Universität Konstanz

Chair of Computer Graphics and Media Design
Department of Computer Science

Bachelor Thesis

Annotation of Changes in Evolving Graphs

Annotation veränderlicher Graphen

for obtaining the academic degree
Bachelor of Science (B.Sc.)

Field of study:	Information Engineering
Focus:	Graph Drawing

by

Josua Krause
(01/697048)

First advisor:	Prof. Dr. Oliver Deussen
Second advisor:	Prof. Dr. Ulrik Brandes

Konstanz, November 10, 2011

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, November 10, 2011

Josua Krause

Abstract

In evolving graphs a great amount of changes can occur, which is difficult to follow at once. Nodes may appear or vanish and edges change connections between them, merging or separating parts. Besides the current practice of animating between time steps, annotations in the graph can be used to help the reader to better understand these changes. In this thesis, static visual representations of graph changes are presented. At first, a case study was conducted to find a common sense in manual graph annotations. Automatic graph annotation methods are developed to detect and highlight node movements, intra-cluster connectivity changes, and vanishing edges between clusters. An overview of visual annotation metaphors is given. Finally, examplary annotated graphs are shown in different styles, serving individual presentational demands.

Contents

1. Introduction

Analyzing and visualizing evolving graph structures underlies rising interest in a very broad range of scientific domains, like social sciences, natural sciences, or visual analytics. Tracking complex changes between evolving steps is still difficult due to limitations of human perception. In this thesis I describe methods to cope with the described problem by static annotation of node link diagrams as one representation of evolving graphs.

Graphs can be represented as adjacency matrices or as node link diagrams. For observing time-step-wise changes in evolving graphs, adjacency matrices can be compared. Visualizations can aim to highlight the changes explicitly, either by a difference matrix, or row and column reordering. But, for comparison on different levels of detail, they do not scale properly.

Node link diagrams (hereafter implied by the term: graph representation) are more intuitive. When arranged by a sufficient layout algorithm, the node positions and linked edges allow fast perception (the features "position" and "direction", according to Cleveland & McGill [5]) and Gestalt laws can be utilized for visualizations, e. g. giving closeness of nodes a meaning.

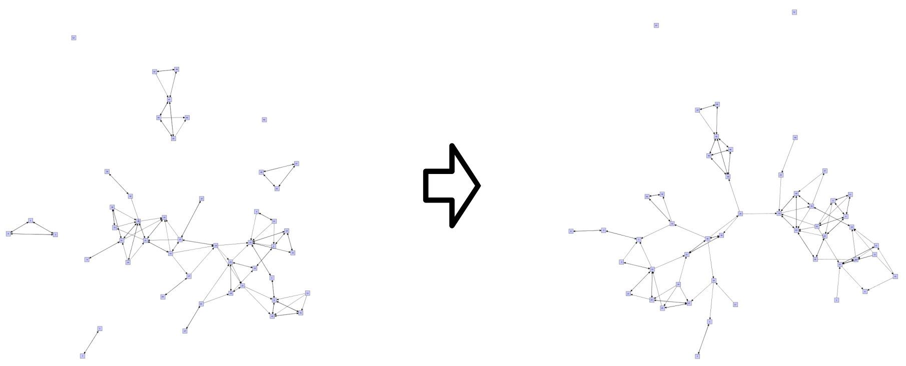


Figure 1.1: A graph, evolving over time.

The connection between graph representations at successive time steps is not easy to conceive, as demonstrated in Figure 1.1. Therefore visual aids are needed to allow fast tracking and quick overviews of the changes that led to the differing representations.

A common aid is the animation of changes between two consecutive time steps. However, the amount of changes also makes it difficult to perceive the animation, since it is difficult to follow movements of too many distinct items [14]. In my work I will focus on static annotations predicting changes to the next time step, in a graph with a data driven layout. Such layouts are assumed to only show logical updates, i. e. no intermediate layouts are allowed, taking only some of the changes between two time steps into account [4].

1.1 Types of changes

Changes in graphs can be divided into two types: Node and edge changes. In these two categories several finer types can be distinguished. Node changes can be nodes appearing or disappearing, and nodes moving from one time step to the next. The movement of nodes can be seen as an indirect change, since it depends on the layout of the graph. In the given graph layout, where the visual distance between two nodes optimally reflects the distance in the graph (i. e. shortest path), the movement of the nodes is produced by changes of the edges. The graph layout is further discussed in Chapter 2.1.

Changes of edges in a graph are just the creation or removal of them. But regarding the importance of an edge change, there are two types of those changes: Inter- and intra-cluster. Inter-cluster changes either strengthen the connection between two clusters up to merging them, or weaken the connection up to completely separating the two clusters. This often leads to a movement of the entire cluster. Intra-cluster changes only have influence on the connectivity within the cluster.

1.2 Granularity

Annotations can vary in their granularity. Showing all changes by themself, e. g. , marking every vanishing and creating edge, and displaying the movements of every individual node, is a very granular and exact annotation. It does not enhance the readability of the graph, though, nor does it provide any context for global changes.

Simplifying all changes leads to schematic annotations. For example showing the overall connectivity change of the graph gives the overall context of the individual edge changes and simplifies them to one statistical feature. A drawback is the loss of precision, though.

The optimal annotation would be a trade-off of granularity, providing the greatest expressiveness, i. e. visualizing schematic annotations with sufficient exactness.

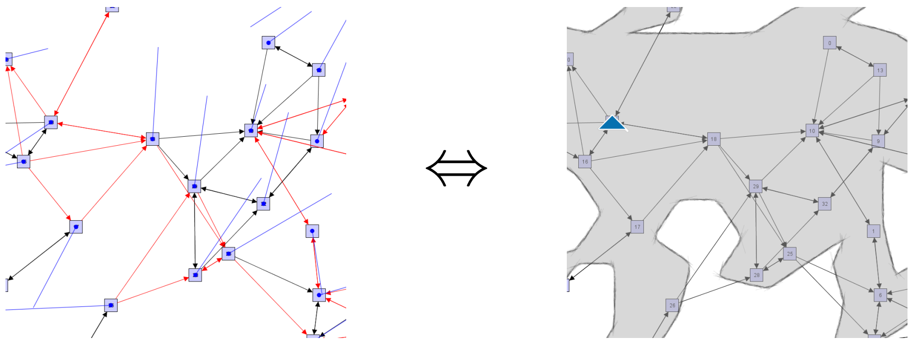


Figure 1.2: The trade-off between detailed (left) and schematic (right) annotations. On the left, vanishing edges are drawn in red and the absolute movement of the nodes is annotated in blue. On the right, just the tendency of the connectivity in the whole graph is shown, indicated by an arrow.

In this thesis the main focus lies on showing intra-cluster connectivity changes, cluster movement, and inter-cluster edge removal. This gives a good granularity trade-off. By ignoring individual edge changes in clusters and showing a summary of them as connectivity change, the concept of the edge changes in a cluster is displayed and individual edge changes do not distract the reader. The cluster movements show the tendency of clusters to merge or disconnect. The annotation of inter-cluster edge removal can illustrate the motifs of this movements. By visually cutting more than one disappearing edge, the context of the removal for the disconnection of clusters can be seen.

1.3 Related Work

Current graph drawing techniques for dynamic graphs focus solely on optimizing layouts for successive graphs. The main goal is to create a consistent, stable, and readable dynamic graph layout [4]. In order to achieve this, layouts try to preserve the mental map of the reader inbetween time steps [8]. Reducing node movements leads to this preservation [12]. However, preserving the mental map can lead to false assumptions about the graph [13] in some cases, or a poor layout performance in later time steps [15]. Saffrey & Purchase [15] suggest a compromise between mental map preservation and good layouts, which is for example provided by Brandes *et al.* [3].

To show the changes of dynamic graphs, animation is used [4][3][2][8]. However, animations can be challenging to analyze. Often, multiple replays are needed to find parts worth focussing [14]. Too many, or asynchronously moving data points can be confusing [14]. Baudisch *et al.* [1] remark that static depictions of motion perform at least as well as animation, while helping to process changes better.

2. Preliminary Work

In this chapter, the projection algorithm for the graphs in this thesis is described. The results of a case study, aiming to find a common sense, are discussed. Also some straight forward annotations are investigated.

2.1 Projection Algorithm

Before annotating a graph, a good node placement has to be found. The quality of the annotations is directly influenced by the layout. Therefore, the layout algorithm should reflect changes in data adequately. Otherwise the annotations would only reflect the changes of the layout. For example, a big movement caused by only a single edge change in a highly redundant graph area leads to wrong assumptions about the changes in the graph, whereby annotations of this movement would become futile.

