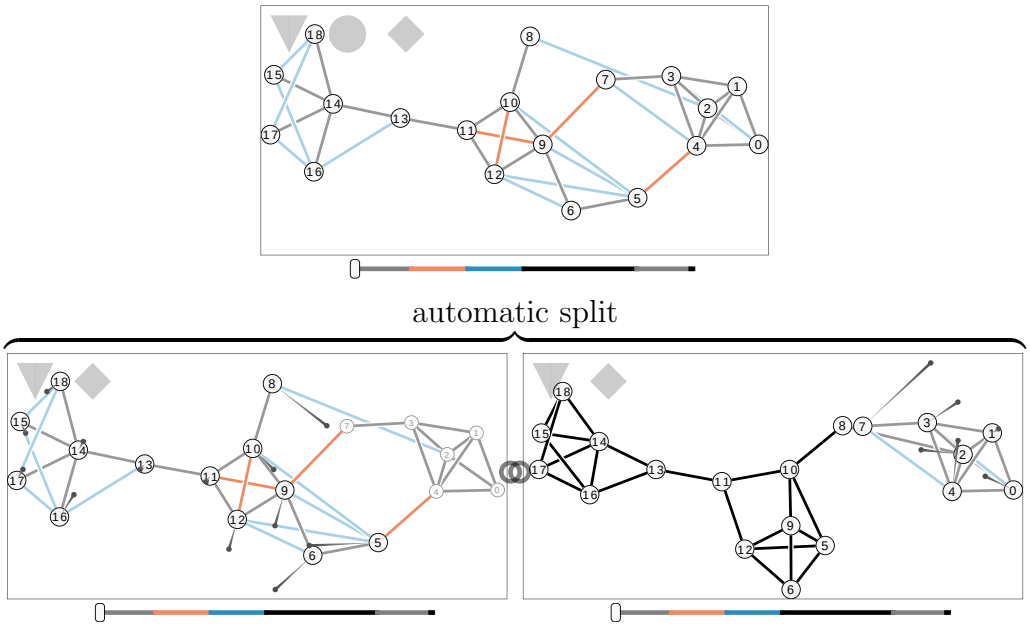


Figure 3.11: Clicking on the circle automatically splits the transition into the connected components of the conjunction graph $G = (V, E_0 \cap E_1)$. Note that this graph unfortunately has only two of those islands because the left two clusters are always connected. The split can create an arbitrary number of panels.

Splitting via Clusters

Another automatic split is to use subsets of nodes defined by graph clusters. However, defining good clusters used for splitting is not easy. Clusters from the beginning state of the graph or from the end state can be taken. This does not yields good results if for example clusters change drastically between the step. Gaertler *et al.* [15] for example provides a cluster algorithm for dynamic graphs.

A much easier heuristic for clustering dynamic graphs is to detect islands of nodes that are connected by paths taking only edges that are present in both time steps i. e. all connected components of the graph $G = (V, E_0 \cap E_1)$. Using those islands as subsets for splits gives a first quick overview of the changes of the graph. However, this heuristic results in too large clusters e. g. when there are persistent edges connecting two clusters that do not merge.

3.5.3 Merging stages

Complementing the creation of new panels via splitting the user has the ability to remove panels. The removal is done by merging two panels. Which panels to merge can be indicated by clicking on a symbol between both panels. For panels that show the different phases it would be tedious to manually have to merge them one by one, so all three panels get merged at the same time. Merging any number of panels is implemented by taking the beginning state of the first panel to be removed and the end state of the last and creating a new panel with those states.

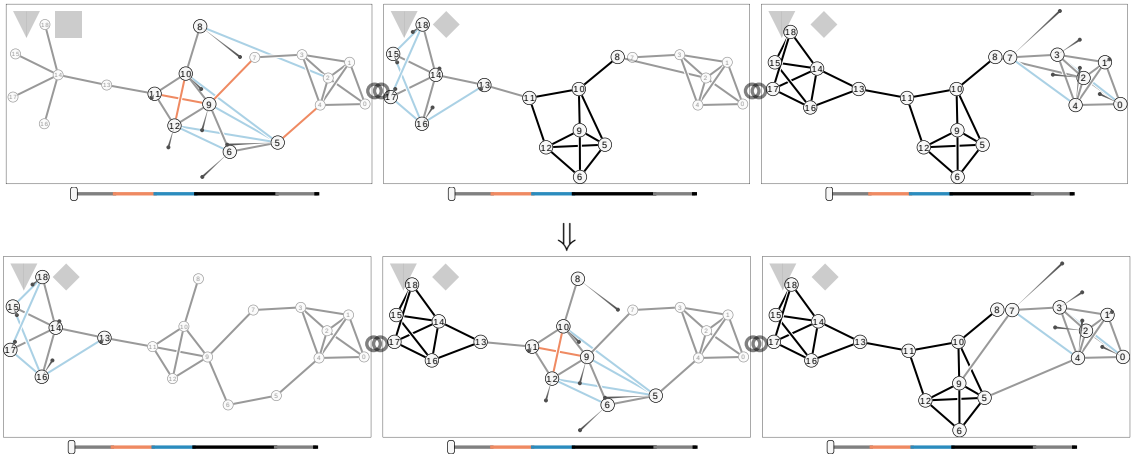


Figure 3.12: The first panel gets moved to left of the last panel. Clicking on the diamond turns it into a square. Then clicking on the diamond of a different panel moves the panel with the square left to this panel.

3.5.4 Reordering stages

In order to gain more insights in the dynamic graph rearranging the transformation panels can be useful. For every transition panel the non active nodes are either at their start state or at their end state. When moving panels around the distribution of nodes in their start and end state is shifted. This allows a user to see a transition in a different context and may lead to new insights about the dynamic graph.

Moving a transition panel around also changes the context of the surrounding panels. The problem of nodes in their beginning state showing edges from their end state can be mitigated by moving panels around so that the edges of the nodes in the panel that haven't been active yet show the correct edges for their state.

Panels that show a single separate transition phase can only be moved with their corresponding other phases. Otherwise it would be possible to create transition panels that create a complete inconsistent view on the dynamic graph, e. g. when only creating edges for nodes that are in their start state.

3.5.5 Zoom-able user interface

The prototype uses a zoom-able user interface to provide the ability to focus on details or get an overview over all panels. Opposed to the zoom-able user interface of Perlin and Fox [25] the zoom does not change the displayed content, though. Navigating the zoom-able user interface can be done in various ways. In addition to the standard panning and zooming, the user can double-click on a panel to zoom it to full size so that only this panel is currently visible. Arrow keys can also be used to navigate between panels when they are arranged as strip. When using arrow keys the ZUI behaves as if the user double-clicked on the panel left or right of the panel that is currently hovered by the mouse.

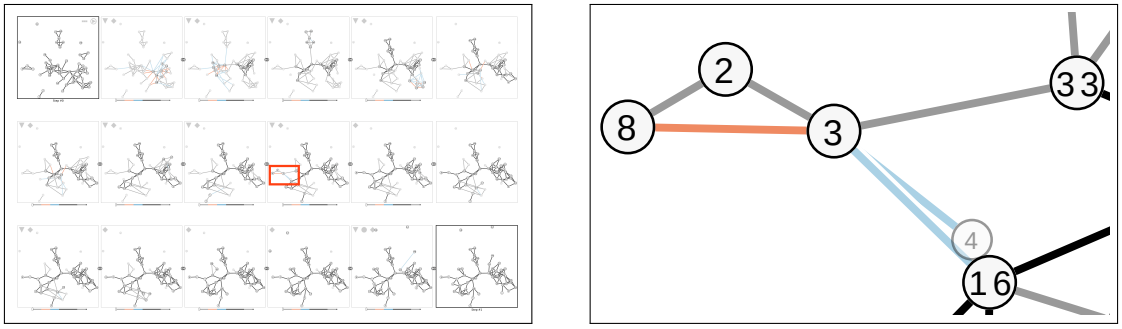


Figure 3.13: The same scene with different zoom levels. The content of the right image is the same as the area enclosed by the red rectangle in the left image.