The graph layout used in this thesis is a stress minimizing layout with anchoring [3]. Stress is defined by the difference between visual distance and distance in the graph, i. e. the length of the shortest path between nodes. In order to get an anchored layout, the following adjusted stress function is used:

$$\text{stress}^{A}_{\alpha}(P^{(t)}) = (1-\alpha)\cdot \underbrace{\text{stress}(P^{(t)})}_{\text{quality}} + \alpha \cdot \underbrace{\sum_{i\in V} \phi_i^{(t)} \|p_i^{(t)} - p_i\|^2}_{\text{stability}}$$

where $P = (p_i)_{i\in V}$ is a reference layout, retrieved for example by the previous time step or the average graph layout. $\phi_i^{(t)}$ is used to fix a node to the reference point differently strong. The ratio between stress minimization and anchoring is given by α.

Minimizing the stress function leads to consistent, stable, and readable layouts [4]. This compromises between mental map preservation and a good layout (see Chapter 1.3).

2.2 Case Study

In order to find a general consent in manual graph annotations, four graph drawing experts and four non-professionals were asked to annotate an evolving graph manually.

The participants were asked to manually annotate the given evolving graph. No restrictions, what or where to annotate, were made. The example graphs were chosen from real life evolving graphs and synthetic graphs showing clear common patterns like splitting clusters.

2.2.1 Results

The main observation of the case study was that there are only few common senses in annotating layouted graphs. Surrounding groups of nodes is common. Although, the outlines serve different purposes. Another recurring technique is to use arrows for depicting motion of nodes.

The manual annotations can also be analyzed in terms of granularity (see Figure 2.1 and its explanation).

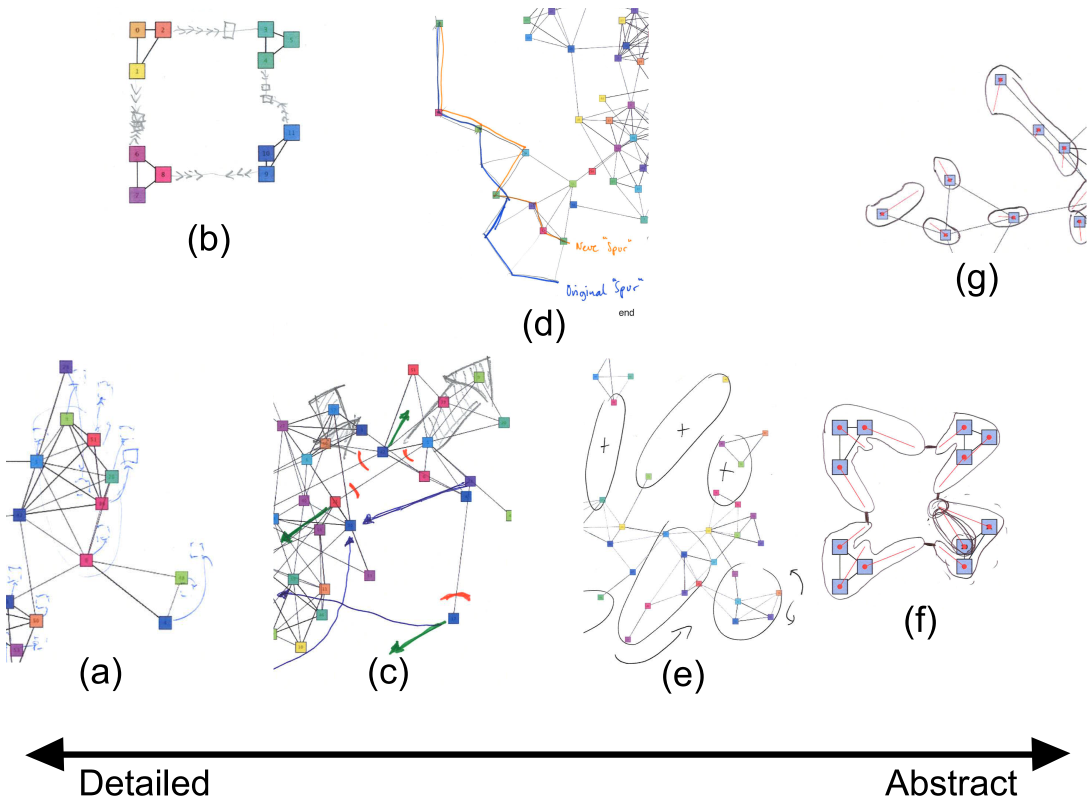


Figure 2.1: Example results of the case study. The position on the x axis determines the granularity as described in Chapter 1.2.

The following table explains the results, shown in Figure 2.1, in more detail.

	Description	Granularity	Discussion
(a)	Showing every node movement by its own.	Very granular and exact.	Only node movements are taken into account. Due to high grade of detail, similarities in the movements are not hinted.
(b)	Combining edge creation and node movement.	Very granular and exact.	The difference between movement and edge creation is not intuitive. It is not feasible for larger graphs.
(c)	Noteable node movements and edge creation depicted as differently colored arrows. General movements of areas shown with larger arrows. Disappearing edges striked out.	In case of the individual nodes, the annotations are exact. The large arrows are not as exact since they combine the movement of several nodes. Overall it is less granular.	The use of the same metaphor (arrows) for different purposes can lead to confusion. Edge changes are not set into context.
(d)	Using a trail along nearby nodes to show the change of it from one step to the other.	Node movements are combined. The individual movement of each node can only be assumed.	The usefulness is restricted due to the difficulty to find good trails. Edge changes are not taken into account.
(e)	Similarly moving nodes are summarized. Nodes that will be connected are circled together. Arrows provide the moving direction and a ”+” distinguishes edge creating circles from others.	The changes are largely summarized.	The arrows give a hint of the movement and newly created edges are put into context.
(f)	Enlarging the area around nodes to their new position. Grouping of similar moving nodes. Showing new connections between the clusters.	Movements are summarized and edge creations are hinted.	Movements are relatively exact, even though they are summarized. In combination with the individual movements of the nodes (red lines), which were not part of the manual annotation, level of detail is provided.
(g)	Enlarging the area around nodes to their new position. Grouping of similar moving nodes.	Movements are summarized.	Similar to (f). However, no hints about edge changes are given.

2.3 Straight forward approaches

In the following, straight forward annotation techniques are discussed, in order to show the need for more advanced approaches.

2.3.1 Node movement

A basic form of annotating node movements is to connect start and end points of cluster members with arrows. This adds one additional observable entity per node. The resulting visualization is heavily cluttered.

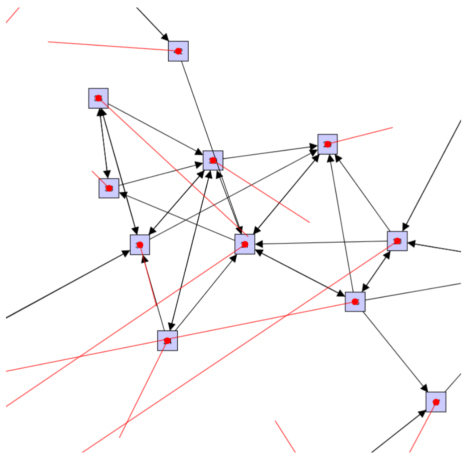


Figure 2.2: A straight forward movement annotation. Movements of the nodes are represented as lines. The end point of a line is the next position of the respective node.

2.3.2 Node metaphors

In an evolving graph not only the edges change but also nodes may appear or vanish.

Normally graph annotations should only predict the future of the graph, but in the case of nodes it is helpfully to also show what happened before, i. e. that new nodes, in this time step can be identified, or that a node existed at a spot in the previous step but does not exist anymore.

An idea to show possible metaphors for edge creation and removal is depicted in Figure 2.3.

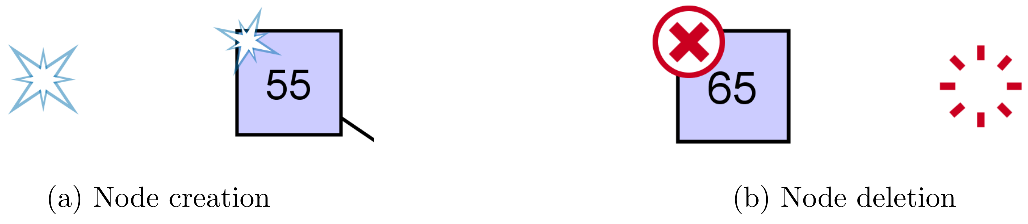


Figure 2.3: Metaphors for emerging and vanishing nodes. The left side of each picture shows what will be happening in the next time step, the right side refers to the previous one.

2.3.3 Disappearing edges

In order to annotate disappearing edges in a simple way, those edges can be marked as vanishing. Using colors or striking them through is an intuitive method. This leads to not relateable edges in crowded areas. In not well connected areas the context of the disappearing edge is lacking. For example, it is hard to see relations between vanishing edges, as they occur when clusters are disconnecting.

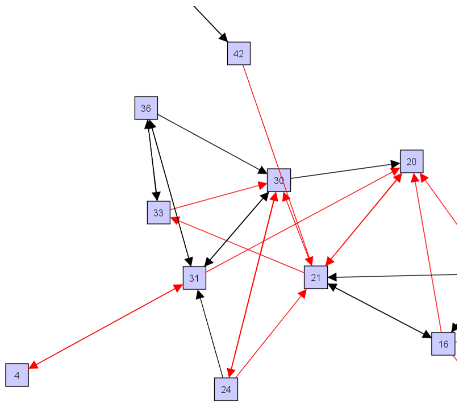


Figure 2.4: Disappearing edges in the graph are colored in red. All other edges are not altered.

2.3.4 Emerging edges

One way to show emerging edges would be to draw the upcoming edges and mark them to be easily identified as new. This would predict the next time step of the graph very well, due to the fine granularity. However, it does not show how emerging edges are related. Another problem is that the position of the nodes are not yet adjusted to the new connectivity situation, therefore leading to long edges across the whole graph. These long edges introduce additional clutter.

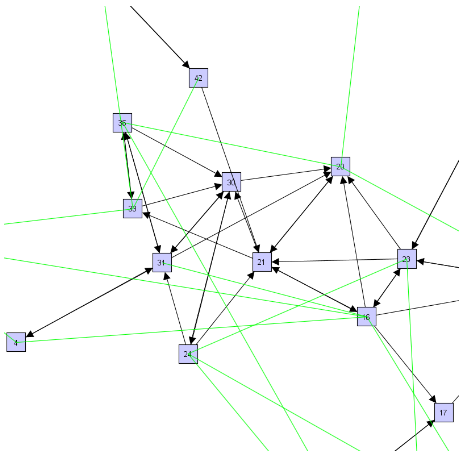


Figure 2.5: In addition to the current edges, emerging edges are drawn in green. The new edges are not yet present in the graph, but will be in the next time step.

3. Node changes

In order to simplify node based annotations, parts of an evolving graph can be clustered. There are different relationships between nodes, that lead to different clusters.

3.1 Clustering based on projected positions

As data driven layouts introduce movements of nodes, one relationship between nodes is the similarity of that movement. In the graph layout, nearby nodes are likely to be strongly connected. On the other hand, weak connectivity does not imply long distances between nodes. Edge changes alter the connectivity of nodes, which leads to their movement. Neighboring nodes are likely to move in a similar direction when edge changes do not influence their connectivity.

Clustering similarly moving neighbors therefore allows annotation of global trends within a graph. Since the node movement is influenced by the edges, the movement of such clusters define the interaction between them.

3.1.1 Generating continuous movements

The movement of nodes is described by the direction and the length of the movement, and the position of the node. A good similarity measure should account for all of those three components. Encoding the position of a node in the similarity measure is a problem, since it has to be weighted against the actual movement of the node. In order to circumvent this problem, a continuous representation of the node movements over the complete plane of positions can be used. The continuity can be achieved by interpolating between the movement vectors of the nodes.

An effective method to interpolate between movement vectors is the use of kernel functions. This can be done by summing up all movement vectors of all nodes, weighted by the distance to the nodes at this point. The weighting function can be chosen freely. A linear function would result in a rough field, but the nodes have limited influence within a certain radius. A better kernel function is a Gaussian curve which results in a very smooth field. The influences of the nodes are also limited, because the exponential function gets near 0 after a certain distance.

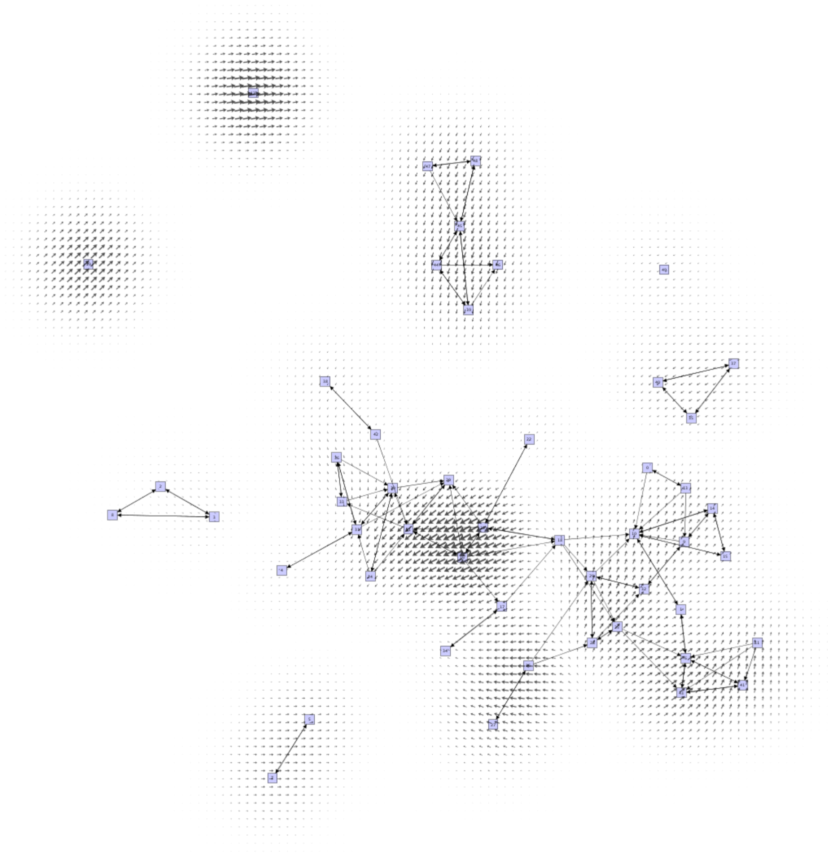


Figure 3.1: The vector field of node movements, interpolated with a kernel function.

The resulting vector field does not depend on the visual ordering of the nodes nor the distance from nodes to each other. Outliers and nodes with small movements may be occluded by surrounding nodes with larger movements.

3.1.2 Clustering the vector field

Having a continuous vector field, the distance measure of the clustering can concentrate solely on the moving direction and length. Telea *et al.* [16] have a good approach, using an elliptic similarity function.

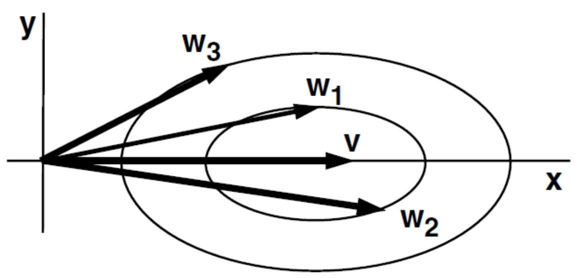


Figure 3.2: The ellipsoid based vector measure. The coordinates are rotated, such that vector v points to the right. Lying on the same ellipsoid, w_1 and w_2 have the same distance to v. w_3 however has a larger distance.

To keep the possibility of changing the granularity of an annotation, hierarchical clustering [11] is used to cluster the vector field. Attempting this goal directly along the edges of a graph can lead to heavily distributed clusters since an edge does not necessarily mean a visual closeness in the layouted graph. To avoid a direct weighting of the distances between nodes, the previously generated continuous movement field can be used and separated in a grid-like manner which is then merged by a hierarchical clustering [16].

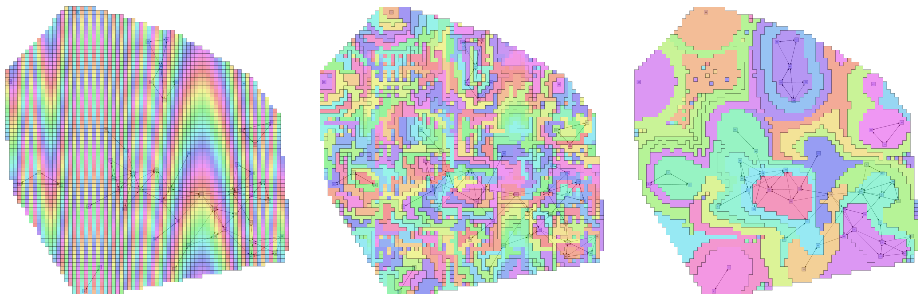

(c) Initial configuration (d) After 10 steps (e) After 140 steps

Figure 3.3: The hierarchical clustering after a certain number of merge steps per cluster.