3.5.6 Animation

In addition to a small multiples representation of different transition stages of one time step of a dynamic graph an animation of those transitions can be used to show the changes. The use of animation, however, is controversial as discussed by Tversky *et al.* [32], and must be handled with care. Tversky *et al.* mention two principles that must be obeyed. The *Principle of Congruence* states that the animated object should have a natural connection to the underlying conceptual change. This is the case when animating dynamic graphs whose node positions correspond to the topology of the displayed graph. The *Principle of Apprehension* is described as follows: "... animations must be slow and clear enough for observers to perceive movements, changes, and their timing, and to understand the changes in relations between the parts and the sequence of events." Additionally they encourage the use of "... annotation, using arrows or highlighting or other devices to direct attention to the critical changes and relations." This quote gives support to the approach taken in the prototype.

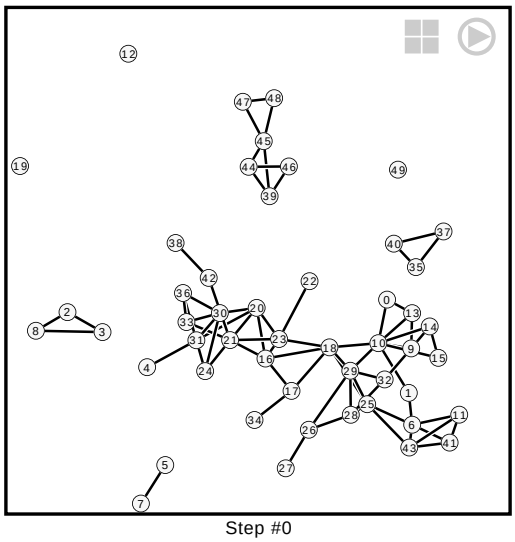


Figure 3.14: The design of the panel containing the initial state of the graph. The buttons in the top right corner can be pressed. The button consisting of four squares rearranges the panels in a multi-line comic mode (see Figure 5.3). The encircled triangle starts a full animation of all consecutive stages. All transitions are performed in one panel and are executed in order.

Interaction with animation

In order for animation to be effective it is vital that the user always has full control over it [2]. This can be achieved by enabling the user to always stop or restart the animation and showing a slider of the current progress of the animation. The slider needs to be able to directly set the progress of the animation [13].

In the transition panels the slider for the animation is always shown at the bottom. Since the transition is split in multiple phases, edge removal, edge creation, and node movement, the color of the slider shows in which phases the progress is currently in. When the slider is set to the very beginning of the animation all three phases are shown at once. This view can be used for a small multiples only representation. As soon as the slider is moved to another position or the animation is started the phases are split up and shown separately. The edge phases let the new or old edges fade in or out. The node movement phases interpolates between the start and end position of the nodes. The colors on the slider are chosen to be orange and blue for the edge phases as the edge colors in those phases. The color of the movement phase is simply grey.

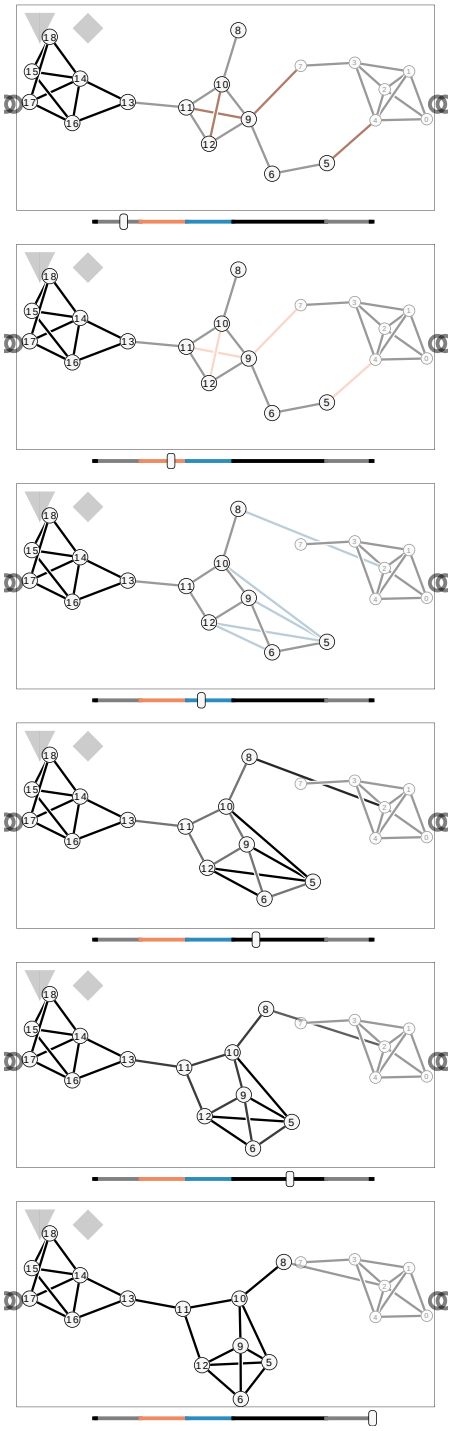


Figure 3.15: Frames of an animated transition.

Moving the slider manually makes it easy for a user to recognize patterns of cluster movements, cluster contractions, or cluster expansions.

In order to use animation as complete substitute for small multiples the user can start a full animation going sequentially through all transitions by clicking on a button in the panel of the initial state of the graph (see Figure 3.14).

Timing and smoothing

The timing of an animation has to be handled with care. It should not be too fast, otherwise it is difficult to follow. On the other hand, a slow animation can take too much time to watch or lose the focus of the user. For the prototype two seconds are a good compromise. Furthermore, smoothing the animation makes it easier to anticipate the movement. Smoothing is achieved by slowing down the animation at the beginning and the end, and having the highest speed at the middle. A quadratic transformation similar to the one in Dragicevic *et al.* [12] is used in the prototype.

4. Implementation

The following chapter highlights some challenges that arose during the implementation of the prototype. Using a straight forward way to implement features often leads to performance bottlenecks.

Firstly, we will look at the implementation of the zoom-able user interface (see Section 4.1). In Section 4.1.1 non trivial interactions with the zoom-able user interface are explained. This includes zooming towards a given point on the screen and zooming the scene so that only a specific rectangle is visible. After that we discuss performance boosters for a zoom-able user interface. Namely, avoiding the computation for rendering non visible items in Section 4.1.2, and using caching techniques to speed up the rendering of all elements in Section 4.1.3. The presented implementation techniques allow for scalability of the prototype considering that many comic panels, showing the same graph in terms of nodes and edges, can be created by the user.

Section 4.2 focuses on animating a larger number of items without performance losses. By choosing a format of representing animation that recomputes the current position from the start and end points incorrect positions are avoided (see Section 4.2.1). Since animated items do not influence each other the computations for animation can be parallelized as shown in Section 4.2.2. Locking (see Section 4.2.3) is used to prevent situations where some rendered items appear at the wrong position when animation computations are taking place during a redraw. This is especially inconvenient when reusing positions for drawing nodes and edges connecting to those nodes. The positions might change and edges appear at a different position than their corresponding nodes.

In Section 4.3 we will look at the implementation of two closure shape generation algorithms. Bubble-sets are a standard technique for generating iso-surfaces (see Section 4.3.1). However, the border is not smooth without significantly increasing the vertex count and the created sets are not coherent when moving their elements around. Kelp-like diagrams provide a cleaner looking alternative (see Section 4.3.2). By imitating the visual style of Kelp diagrams and removing some constraints, those iso-surfaces are easy to implement while having a clean look. Kelp-like diagrams are used as standard set closure shape in the prototype.