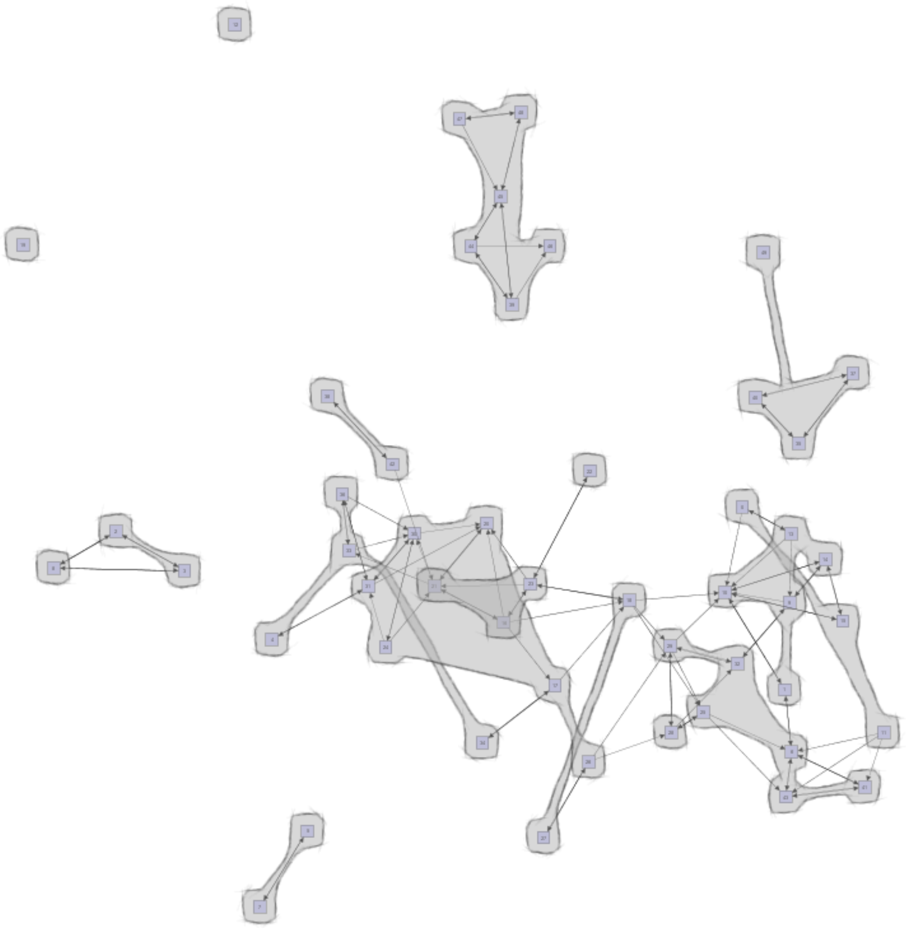


Figure 3.4: The final clustering based on interpolated node movements. Bubble Sets are used to render the outlines of the clusters.

3.2 Clustering based on the graph structure

For non-evolving graphs many clustering methods based on the graph structure are known. In the case of dynamic graphs, one method to cluster a graph is to aggregate the graphs and then cluster it. This represents the overall connectivity between both time steps most accurate. However, since the layout positions of the graph are computed according to the first time step, the cluster have little spatial proximity. This leads to a cluttered representation.

Another method is to cluster the first graph only. The changes to the next time step can be shown with other annotations (see Chapter 4.1).

In this thesis I use the Markov Clustering algorithm [17]. Though, other feasible clustering algorithms would work as well. The Markov Clustering algorithm uses a deterministic way to compute the probabilities of random walks in the graph. These random walks are then used to find clusters, using the property that the number of longer paths between two arbitrary nodes of a natural cluster is high [7]. The result of the clustering is shown in Figure 3.5. This clusters are much more distinct than the movement based clusters Figure 3.4.

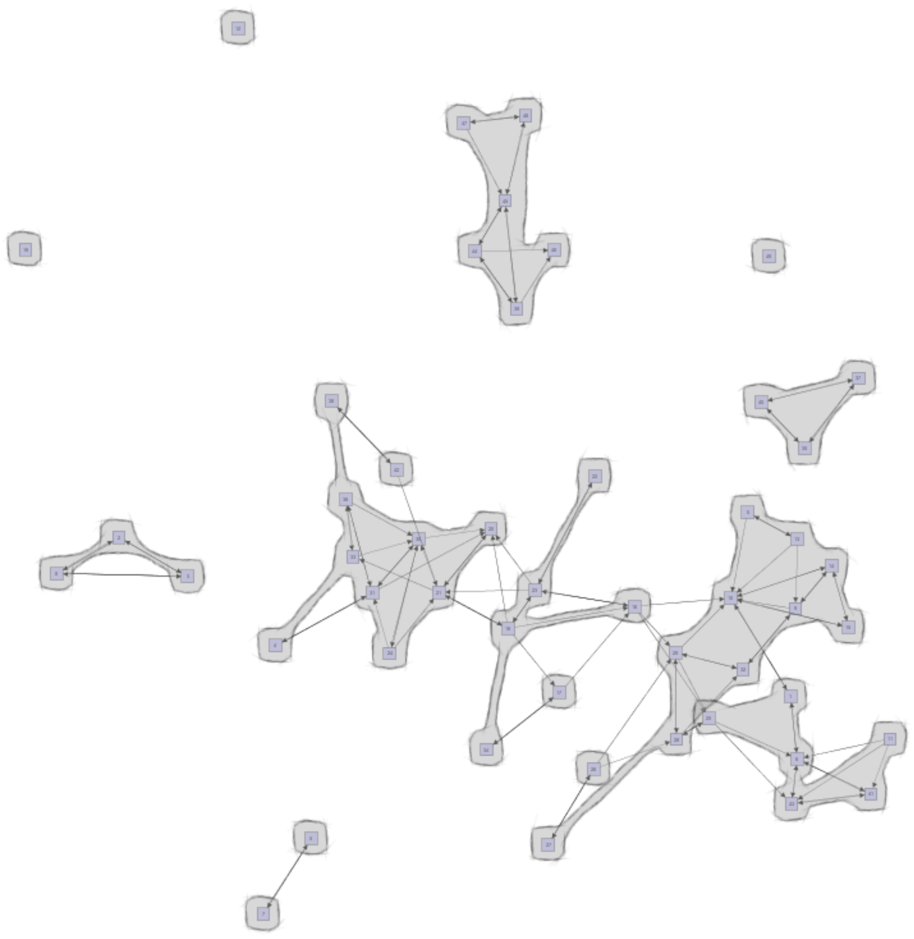


Figure 3.5: A clustering generated by the Markov Clustering algorithm. Bubble Sets are used to render the outlines of the clusters.

3.3 Visual representation of clusters

Having defined clusters, it is necessary to visualize the membership of nodes. One method would be to assign each group a color and mark every node with the color defined by its group. The coloring can reduce the set of visual features to encode other informations in a graph. One way to avoid this is to surround the members of a cluster by an outline.

3.3.1 Outline creation

The outline of cluster members can be the convex hull. Alternatively, Bubble Sets [6] produce concave outlines of cluster membership.

Convex Hull

To generate a convex hull, the Graham's Scan algorithm [9] can be used.

Bubble Sets

Bubble Sets [6] are an implicit boundary of visual items, i. e. represented as rectangles or lines. They are computed by generating a potential field around those items and connecting distant parts with delaunay based lines. A great advantage is the possibility of avoiding items that are not part of a cluster, up to bending lines between items.

Comparison of convex hull and Bubble Sets

A convex hull can be fast to compute, but since nodes may be distributed, or clusters can be interleaving, the convex hull produces overlaps which make it difficult to read them properly (see Figure 3.6). Bubble Set outlines try to avoid each other, thus minimizing overlaps. This makes it easier to distinguish clusters. However, overlaps are still possible (see Figure 3.4).

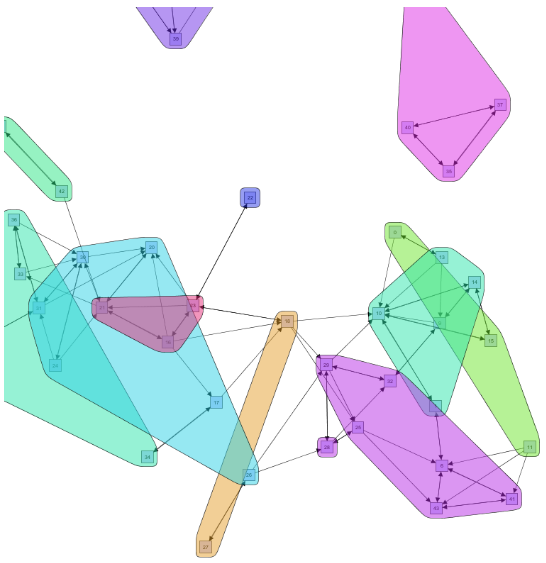
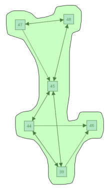
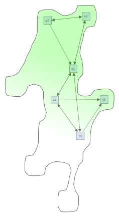
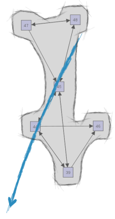
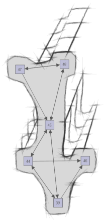
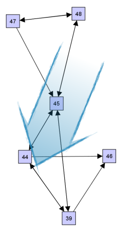


Figure 3.6: Multiple convex hulls overlapping, making it difficult to distinguish them. The colors of the areas are chosen randomly to ease the discriminability of the clusters.