4.1 Zoom-able user interface

The main part of the prototype is the zoom-able user interface. The zoom-able user interface displays a portion of the canvas, which is an theoretically infinite area with infinite zoom bounded by the precision of floating point numbers. Visible elements are divided into render items. Render items have a bounding box which defines the area they can occupy and restricts graphic operations to that area, and can be organized hierarchically forming a nested two dimensional implicit tree, called scene graph. The bounding box of a render item can change for every frame allowing for example ordered groups of render items to add or remove render item children. In order to make handling of render items easier the offset of the bounding box is with respect to their parent item and the scaling of the graphic context does not change when traversing the scene graph.

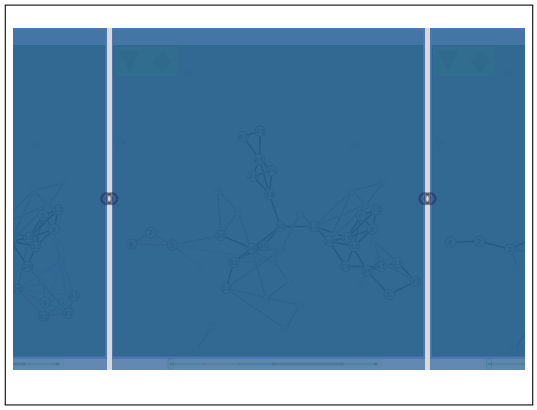


Figure 4.1: Bounding boxes of the render items rendered as semi transparent rectangles in the current scene. Due to the number of container render items used to position symbols, the node-link diagram, and the slider, the bounding boxes of a panel overlay and appear opaque.

4.1.1 Zooming and panning

At any time only a small portion of the canvas is visible. Zoom-able user interfaces provide two basic operations to change the visible rectangle of the canvas: zooming and panning. The transformation from screen coordinates (Matrix M) to canvas coordinates (Matrix C) can be described as

$$C = S \cdot T \cdot M$$

where S is the scale of the canvas and T its offset in the form of

$$S = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad T = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

using homogeneous coordinates. With this representation panning is easily done by adding the movement of the mouse in pixels to T to get an intuitive pan operation. A zoom operation is a little bit trickier since just changing S always changes the visible canvas with respect to the top left corner of the screen (the origin of the screen coordinate system). An intuitive zoom operation can be achieved by adjusting

the translation of the canvas such that the point under the mouse remains at the same position. Given p_x and p_y as mouse screen coordinates and f as zoom factor, we shift the offset to the mouse position, apply f, and shift back. The resulting transformation matrices are:

$$S = \begin{pmatrix} sf & 0 & 0 \\ 0 & sf & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad T = \begin{pmatrix} 1 & 0 & (t_x - p_x)f + p_x \\ 0 & 1 & (t_y - p_y)f + p_y \\ 0 & 0 & 1 \end{pmatrix}$$

Another operation that is not required but very useful, is to be able to zoom in on a given rectangle. This means that after the operation is performed the given rectangle is fully visible while being shown as large as possible. The first step is to figure out the scaling. This can be easily done with the factor $f = \min\{w_s/w_r,\ h_s/h_r\}$ where w and h means width and height respectively and s is the visible rectangle in screen coordinates and r the given rectangle in screen coordinates. When taking the maximum of the values as factor, the screen is zoomed so that the rectangle fills the whole view but is not necessarily shown completely. Now, setting S to the identity and T to

$$T = \begin{pmatrix} 1 & 0 & (w_s - w_r)/2 - x_r \\ 0 & 1 & (h_s - h_r)/2 - y_r \\ 0 & 0 & 1 \end{pmatrix}$$

where x_r and y_r is the top-left corner of r, and the zooming with the factor f towards the center of the rectangle r yields the correct result. Putting everything in the zoom formula gives us the following new translation and scale. Note that the zoom position needs to be in screen coordinates which happens to be in the same scale as the current canvas coordinates:

$$\begin{aligned} p_x &= x_r - t_x = (w_s - w_r)/2 \\ p_y &= y_r - t_y = (h_s - h_r)/2 \\ t'_x &= (t_x - p_x)f + p_x = -x_r f + (w_s - w_r)/2 \\ t'_y &= (t_y - p_y)f + p_y = -y_r f + (h_s - h_r)/2 \end{aligned}$$

$$S' = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad T' = \begin{pmatrix} 1 & 0 & \frac{w_s - w_r}{2} - f x_r \\ 0 & 1 & \frac{h_s - h_r}{2} - f y_r \\ 0 & 0 & 1 \end{pmatrix}$$

4.1.2 Pruning the scene graph

A heavy decrease in performance of drawing the nested set of render items called scene graph occurs when a fairly zoomed in portion of the canvas is visible. This is due to the fact that visible items take longer to draw because of their increased size, in combination with the fact that even though some render items are not visible they are still drawn. Even when the system provided way of drawing primitives detects primitives that are fully outside of the clipping rectangle and returns immediately, the calculations leading to the operation of drawing the primitive are still performed. This is not necessary. Consider, for example, a non fully visible node-link representation. The color of the nodes that are outside of the visible rectangle are still computed, which may be an expensive operation when the color depends on the current state of the node-link diagram and is dynamically generated.

In order to overcome the problem of calculating properties of render items that are not visible, a transformed representation of the visible rectangle is handed through the drawing routine traversing the scene graph. Since every render item has a bounding box it can be checked for intersection against the transformed visible rectangle before attempting to calculate properties for drawing. Since render items nest and a child bounding box is always fully enclosed by its parent's bounding box the complete sub-graph of a render item outside of the view can be skipped.

Calculating the bounding boxes creates a large number of rectangles (at least one rectangle per render item) every frame. Because of the Java memory model those rectangles cannot be created solely on the stack and have to be allocated on the heap. This produces a lot of garbage which leads to more frequent garbage collections. The garbage collection pauses can cause noticeable freezes during animation or user interaction and should be avoided. Even though this problem cannot be solved completely due to the nature of the Java memory model, it can be mitigated by producing less objects during drawing. This in turn can be achieved by reusing existing objects during the drawing process. This programming style, however, is error-prone and should be avoided in non performance critical areas.

4.1.3 Using caching for smaller zoom levels

Pruning the nested set of render items called scene graph does not help to improve performance when all render items are fully visible. Since every render item is shown smaller it takes less time to actually draw the objects but the time to compute the content to draw usually takes more time than the drawing. However, when zoomed out most objects are static because they are too small to interact with. We can exploit this fact by rendering items once into an image that we keep in memory, given that the items are zoomed out far enough. For every successive paint operation we can use this image to render the item fast. This speeds up the rendering process significantly when showing many items. When a render item does change, e. g. due to an animation, it can set a flag so that the cached image is removed and not used until the animation is over. With many cached render items we quickly run into a memory problem. Therefore we have to use low resolution images and return to actually drawing the render item when we are closer than the image allows us to be.

4.2 How to animate many items

In this section I describe how to boost performance when implementing animation. Animation can be used to have smooth transitions between two stages of the graph. Animated transition needs to be enabled for visible objects in the scene, e. g. nodes with their edges, and the camera showing the canvas. Animation of the camera can be used to e. g. make the transition of zooming in on a panel smooth. Additional to that it may be useful to schedule certain actions or events for when an animation has been completed. This allows for example for animating the complete sequence of panels without making the implementation complicated. Every panel needs only to know one transition. Showing the complete sequence can be done by positioning all panels on top of each other and making only one of them visible. After the current visible panel has completed its transition it is turned invisible and the next panel is shown with its transition.

4.2.1 Deciding on a format

Every transition of an object is defined by a start configuration σ_0, an end configuration σ_t, and a time span t. The easiest way to represent animated transitions for objects is then to compute the number of frames of the time span, and how much the current configuration has to change every frame. However, this method has some issues. First of all, it is not guaranteed to exactly reach the end point of the transition after the given number of frames. This can be due to numerical instabilities when adding relatively small values to the current state. Also, the time span may not be precisely converted into an integer number of frames if the time span is not a multiple of frame times which leads to a slightly different timing than intended. Moreover, this kind of transition is heavily dependent on frame times. If a frame takes longer to compute or drawn the animation slows down. Which makes it more difficult to follow elements on the screen, because the eyes have to constantly adjust to the current speed of the objects. Attempting to have non-linear transitions with this method is even more difficult.

Another approach on representing transitions is to remember the start configuration σ_{start}, the start time t_{start}, the end configuration σ_{end}, and the end time t_{end}. t_{end} can be computed by adding the time span to the start time $t_{end} = t_{start} + t$. With this information the current configuration can be computed any time with $\sigma_{cur} = (1 - p) \cdot \sigma_{start} + p \cdot \sigma_{end}$ and the progress $p = \frac{t_{cur} - t_{start}}{t_{end} - t_{start}}$. Note that the progress p is a value between 0 and 1 as long as t_{cur} is between t_{start} and t_{end}. If $t_{cur} \geq t_{end}$ we can set $\sigma_{cur} = \sigma_{end}$. This guarantees us that we always end up in the correct position. Also, when the time span is not a multiple of frame times the timing of the transition is still correct. When frames take longer to compute or draw the animation is not affected. Even when lag occurs, after the next redraw the object is at the position the eye anticipated because of the previous movement of the object.

The animation can be controlled by mapping the progress p to different values between 0 and 1 which can be used to influence the speed of the transition during animation. Which mappings to use in order to get an appeasing transition is discussed in Section 3.5.6.

A drawback of this approach is that the state of an object has to be computed every time it is accessed. This leads to two problems. When computing the configuration for objects inconsistencies can occur because the time changes during computation. This results in out of sync positions for objects or even worse wrong positions of one object that is accessed multiple times. For example, when the edges of a graph are drawn the position of the nodes have to be accessed again, which gives slightly different locations of the nodes with respect to the edges. This problem can be solved by handing in the initial time of the call to the draw method. However, this leads to the second problem that the current time influences the state of objects in all code. So keeping a consistent time over all areas of the code base requires a global state that is accessible from all objects that have animated transitions. This makes handling those objects more difficult.

A far more simpler solution to this problem is to have the current state σ_{cur} of an object stored in the object. The state is then updated by an animator thread whose job it is to compute the current states of all objects that can have animated transitions. (Note that since the actual value for t_{cur} is controlled by the animator

thread, time can be stopped or advanced in smaller steps e. g. for creating a series of screen-shots which otherwise would potentially miss some frames of a transition) The animator is run at least once per frame. This also gives the opportunity to have events that are triggered by the end of an animation, because the time when the event happens is defined. Without a dedicated animator thread, a triggered *animation end* event would be executed whenever the first access to the object, after the animation ended, is made which can be anywhere in the code base (This would lead to undefined behavior in the case of an exception and the need for locking so that the triggered event is executed only once).

Not exposing the current time t_{cur} to users of the objects, however, makes it difficult to start a transition, since the current time is needed to compute the correct times. Again, directly using the current time could lead to inconsistent animations when multiple objects are affected. The solution to this problem is to memorize which actions have been done to an object and execute them, in the correct order, when the animator thread computes the new state of the object. When doing actions on an object the impact has to be visible immediately sometimes, so actions have to directly manipulate the configuration of an object and have to be memorized in order to be replayed in the correct order by the animation thread. This is necessary because an *animation end* event may be triggered when the state was changed.

Having an animator thread with defined times for *animation end* events can be used to implement scheduled actions. By creating an animated object that has no actual changing state an animation can be started with the desired time schedule. When the "animation" ends an *animation end* event is triggered and the scheduled action is executed.

4.2.2 Fork join pools

Another advantage of having an animator thread to compute object states is that the actual computations can be parallelized. This is possible since the actual computation of σ_{cur} depends only on the object and the current time. *Animation end* events have to be collected and executed after the parallel phase, though, since they usually have an impact on the surrounding state.

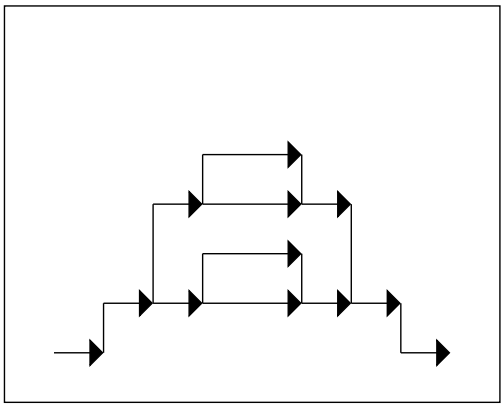


Figure 4.2: The flow diagram of a fork join pool. Horizontal lines are tasks. Tasks that are above each other can be computed in parallel. A task can be forked and its result can be awaited.

Operations on random access lists can be easily parallelized with the use of fork join pools. A fork join pool is a thread pool that allows tasks to fork, computing both paths in parallel, and join again when both results are available. This can be used to split a long task into multiple shorter tasks until a reasonable length is achieved and computing those tasks in parallel. However, since using a fork join pool has a slight overhead it is recommended to fall back entirely to sequential computation if the number of elements in the list is small enough.

In order to avoid a complex mechanism of removing objects from the animator list weak references can be used. Weak references are references to objects that are not counted during garbage collection. This means, a weak reference may be null after garbage collection when the referenced object is not referenced anywhere else. After the animation computation, weak references that are null can be removed from the list in order to compact it. This makes handling animated objects easier because when an animated object is not used anymore it does not have to be removed manually from the animation thread.

4.2.3 Locking

A drawback of animating and drawing concurrently is that, at the point of drawing, some objects might already be in the next state and some may still in the current state. This can lead to weird effects in node-link diagrams where the edges of a node differ from the actual position of the node. A possible solution to this problem would be to keep two states in one object. The active state being the one that is used for drawing and the inactive state being updated by the animator. However, this requires a global state to determine which object configuration to use and locking for the global state so that it is not switched during either animation or drawing. Another solution which requires less complexity and memory is to lock a semaphore when either animation or drawing is active. The additional waiting time for a drawing operation when waiting for the semaphore to be freed is negligible. For sufficiently fast animators the user interface still feels responsive. On the other hand, a complete split between animating and drawing enables quick redraws of the user interface and immediate reactions to interaction between two animation frames.

4.3 Iso-surfaces for sets

Using small multiples it can sometimes be difficult to find nodes that have been moved from one panel to the next. Coloring nodes to indicate their position between panels removes one visual feature from the repertoire. This feature could for example be used to mark a given node or show additional information about the dynamic graph e. g. groups in the graph. Given that only active nodes in a transition panel can move, another way of showing their position between frames is to enclose all active nodes of a panel in both the current and successive panel. Cycling through multiple colors for the closures and using the same color for the same group of nodes makes it easy to identify nodes that move between panels. At most two of those closures need to be shown in one panel. The current set of active nodes and the previous set of active nodes. This also enables to read a small multiples representation without having to actually search for the nodes that changed between frames.

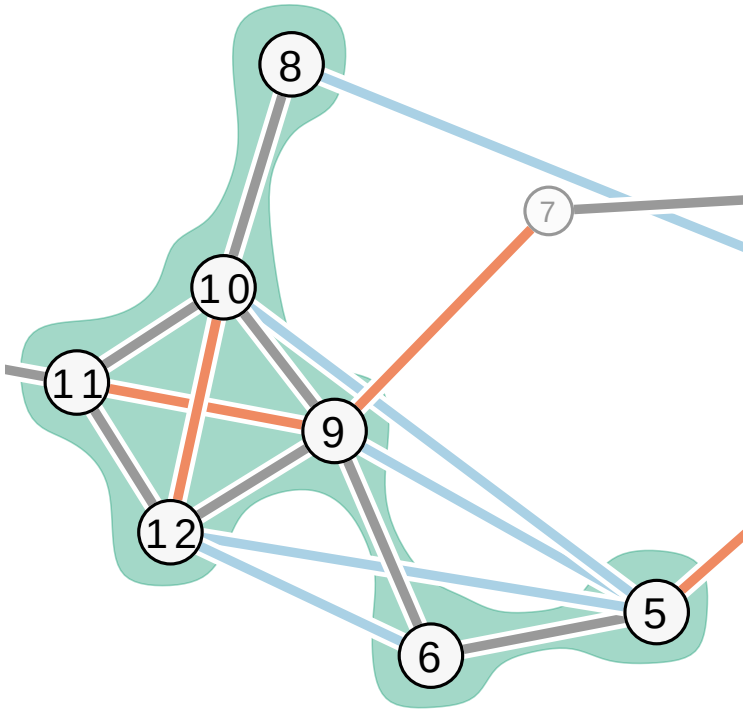


Figure 4.3: Set outlines created with the Bubble-sets algorithm.