3.3.2 Cluster movement

Having a visual representation of clustered nodes, informations about their movement would be a helpful addition. The following table explains some techniques to show the movement of clusters.

Type	Description	Benefits & disadvantages
Color coding 	The direction of the movement correlates with colors on the color circle. The length of the movement is coded with the saturation of the color.	Since the color direction mapping is not very intuitive, a color key is needed. Additionally the saturation acts only as a relative measurement. However this technique can be used as an addition to other methods of movement annotation, when the color of a cluster area is free to choose.
Larger Areas 	Extending the area of cluster outlines to end positions of the node movements.	This method does not rely on average movements and is therefore exact, with respect to new node positions. Yet, the exact new position of an individual node can not be determined. Due to the expansion of the cluster area, clutter can be introduced from overlapping cluster regions.
Arrows 	Showing the movement with arrows.	There are many possibilities to draw arrows showing the movement. The start position of the arrow could be chosen at the border of the outline or the center of the area. The length is also not quite definite. The mean movement of all nodes can be chosen, or the tip of the arrow can for example be at the same relative position as the start but on the final destination of the outline. Therefore the arrow does not represent the exact movement, but the direction and a hint of how long the movement is.
Motion lines 	Motion lines are a common metaphor for movement in comics. They are drawn behind the moving object.	Motion lines are well established in comics to give a hint of the direction. However, the length of a movement is not addressed very well. Since the lines are in the opposite direction, the end of the line can not hint the new position of a moving area.

Arrows (no outline) 	Without drawing the outline, arrows can be placed directly at the center of the area.	Missing the outlines, the picture gets less crowded. On the other hand it gets less clear which nodes are contained. Due to this, the arrows cannot exceed the area of the nodes without risking to include other nodes that are not part of it. Therefore an arrow, pointing directly to the new position, is generally not possible. However, a hint of length and direction can be made.

4. Edge changes

Graphs have changes of node connections, i. e. changes of edges. Having defined clusters, two types of edge changes are introduced: Intra- and inter-cluster changes. Individual intra cluster edge changes are less important than the overall change of connectivity in the cluster. For inter cluster edge changes, it is of interest to know where edges between clusters are created and which edges are vanishing.

4.1 Showing connectivity changes in clusters

In areas of a graph with a high connectivity, showing individual emerging and vanishing edges can become very cluttered. Since these regions are already well connected the creation or the removal of a few edges has only little influence in the connection of parts and serves more for the inner connectivity of such a cluster. Therefore, the display of particular changes can be replaced by an estimation of the overall connectivity change.

4.1.1 Computing the connectivity change

In order to determine the change of connectivity in a cluster, the edge density of the cluster in both time steps can be computed. The difference between those values then is a measure for the change in the overall connectivity of the cluster.

The edge density of a cluster d_c in the directed graph $G = (V, E)$, where $V_c \subseteq V$ is the set of nodes in the cluster and $E_c = E \cap (V_c \times V_c)$ the set of edges within the cluster, is computed as follows:

$$d_c = \frac{|E_c|}{|V_c|(|V_c| - 1)}$$

where d_c is the ratio of edges, to the number of edges of a fully connected cluster of the same magnitude.

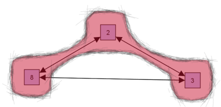
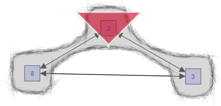
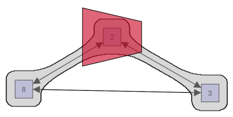
However, this measurement does only account for the number of edges in the cluster and ignores that edges can be differently important within a cluster, e. g.

separating the cluster by removing an edge. In order to reflect such behaviours, other measurements have to be used.

Clusters are not fully connected in the most cases. Using the mentioned measurement, this can result in small values, making it difficult to represent them. A logarithmic scale can be used to avoid this.

4.1.2 Visual representation of connectivity changes

To present the connectivity changes as annotations, different approaches were used.

Type	Description	Benefits & disadvantages
Color coding 	Showing connectivity changes as cluster outline colors. Blue representing increasing connectivity and red representing the decrease of it. The intensity can be coded as the saturation of the color.	Choosing arbitrary colors, a key is needed. Additionally the saturation acts only as a relative measurement. However this technique can be used as addition to other methods of movement annotation, when the color of a cluster area is free to choose.
Arrows 	Using upward and downward arrows, depicting increase or decrease of connectivity. The effect can be emphasized with colors.	Coding the strength of change into the size of the arrow is not easy to read. The shown changes are only relative.
Trapezoid 	A trapezoid can show the absolute connectivity change of a cluster. The length of the left line projects the connectivity at the first time step and the length of the right line projects the connectivity at the second time step. Again, colors can be used to emphasize the changes.	Assuming a left-to-right text direction, the metaphor is intuitively graspable. However, the width of the trapezoid can change the steepness of the lines and therefore the preattentive anticipation of connectivity changes.

4.2 Summarization of disappearing edges

The change of inter-cluster edges can be very informative regarding groups of edges. For example, disappearing edges can lead to separation of clusters. Seeing this change without its context can be misleading, since one vanishing edge may just slightly affect the connection of those clusters. However, the contexts of edge changes are difficult to find visually, when they are not hinted to the reader. Therefore a good way of showing these relations is to connect relating edges by striking them out together.

Appearing edges also have these relations. Although, the naïve display of them leads to a cluttered drawing, since the nodes are in a position that is not yet affected by the new edges. A possible way of showing the appearance would be to investigate the evolution of the graph in reverse order and examine the disappearance. Therefore, this chapter focuses on disappearing inter-cluster edges, by finding suitable strike out curves.

There are many ways to generate edge strike out curves. Unfortunately, the understanding, what a good strike out curve is, diverges. Having a profound algorithm does not necessary mean getting a visually effective result.

4.2.1 Heuristical edge strike out curves

Edge strike out curves can be generated using heuristical methods.

Algorithm and visual representation

The algorithm to generate such curves is split into phases. In each phase, one curve is generated. A phase starts by taking points on the edges to combine them to curves (the number of points per edge is later called edge resolution). Out of this set of curves, the best curve, according to the heuristic, is taken. The edges cut by this curve are removed from the set of edges and the next phase starts, when this set is not empty. A pseudo code representation of this method is described by algorithm 1.

Algorithm 1: A general way of computing heuristical strike out curves. $E_{\text{vanishing}}$ is the set of vanishing edges and E_{other} is the set of the other edges present at the current time step. A tuple (Edge e, Number$n$) describes the n^{th} point on the edge e. The maximum of n is the edge resolution r. This algorithm uses the function *nextCurve*, which is described in algorithm 2.

```
$C \leftarrow \emptyset$
while $E_{\text{vanishing}} \neq \emptyset$ do
  $c \leftarrow$ null
  $t \leftarrow$ null
  loop
    $t \leftarrow$nextCurve($t, E_{\text{vanishing}}$)
    if $t$ = null then
      break
    end if
    if $t$ intersects any curve $\in C$ then
      continue
    end if
    if $t$ intersects any edge $\in E_{\text{other}}$ then
      continue
    end if
    if $c$ = null $\vee$ heuristic($t$) $>$ heuristic($c$) then
      $c \leftarrow t$
    end if
  end loop
  if $c \neq$ null then
    remove all edges $\in c$ from $E_{\text{vanishing}}$
    add $c$ to $C$
  end if
end while
return $C$
```

Algorithm 2 shows how to generate all possible curves, by defining an implicit ordering of the curves. For a curve, the algorithm computes the next one. It works by successively extending the curve, until she reaches its maximum length. When this happens, the points on the last edge are iterated. After that the next edge is taken at the rearmost position of the curve, shortening it when no next edge is available.

Algorithm 2: This algorithm describes the function *nextCurve*, used in algorithm 1. The function takes the current curve c, which may be **null**, and the remaining set of vanishing edges $E_{\text{vanishing}}$, which has to have a fixed ordering. A curve is a list of tuples (Edge e, Number$n$). Each edge is segmented in equidistant points. The tuple describes the n^{th} of this points for edge e. n is limited by the edge resolution r.

```
if $c = $ null then
  return $[(\text{firstEdge}(E_{\text{vanishing}}), 0)]$
end if
if $\exists e \in E_{\text{vanishing}} : e \notin c$ then
  return $e$ appended to $c$
end if
while length($c$) $> 0$ do
  $n \leftarrow$ number of last point in $c$
  $n \leftarrow n + 1$
  if $n < r$ then
    return $c$ with the new $n$ as number of the last point
  end if
  $e \leftarrow$ last edge in $c$
  if $\exists t \in E_{\text{vanishing}}$ coming after $e$ then
    return $c$ with the last tuple replace by $(t, 0)$
  end if
  remove the last element of $c$
end while
return null
```

Having to test all possible curves, this algorithm has a bad performance. However, taking just a small number of equidistant points on each edge, can give fairly good results in an acceptable amount of time.