There are a number of techniques to enclose a set of objects. The easiest method is to compute the convex hull of the objects. This, however, very likely also encloses nodes that are not in the set of active nodes. Iso-surfaces that try to minimize the area not covering any enclosed object are better suitable. Two iso-surface algorithms are presented here.

4.3.1 Bubble-sets

Well known iso-surfaces are Bubble-sets from Collins *et al.* [10]. Bubble-sets use potential fields and attempt edge rerouting to avoid overlaps of different sets. Since at most two Bubble-sets are shown at once in a panel avoiding overlaps between those iso-surfaces is not that important. However, by creating an invisible third set containing the rest of the nodes Bubble-sets can be used to avoid including nodes that are not active.

Bubble-sets are created by first building creating edges connecting the nodes that will be enclosed. In the case of graphs, those edges can be used, but it may be necessary to create additional edges in order to get a connected graph of all nodes in the set. After that a discrete potential field is created. The potential field is a grid containing values that is laid over the canvas with a given resolution. For every included node and the above mentioned edges, the value of the field is increased. For every node that is not included the value of the field is decreased. By using a smooth kernel over all objects the created sets are less tight. After creating the potential field it is converted into a dichotomous field by using a threshold. By walking around the edge of this field using the marching squares algorithm the actual Bubble-set shape is created. An open source Java implementation is provided by Krause [21].

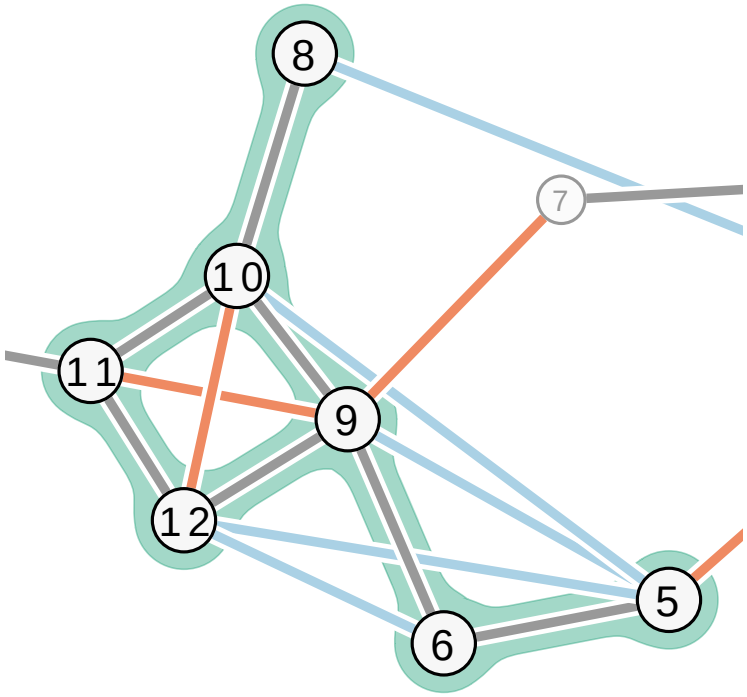


Figure 4.4: Set outlines created with the algorithm for Kelp-like diagrams.

4.3.2 Kelp-like diagrams

Bubble-sets compute more complex shapes than necessary for our use case, since we do not need extensive overlap avoidance. Also, they are strongly dependent on the resolution of the potential field which often yields results with jittery outlines. This effect increases when using them during animated node movement interpolation. Kelp diagrams by Dinkla *et al.* [11] (see Figure 4.6) have a less distracting look and behave more consistent. However, they still are too complex for our use case. Inspired by this the application uses Kelp-like diagrams.

Kelp-like diagrams firstly also need to create a completely linked set of edges like Bubble-sets. This is done by using the edges of every graph component induced by the set of nodes. Those components are then connected by creating edges between the closest nodes of two components. The algorithm was retrofitted into the Bubble-set implementation so that the created Bubble-sets can take advantage of the graph topology. After creating the edge set a thick version of the edges are connected to create the shape. At the nodes a circle shape is used and smoothed to connect to the edges. All those shapes combined define the Kelp-like shape (See Figure 4.5 for construction details). By the nature of those diagrams overlaps with non active nodes are negligible because only nodes that are contained in the set create a circle in the set. For the hypothetical case that a node is contained in two sets at once (which cannot be the case in this application) the radii of both circles are changed such that both shapes are distinguishable.

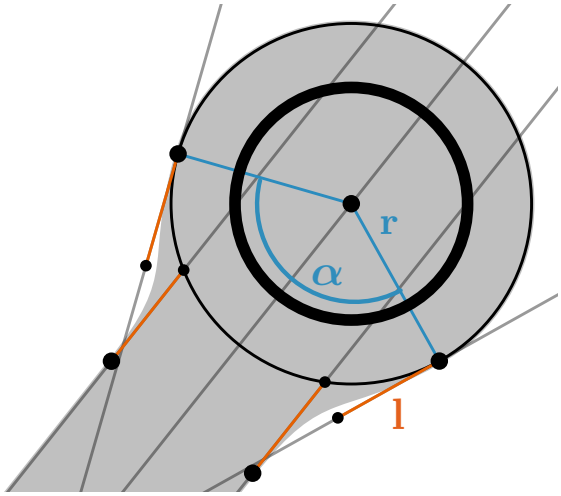


Figure 4.5: Kelp-like closure creation. The big dots on the edge and on the circle are the end points of the bezier curves used to generate the outline. Small dots are additional control points. Parameters are the angle α and the lengths l and r. The grey area is the final shape and the thick circle represents the actual node.

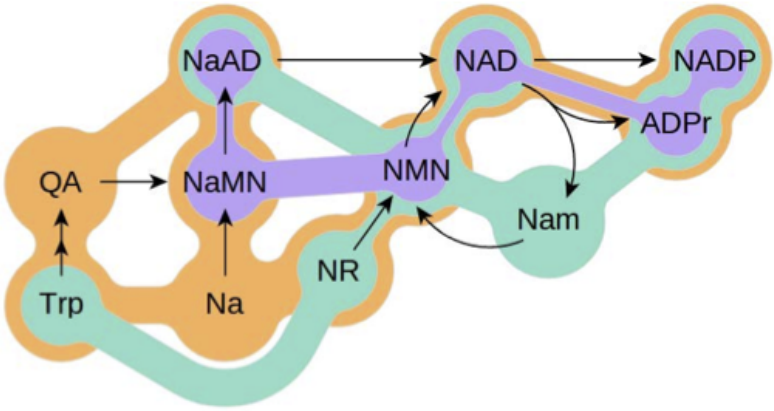


Figure 4.6: The original Kelp diagrams. Note the additional complexity of possible created shapes which is not necessary for our use case.

5. Case studies

In the following chapter the prototype is used to analyze three example graphs.

The first example is a synthetically generated dynamic graph with 19 nodes. It has three clusters which change their densities and relations to each other. The layout of the nodes uses techniques described by Brandes and Mader [6].

The second graph depicts face to face contacts of conference attendees at the ACM Hypertext 2009 conference. The complete data set is provided by SocioPatterns [31]. Contacts were detected via close range radio badges worn by the participants. In our discussion we will focus on contacts lasting at least 5 minutes that happened before and after lunch on the first day. The graph consists of 34 nodes being present at both time steps. The layout was generated similar to the first example.

The third graph is created with the data from the *Teenage Friends and Lifestyle Study* of the SIENA project [30]. The network contains snapshots of friendships between 50 girls ranging from 1995 – 1997. We will discuss the changes from 1995 to 1996. The used layout focuses on stability.