Because of the generation of the curves, they can be directly drawn. Although, the segments between two edges are a priori straight lines. This leads to angular curves. B-splines provide a smoother way to draw the curves.

Only straight lines

Defining the quality of a curve as a straight line, cutting the most edges, performs very well on even cluster surfaces. But on interlaced surfaces, the lines become very cluttered. However, due to the limitation to two control points, the algorithm computing the lines can be simplified to two nested iterations over the edges. An example of this heuristic is shown in Figure 4.1.

Angle sum

Another heuristic is to account for the total length of a curve and prefer curves with less bend.

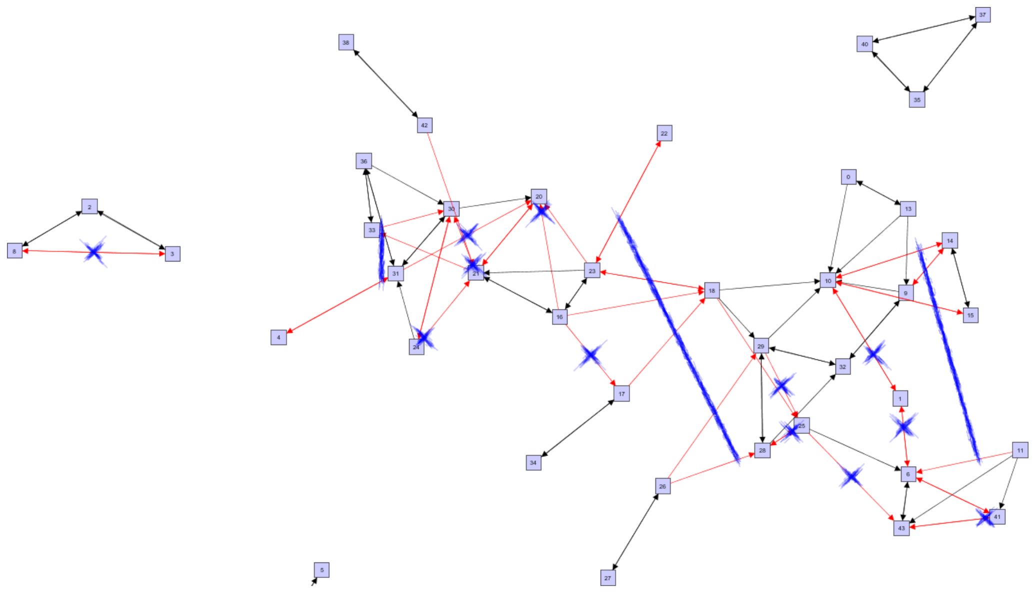


Figure 4.1: A strike out line heuristic, focussing on long straight lines. The edge resolution is 8. Disappearing edges are colored red and strike out lines crossing only one edge are drawn as crosses.

The bend can be defined in different ways. One way would be to take the sum of angles at each control point of the curve. Additionaly this serves as length measurement, too, since only more control points can generate higher sums. The maximum angle sum of a curve is $\pi \cdot n$, where n is the number of control points. Such a curve would be a straight line. However, a curve with only one control point has to have a fixed value. This can lead to the preference of curves cutting one edge over curves cutting two edges. Figure 4.2 shows an example of this heuristic.

Total length and angle deviation

Another way to measure the bend is to take the standard deviation of the angle of the curve segments. Care has to be taken, since the standard deviation can not be computed with the angles directly, due to the fact that the angle 2π is the same as 0. This would give wrong results, if the segments are near to those values. In order to get the correct results, the standard deviation σ has to be computed element by element.

$$\sigma^2 = \frac{1}{n}\sum_{i=0}^{n} \text{angleDifference}(\text{segment}_i, \text{mean\_segment})^2$$

The mean segment is the sum of all normalized segments. The function *angleDifference* takes two vectors and computes a safe angle difference out of their directions.

In order to guarantee long lines, the length of curves has to be compared, too. By binnig subsets of possible angles, the length of curves only needs to be computed if the bend of the curve is in the same bin. In Figure 4.3 an example of this heuristic is shown.

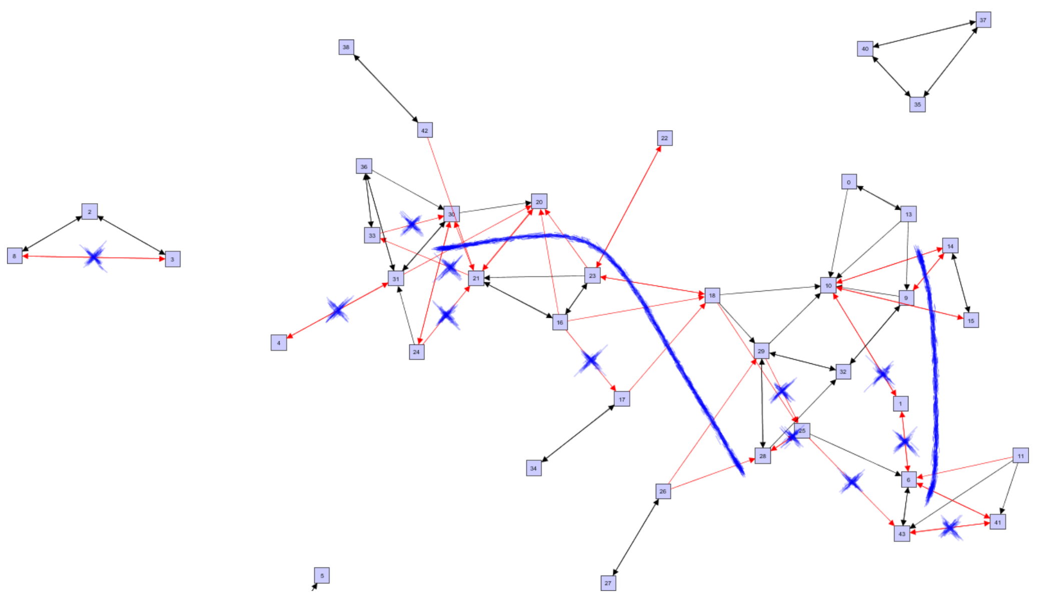


Figure 4.2: A heuristic, favoring curves with many control points and low bend by summing up the angles of the lines at the control points. The edge resolution is 4. Disappearing edges are colored red and strike out lines crossing only one edge are drawn as crosses.

4.2.2 Clustered edge strike out curves

A non-heuristic, analytical way to generate strike out curves is to cluster edges in an appropiate space. This method is not integrative, therefore the task has to be split into cluster generation and representation.

Edge cluster generation

Edges can be described by four dimensional vectors. The meaning of each dimension should be chosen thoroughly. For example, clustering the edges by their start and end positions can be incorrect. Constellations occur, where for example, the start position is nearer to another end position than the respective other start position. The proximity of the two nodes is not encoded correctly, due to the positions coded in different dimensions. This would lead to wrong clustering results.

In order to solve this proximity problem, the Hausdorff-distance [10] can be used to encode the distance between lines. In the case of two lines (L_1 and L_2), the Hausdorff-distance is

$$\delta_H = \max\{\sup_{x\in L_1}\inf_{y\in L_2} d(x,y), \sup_{x\in L_2}\inf_{y\in L_1} d(x,y)\}$$

for a metric d, sup being the supremum, and inf being the infimum. Figure 4.4 shows the results of a clustering based on the Hausdorff distance between edges.

Another choice of features can be the use of one point on the edge (e. g. the midpoint) and the difference of the start and end point, i. e. the direction of the edge. The results of a clustering based on these features are shown in Figure 4.5.

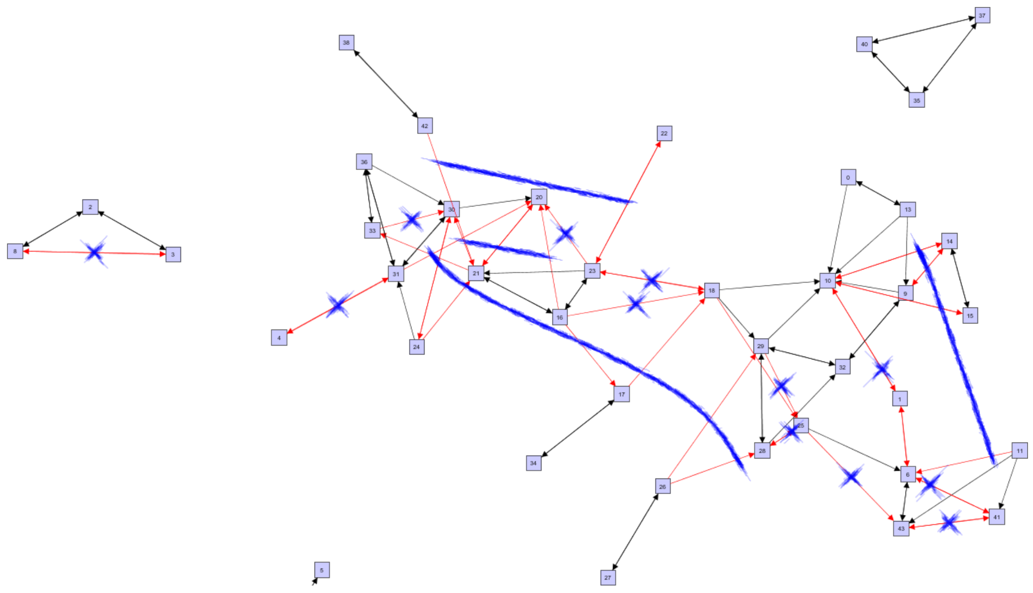


Figure 4.3: Strike out curves generated chosen with as few bend as possible. The bend is computed by calculating the standard deviation of the angle difference of successive curve segments. The edge resolution is 4. Disappearing edges are colored red and strike out lines crossing only one edge are drawn as crosses.

Edge cluster representation

Since elements of a cluster have no inner ordering, a way to draw a single curve through the elements is needed. This is a rather complex task. For the previous shown pictures, the Bubble Sets from Chapter 3.3.1 were used.

Problems with clustering

The clustering of disappearing edges yields some problems.

For example, avoiding edges that must not be clustered, can not be directly integrated in the clustering algorithm. Therefore the generated areas do also strike out some persistent edges (see Figure 4.5 and Figure 4.4). Using line-line visibility, avoiding persistent edges can be encoded in the clustering.

Another problem is, that clusters tend to be compact and circular, which is desireable in the most cases. But this is not desireable on edge clustering, where long, straight clusters are favoured.

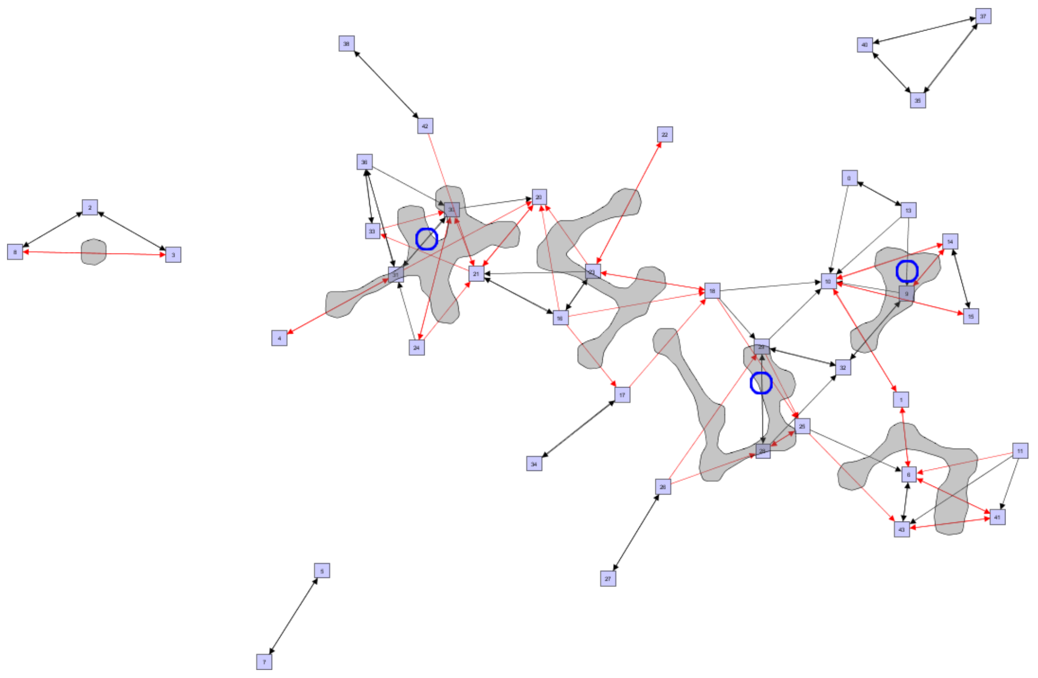


Figure 4.4: Disappearing edges clustered based on the Hausdorff distance. The distance matrix of the Hausdorff distances between the edges was clustered with a hierarchical algorithm. Disappearing edges are drawn in red, and some segments crossing non-disappearing edges are circled in blue.

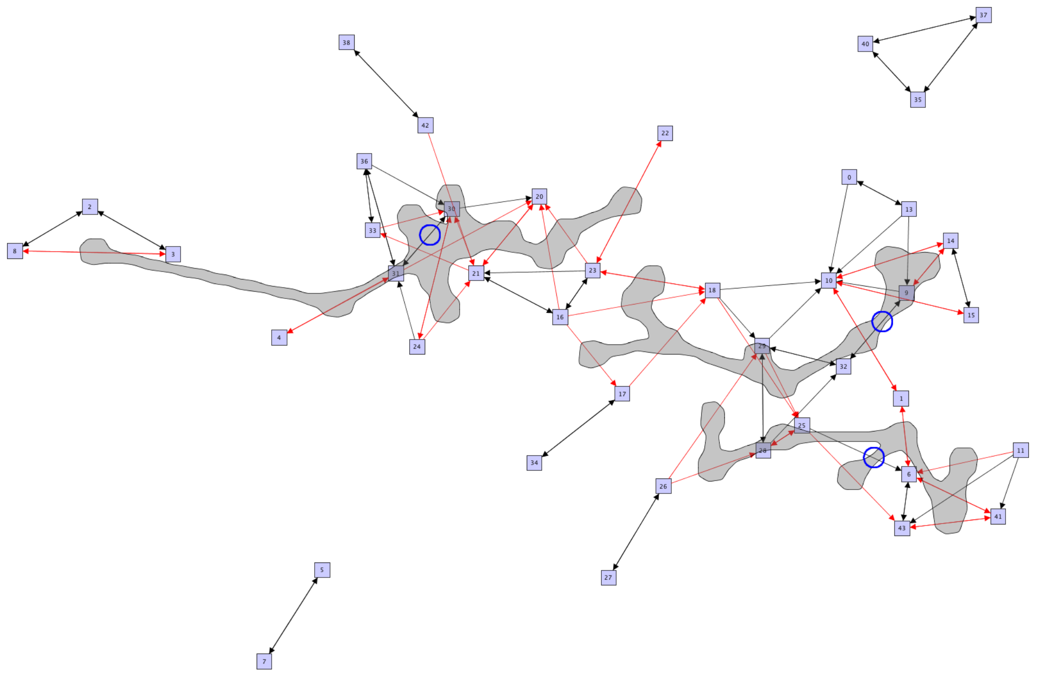


Figure 4.5: Disappearing edges clustered based on the midpoint of the line and its orientation. Those four dimensional vectors were clustered with an xmeans algorithm. Disappearing edges are drawn in red, and some segments crossing non-disappearing edges are circled in blue.

5. Style Variations

Annotations can be drawn in different designs to serve different purposes.

A scribble annotation style could be used to assure that the annotations can be distinguished from the rest of the visual representation of the graph. Also, it reminds the reader of the annotations that they are merely a hint of the changes of the graph and not an exact representation of all changes in it.

For technical applications of graph annotations, a style which blends in with the rest of the visual representation of the graph is the better choice.