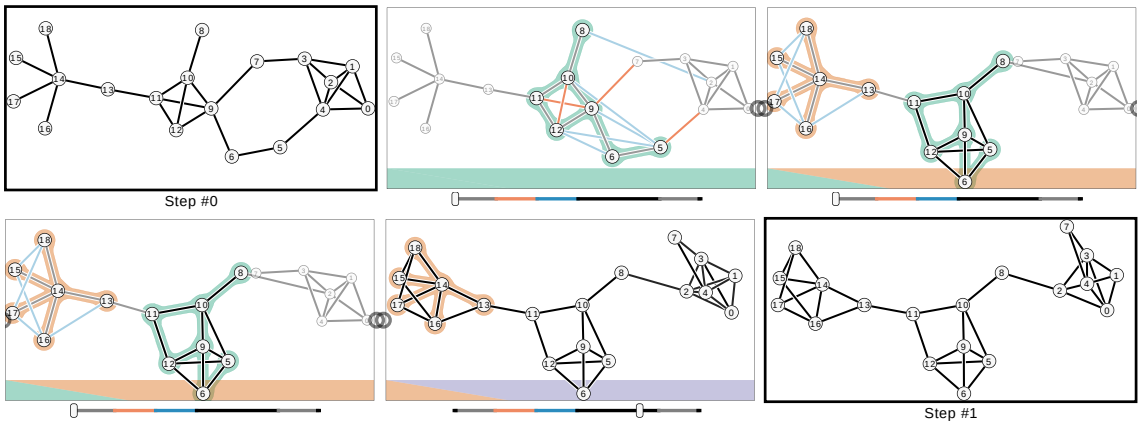


Figure 5.1: The final graph comic of the analysis of the example graph in Section 5.1. The transitions of the three major groups are split into sub-transitions. In the first stage the center group breaks up the former connections to the right group while gaining an alternative route. The star-like group on the left becomes a cluster in the second stage. The animation slider of the third stage is moved halfway through the node position interpolation. Since all edge changes of the right group happened in the first stage, the last sub-transition only consists of node movements.

5.1 Small example graph

In the small example graph there are three major groups of nodes. The nodes 0, 1, 2, 3, and 4 and 8, 9, 10, 11, and 12 form clusters that are almost fully connected (i. e. cliques), while the nodes 14, 15, 16, 17, and 18 are only connected through node 14 in the initial configuration of the graph. Naturally, we split the transition to the final configuration of the graph into three stages, to analyze how those groups change. When we look at the overall transition we notice that nodes 6 and 5 get connected to the central cluster so we include them in our split. In the first panel, we can then clearly see that the two bridges (3–7–9 and 4–5–6–9) get disconnected and remove the connection of the center cluster to the right cluster. This could be explained by 5 and 6 getting closer friends with 9, 10, and 12 while 5 and 9 start neglecting nodes 4 and 7 respectively. Meanwhile, 8 becomes a friend of 2 reestablishing a connection between both clusters.

As next step, we split the left group of nodes from the remaining transition. Those nodes gain connections to each other and 16 even gets connected to 13. This reduces the importance of 14 while increasing the importance of 13 as connection from the left cluster to the center cluster.

In the remaining transition there are no edge changes left, since within the cluster nothing changes and 8 got already connected to 2 in the first panel. Therefore, the last panel only shows the correction of the positions of the nodes in the right cluster to represent the new topology of the graph.

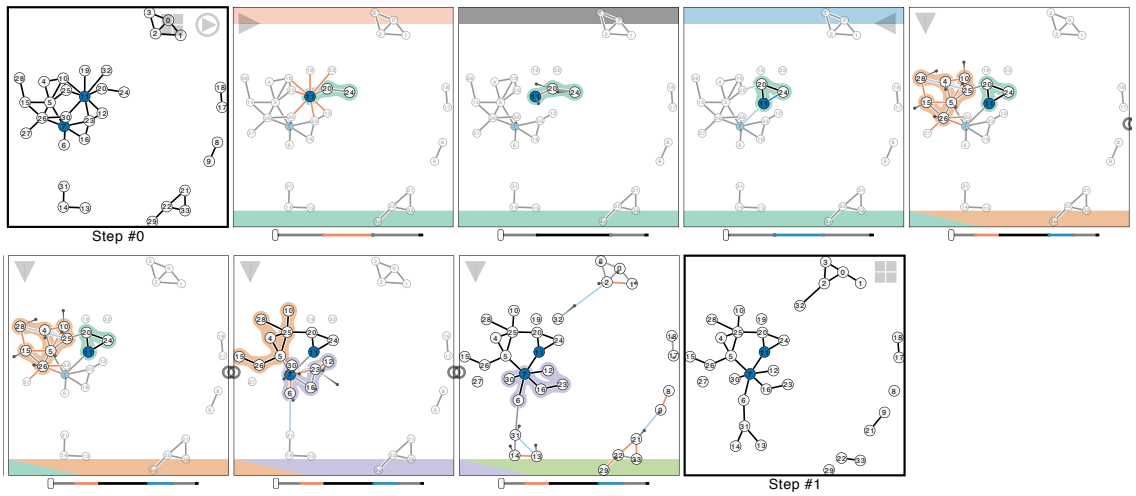


Figure 5.2: The annotated transition of interactions between attendees of the ACM Hypertext 2009 conference from before to after lunch on the first day as described in Section 5.2. Node 13 (upper) and 7 (lower) are selected.

5.2 Hypertext Conference Graph

In the face to face contact graph we want to explore how interactions between attendees of the conference change from before lunch of the first day to after lunch. Only contacts lasting at least 5 minutes are considered longer conversations and thus are selected. The graph can be seen in Figure 5.2 which shows the final composition used in the analysis.

When just looking at the full transition we can already see some facts. Nodes 18 and 17 only interact with each other. Also, the rest of the lower right side stays on its own. The bottom left sub-graph connects with the main graph after lunch, and the top right sub-graph makes an acquaintance with node 32.

Nodes 5 and 11 stick out as they obviously know a lot of people. Playing with the animation slider encourages to select 11, 20, and 24 for a split as they move together and are constantly connected. Splitting up this sub-transition further into phases reveals that 11, after being very communicative before lunch, retreats in the afternoon with 20 and 24. Indicated by the triangle connection between those nodes, with a high probability, all three are having a discussion with each other.

Turning our focus to node 5 we split using the top left part of the main graph. We see that 4, 5, and 25 form triangles both before and after lunch. However, before lunch they only converse together with 10, but not all at the same time since the four nodes do not from a clique. Node 28 shifts from talking to 15 to talking to 25 which he both already knows. If node 28 would for example be introduced to 25 by 15 they would share a triangle connection in one of the time steps.

Splitting with the last part of the main graph reveals that node 7, who is very active in the afternoon, was not active before lunch. Speculatively, 7 was running late and could therefore not be as active in the morning as he was in the afternoon. Moving the slider around for a bit confirms that node 6 is in fact not connected to 7 before lunch but rather connected to node 30, which was indefinite due to unfortunate node placement.

5.3 SIENA Friendship Graph

The graph comic can be seen in Figure 5.3. By only observing the start and the end stage of the graph in a traditional manner, we can quickly see that the big lower sub-graph is split into two separate parts connected only by one node. This node then also connects to the sub-graph in the upper part of the panel. More smaller sub-graphs connect also to the main graph.

We use this first observation and split with the top sub-graph first. We now see where this sub-graph gets connected while the connecting node is still in its initial position. We do this with the two remaining sub-graphs on the left side as well. The left sub-graph connects over one node via three edges with the main graph. The lower connects over one node. So far, only nodes have been of interest that do not decrease their degree.

Next, we focus our attention on the two nodes in the upper left corner with very strong movements. Those nodes do not connect to any other nodes, but will perhaps in later time steps. The movement can clearly be identified as artifact of the layout algorithm trying to position them in the most empty area.

Now it is time to clear up the main graph. While playing around with the slider we notice that the cluster bounded by the nodes 9, 10, 18, 28, and 29 has very consistent behavior. After splitting with it, however, the behavior is still too complex to grasp. So we split the transition into its phases. Now, we can see that the connections to the left part of the graph are cut completely and are reintroduced from node 18 via node 20 and 22. Also, nodes 14 and 15 are cut off and not yet reconnected.

We now turn our focus to the nodes in the upper right part of the node-link diagram. Node 22 loses its connection to the left part of the graph, which promotes edge $18 - 20$ to be the last connection connecting both parts of the graph. To confirm this we check the panel with the remaining changes. Furthermore, we can see that nodes 14 and 15 from the last split get reconnected to the upper part of the graph.

Splitting with the remaining nodes of the right side shows that node 14 directly connects the lower part of the right side to the upper part. The other two connections, $10 - 25$ and $28 - 41$, which do not change, only connect to the middle part. Note that 14 previously was a rather unimportant node at the border of the graph.

We can now merge again the individual parts of the right side and look at the complete transition of this part. This is now easier to follow, since we know where to look.

Turning to the left part of the graph we first split with 34 and 17. Then we see an interesting behavior of the rightmost bottom-most nodes of the left side. Splitting with these leads to a complicated change which we split into its phases. We can now see that the nodes 16, 21, and 23 change almost their complete neighborhood.

The remaining graph can be well interpreted. Although, due to the collinearity of the nodes 4, 20, and 31 it is not clear whether 4 and 20 share an edge. We can clear this up by splitting with node 31 and see that 4 and 20 are in fact only connected via node 31. Those two edges vanish to the next time step. For further analysis we merge the stages again.

Then node 34 connects with node 36 and pulls it down. Node 34 takes a similar role as node node 14 on the right side of the graph. However, this side has no middle part, and node 16 is equally important for the connection of this sub-graph.

Looking again at the complete transition from the start state to the end state of the graph. While appearing to be important in the initial state, node 20 only becomes important, connecting the left and right part of the graph, in the end state. Node 18 on the other hand has this role in both time steps.

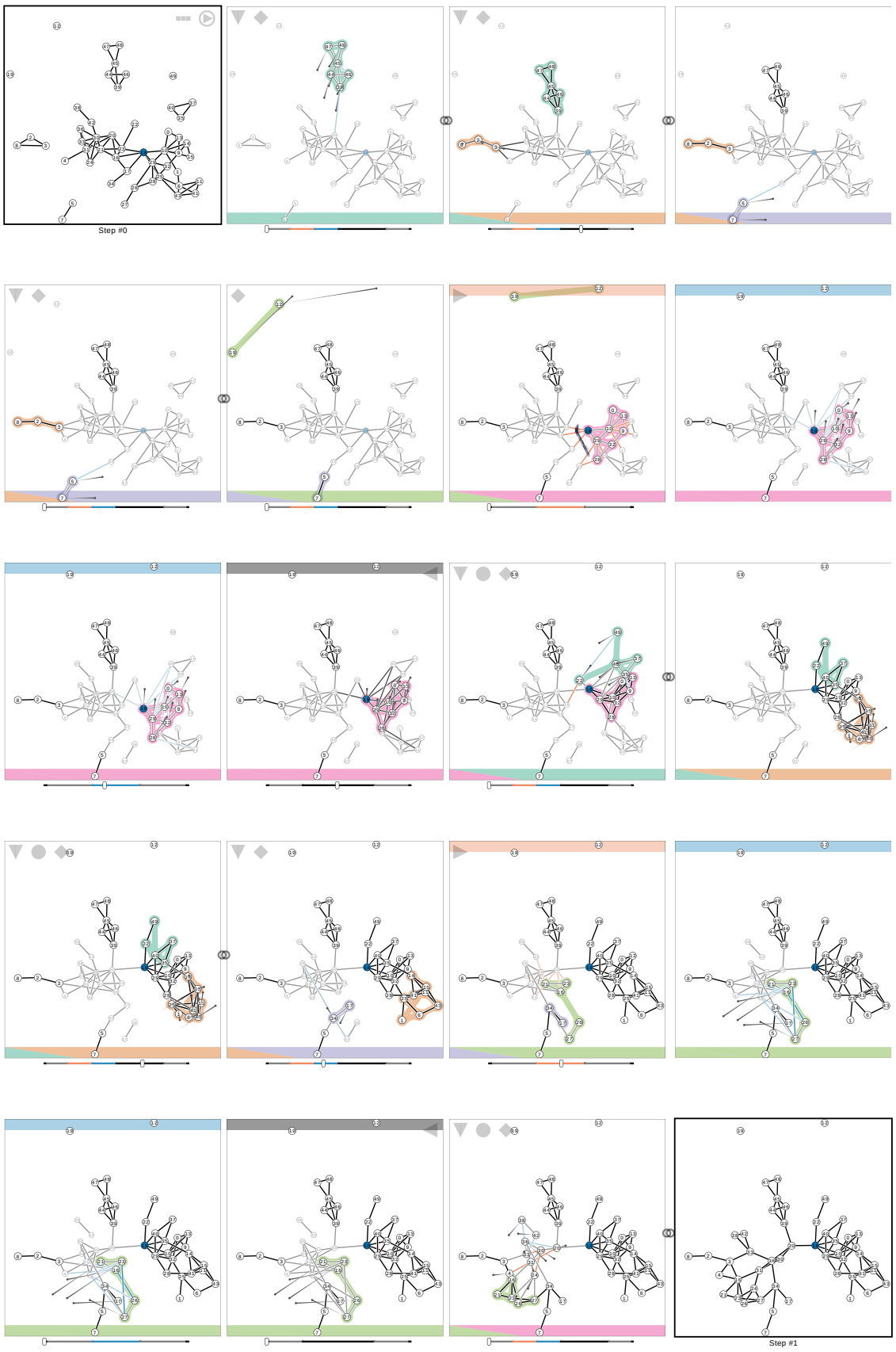


Figure 5.3: Analysis of the SIENA friendship graph. The panels are arranged as comic. Rightmost panels are copied to the beginning of the next row to consistently allow pair-wise comparisons between panels. Node 18 is selected due to its role mentioned in Section 5.3.

6. Discussion

This chapter discusses the usefulness of the technique for dynamic graph analysis. Conducting evaluations of the approach is difficult, because it aims on gaining full understanding of a dynamic graph. However, the usefulness of the prototype was shown with an analysis of three example graphs in Section 5.

The major advantage of node-link diagrams in graph analysis is the possibility to easily inspect non-statistical features and relationships beyond simple neighborhood in a graph (see Section 3.1.2). In dynamic graphs properties like new paths maintaining the connection of two clusters that would get separated otherwise or changes of connections within a cluster that may or may not disintegrate the cluster are from great interest. Those properties cannot be easily expressed as statistical values and go beyond the scope of local neighborhoods. Evaluating an application against interesting features of a dynamic graph is difficult.

Tasks like "Are two given nodes or clusters connected in one of both time steps?" are easily solvable by looking at both node-link diagrams and checking the connection. They don't require the graph to be dynamic either and cover only a small part of the interesting feature of the dynamic graph while assuming or implying to be embedded in a greater context.

However, generalizing the task to something like "Describe how the relations between two given clusters change" leaves too much freedom for a participant and might lead to results that are difficult to interpret.

Given that the prototype aims for creating full insight in the changes of a dynamic graph, evaluating the usefulness of the tool via task focused evaluations seems to be counter productive. Instead, the usefulness can be shown better by performing dynamic graph analyses on example graphs. This also reveals the other main strength of the prototype which is the practicality of being able to present the acquired knowledge about the dynamic graph to others.

In this thesis I presented a prototype for the analysis of dynamic graphs. The prototype is capable of presenting a dynamic graph with standard techniques, as small multiples or using animation to show changes. Also, visual annotations

like arrows indicating node movement can be used. Too many visual annotations, however, can lead to visual cluttering and information overload in a panel which adds complexity. Therefore, visual annotations can be selectively enabled. Increasing the pool of visual annotations to pick from, like using strike-out curves to show the removal of a group of edges, is left as future work.