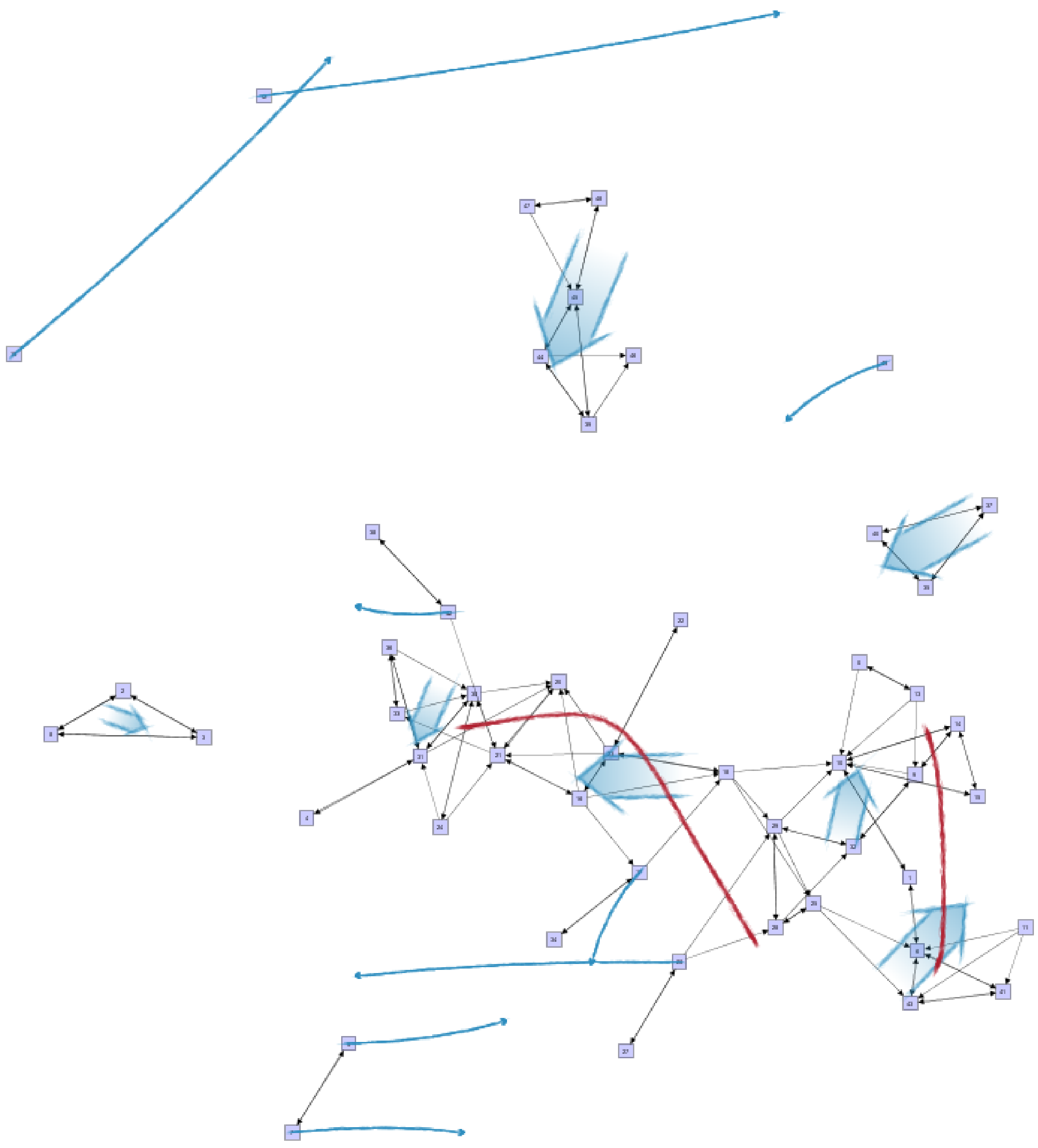


Figure 5.1: Showing annotations for a graph with a sketchy look.

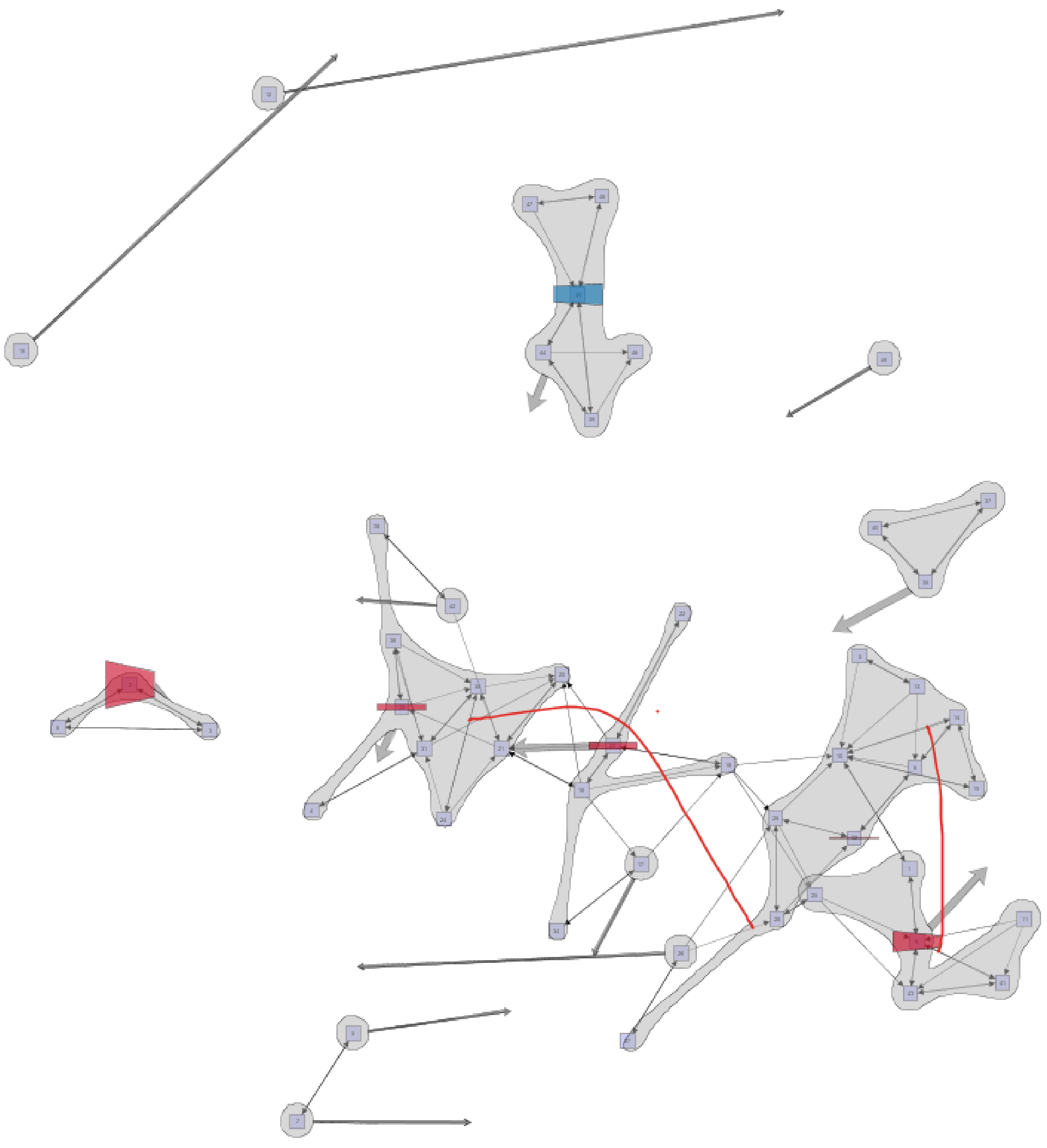


Figure 5.2: Showing annotations for a graph with a technical look.

6. Discussion & Future Work

Changes in evolving graphs can be annotated in different ways. In this thesis, some possibilites are given. However, there is nothing that can be called the optimal annotation for all scenarios.

Having a set of different annotations per type gives the possibility to adjust them for varying presentational needs or subjective preferences. Properly used, annotations are a good addition to conventional dynamic graph drawing techniques.

One advantage of annotations is that they can be combined to increase the richness of the information. This has to be done carefully, though, to not generate new clutter. A possible way to set the granularity of an annotation and tune for the optimal ratio of exactness and expressiveness, would be to vary the number of annotations used at once, and, regarding clustering algorithms, choosing a different number of clusters. Also different annotation types for different parts in a graph are thinkable. Providing more templates of annotation styles for a user, can reduce the task of manually choosing feasible annotations.

Algorithmic improvements are conceivable. A better way to cluster nodes of a graph has to be found, respecting both time steps but introducing no heavy distribution of cluster members and overlaps. Within a cluster, a better measurement for the connectivity change is needed, which encodes the importance of a changing edge. As of inter-cluster edge changes, an appropriate and clusterable representation of edges is of interest. Also a way to encode avoidance of other edges, presumable a line-line visibility measure, is required. Another task is to find a way to transform edge clusters into spline curves.

List of Figures

Bibliography

[1] 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.

[2] 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.

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

[4] Ulrik Brandes and Dorothea Wagner. A bayesian paradigma for dynamic graph layout. In *Proceedings of the 5th Symposium on Graph Drawing (GD'97), volume 1353 of Lecture Notes in Computer Science*, pages 236–247. Springer-Verlag, 1997.

[5] William S. Cleveland and Robert McGill. Graphical perception and graphical methods for analyzing scientific data. *Science*, pages 828–833, August 1985.

[6] 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.

[7] A. J. Enright, S. Van Dongen, and C. A. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Research*, 30(7):1575–1584, 2002.

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

[9] R.L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters 1*, pages 132–133, 1972.

[10] Felix Hausdorff. *Grundzüge der Mengenlehre*. Springer, 1914.

[11] Jr. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244, March 1963.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] Alexandru Telea and Jarke J. van Wijk. Simplified representation of vector fields. 1999.

[17] Stijn Van Dongen. Graph Clustering Via a Discrete Uncoupling Process. *SIAM Journal on Matrix Analysis and Applications*, 30(1):121–141, 2008.