The prototype utilizes a novel technique of splitting a transition into multiple stages to ease the understanding of the changes. Those sub-transitions are arranged in panels which can be reordered or be merged again. I showed the usefulness of the approach with three example graphs. However, careless use of splitting may lead to small sub-transitions resulting in loss of context. A good compromise between context of changes and their complexity has to be found by the user. Automatic splitting strategies, using heuristics, can overcome those limitations by providing a good compromise of complexity and granularity. Finding good heuristics for splits is left as future work, though. Also, sub-transitions may be interpreted falsely as valid intermediate step of the dynamic graph. However, this is mitigated by using a different visual representation for nodes and edges in temporary invalid configurations.

List of Figures

Bibliography

[1] Daniel Archambault, Helen C. Purchase, and Bruno Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Trans. Vis. Comput. Graph.*, 17(4):539–552, 2011.

[2] Benjamin Bach, Emmanuel Pietriga, and Jean-Daniel Fekete. Temporal Navigation in Dynamic Networks. In *IEEE Vis Week*, Seattle, USA, 2012.

[3] Patrick Baudisch, Desney Tan, Maxime Collomb, Dan Robbins, Ken Hinckley, Maneesh Agrawala, Shengdong Zhao, and Gonzalo Ramos. Phosphor: explaining transitions in the user interface using afterglow effects. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, UIST '06, pages 169–178, New York, NY, USA, 2006. ACM.

[4] F. Beck, M. Burch, C. Vehlow, S. Diehl, and D. Weiskopf. Rapid serial visual presentation in dynamic graph visualization. In *Visual Languages and Human-Centric Computing (VL/HCC), IEEE Symposium on*, pages 185–192, 2012.

[5] Ulrik Brandes, Vanessa Kääb, Andres Löh, Dorothea Wagner, and Thomas Willhalm. Dynamic www structures in 3d. *JOURNAL OF GRAPH ALGORITHMS AND APPLICATIONS*, 4(3):2000, 2000.

[6] Ulrik Brandes and Martin Mader. A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. *Proc. 19th Intl. Symp. Graph Drawing*, 2011.

[7] Ulrik Brandes and Dorothea Wagner. A bayesian paradigma for dynamic graph layout. In *Proceedings of the 5th Symposium on Graph Drawing, volume 1353 of Lecture Notes in Computer Science*, pages 236–247. Springer-Verlag, 1997.

[8] Till Bruckdorfer, Sabine Cornelsen, Carsten Gutwenger, Michael Kaufmann, Fabrizio Montecchiani, Martin Nöllenburg, and Alexander Wolff. Progress on partial edge drawings. In *Proceedings of the 20th International Conference on Graph Drawing*, GD'12, pages 67–78, Berlin, Heidelberg, 2013. Springer-Verlag.

[9] Stuart K. Card and David Nation. Degree-of-interest trees: a component of an attention-reactive user interface. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 231–245, New York, NY, USA, 2002. ACM.

[10] Christopher Collins, Gerald Penn, and M. Sheelagh T. Carpendale. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1009–1016, 2009.

[11] Kasper Dinkla, Marc J. van Kreveld, Bettina Speckmann, and Michel A. Westenberg. Kelp diagrams: Point set membership visualization. *Comput. Graph. Forum*, 31(3):875–884, 2012.

[12] Pierre Dragicevic, Anastasia Bezerianos, Waqas Javed, Niklas Elmqvist, and Jean-Daniel Fekete. Temporal distortion for animated transitions. In Desney S. Tan, Saleema Amershi, Bo Begole, Wendy A. Kellogg, and Manas Tungare, editors, *CHI*, pages 2009–2018. ACM, 2011.

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

[14] Carsten Friedrich and Peter Eades. Graph drawing in motion. *Journal of Graph Algorithms and Applications*, 6:2002, 2002.

[15] Marco Gaertler, Robert Görke, Dorothea Wagner, and Silke Wagner. How to cluster evolving graphs. In *Proceedings of ECCS*, volume 6. Citeseer, 2006.

[16] Emden R. Gansner, Yehuda Koren, and Stephen North. Graph drawing by stress majorization. In *GRAPH DRAWING*, pages 239–250. Springer, 2004.

[17] Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *Proceedings of the IEEE Symposium on Information Visualization*, INFOVIS '04, pages 17–24, Washington, DC, USA, 2004. IEEE Computer Society.

[18] Jeffrey Heer and Stuart K. Card. DOITrees Revisited: Scalable, Space-Constrained Visualization of Hierarchical Data. In *Advanced Visual Interfaces*, pages 421–424, 2004.

[19] Jeffrey Heer and George Robertson. Animated transitions in statistical data graphics. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 13:1240–1247, 2007.

[20] Marc Khoury, Yifan Hu, Shankar Krishnan, and Carlos Scheidegger. Drawing large graphs by low-rank stress majorization. *Comp. Graph. Forum*, 31(3pt1):975–984, June 2012.

[21] Josua Krause. Set outline library. https://github.com/JosuaKrause/Bubble-Sets/.

[22] John Lasseter. Principles of traditional animation applied to 3d computer animation. *SIGGRAPH Comput. Graph.*, 21(4):35–44, August 1987.

[23] Yi-Yi Lee, Chun-Cheng Lin, and Hsu-Chun Yen. Mental map preserving graph drawing using simulated annealing. In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation - Volume 60*, APVis '06, pages 179–188, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[24] Scott McCloud. *Making Comics: Storytelling Secrets of Comics, Manga and Graphic Novels*. William Morrow Paperbacks, 2006.

[25] Ken Perlin and David Fox. Pad: an alternative approach to the computer interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 57–64, New York, NY, USA, 1993. ACM.

[26] Catherine Plaisant, Jesse Grosjean, and Benjamin B. Bederson. Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02)*, INFOVIS '02, pages 57–, Washington, DC, USA, 2002. IEEE Computer Society.

[27] Helen C. Purchase and Amanjit Samra. Extremes are better: Investigating mental map preservation in dynamic graphs. In Gem Stapleton, John Howse, and John Lee, editors, *Diagrams*, volume 5223 of *Lecture Notes in Computer Science*, pages 60–73. Springer, 2008.

[28] George Robertson, Roland Fernandez, Danyel Fisher, Bongshin Lee, and John Stasko. Effectiveness of animation in trend visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14:1325–1332, November 2008.

[29] Peter Saffrey and Helen Purchase. The "mental map" versus "static aesthetic" compromise in dynamic graphs: a user study. In *Proceedings of the ninth conference on Australasian user interface - Volume 76*, AUIC '08, pages 85–93, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[30] Simulation Investigation for Empirical Network Analysis. Teenage friends and lifestyle study. http://www.stats.ox.ac.uk/~snijders/siena/s50_data.htm.

[31] SocioPatterns. Hypertext 2009 dynamic contact network. http://www.sociopatterns.org/datasets/hypertext-2009-dynamic-contact-network/.

[32] Barbara Tversky, Julie Bauer Morrison, and Mireille Bétrancourt. Animation: can it facilitate? *Int. J. Hum.-Comput. Stud.*, 57(4):247–262, 2002.

[33] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J.J. van Wijk, J.-D. Fekete, and D.W. Fellner. Visual analysis of large graphs: State-of-the-art and future research challenges. *Computer Graphics Forum*, 30(6), 2011.

[34] Colin Ware, Peter Mitchell, and John Kelley. Designing flow visualizations for oceanography and meteorology using interactive design space hill climbing. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, San Antonio, TX, USA*, pages 355–361. IEEE, 2009.

[35] Ka-Ping Yee, Danyel Fisher, Rachna Dhamija, and Marti Hearst. Animated exploration of dynamic graphs with radial layout. In *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, INFOVIS '01, pages 43–, Washington, DC, USA, 2001. IEEE Computer Society.

[36] Douglas E. Zongker and David H. Salesin. On creating animated presentations. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 298–308, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.