Graph Drawing in Motion II

Carsten Friedrich[1]* and Michael E. Houle[2]**

[1] Basser Department of Computer Science
The University of Sydney
Australia
carsten@cs.usyd.edu.au
[2] IBM Research
Tokyo Research Laboratory
Japan
meh@cs.usyd.edu.au

Abstract. Enabling the user of a graph drawing system to preserve the mental map between two different layouts of a graph is a major problem. Whenever a layout in a graph drawing system is modified, the mental map of the user must be preserved. One way in which the user can be helped in understanding a change of layout is through animation of the change. In this paper, we present clustering-based strategies for identifying groups of nodes sharing a common, simple motion from initial layout to final layout. Transformation of these groups is then handled separately in order to generate a smooth animation.

1 Introduction

In many applications where graphs are used to convey relational information, user and application actions can result in drastic changes in structure and layout. Preserving the so-called 'mental map' during these changes has been identified as crucial to the usability of a system [2]. There are two possible approaches to maintaining the user's mental map: either to develop graph drawing algorithms that seek to minimize change [3], or to use visual or other cues to help support the mental map during the change. Perhaps the most natural and effective way to communicate substantial changes is by means of *animation*; that is, a smooth transition from the old layout to the new layout.

Most existing animation techniques [5,9,10] move the nodes on a straight line from their initial positions to their final positions. In [6,7], Friedrich and Eades showed that in many cases this technique yields poor results. As an alternative, they also provided a method which, as far as possible, seeks to interpret the overall motion of the node set as an affine linear transformation in $\mathbb{R}^2$. Linear regression techniques are used to break down the overall motion into components

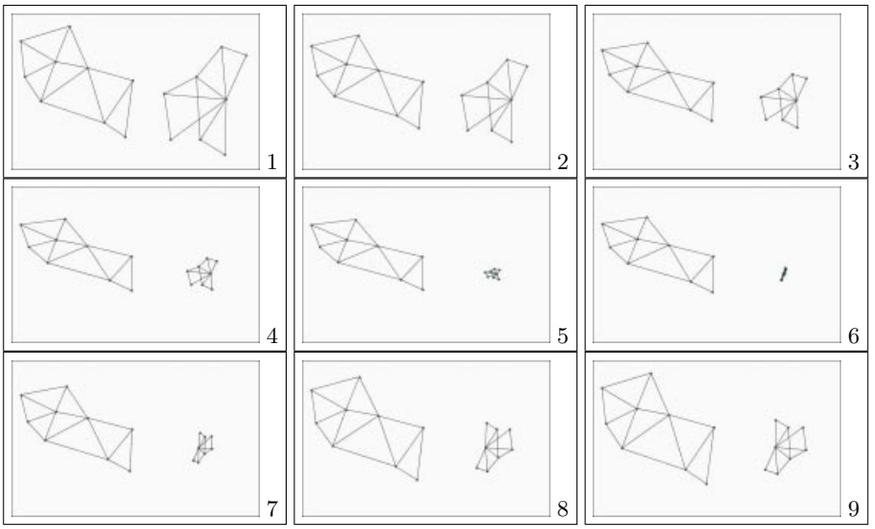
* Carsten Friedrich's research partially supported by the Australian Defence Science and Technology Organization (DSTO).

** Michael Houle is on leave from the Basser Department of Computer Science, University of Sydney.

P. Mutzel, M. Jünger, and S. Leipert (Eds.): GD 2001, LNCS 2265, pp. 220–231, 2002.

due to *scaling*, *rotation*, *shear*, *flip*, and *translation*. These components are then interpolated in parallel and recombined to generate frames for the animation.

Although this method works well when the motion of the nodes is more or less uniform, it performs poorly when several subgraphs have very different underlying motions, as is generally the case when only part of a graph layout is updated. The method attempts to compute an average over all motions of individual nodes, and as a result can decide upon a motion that is not well suited to all parts of the graph. Figure 1 on page 221 shows an example where computing the best overall affine linear transformation produces a poor result[1].

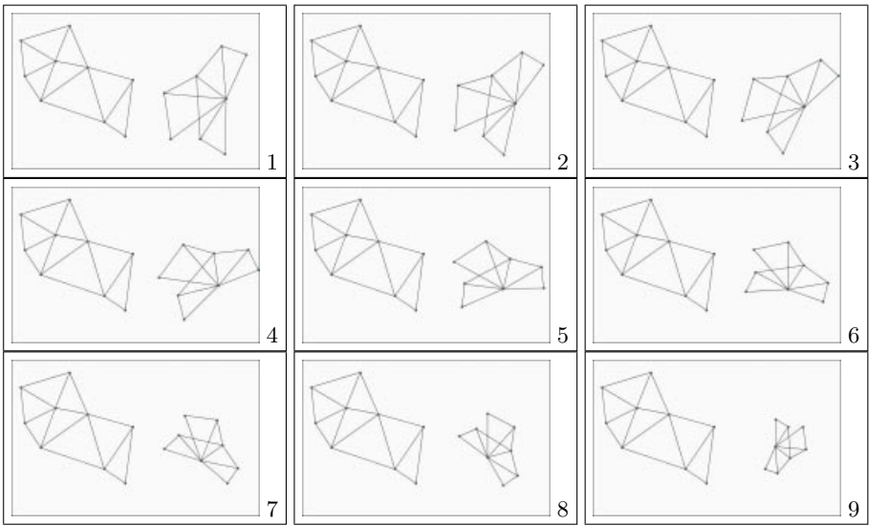


http://www.cs.usyd.edu.au/ carsten/gd01/a.mpg

Fig. 1. Example of a bad animation, starting at Frame 1 and ending at Frame 9. Although only the right-hand component of the graph should move, the calculation and application of an average movement causes distortion in the left-hand component.

In this paper, we use the approach of [7] as the basis of several clustering heuristics, which seek to identify subgraphs which share a similar, structured motion. Applying different transformations to the subgraphs identified allows for the partial update of graph layouts. Figure 2 shows the result of applying the method introduced in Section 3 to the situation of Figure 1.

[1] All examples, in the form of mpeg videos, and a free MPEG player for Windows-NT/95/98 (as well as references to free players for Unix) are available from http://www.cs.usyd.edu.au/ carsten/gd01/.



http://www.cs.usyd.edu.au/ carsten/gd01/b.mpg

Fig. 2. Example of a good animation. Different motions are computed for each of the components of the graph. The result is an animation where each subgraph moves in an individual and much more intuitive way.

2 Determining Candidate Transformations

For each node v there is an infinite number of affine linear transformations that can take it from its initial position to its final position. If the transformations applied to individual nodes were chosen arbitrarily and independently, the resulting animation would appear chaotic. In order to effect a smooth animation, it stands to reason that nodes should be grouped together under common transformations whenever possible.

Using this principle, we can restrict the candidate transformations for v to those that v might share with other nodes. In general, three nodes (associated with three non-collinear initial locations and three non-collinear final locations) are required to uniquely determine an affine linear transformation in the two-dimensional plane. Naturally, if four or more nodes were to determine a common affine transformation, this transformation would also be determined by at least one of its subsets of cardinality three. This suggests that the candidate transformations for v could be limited to those generated by triples of nodes that include v. However, the total number of transformations generated would be cubic in the number of nodes. For the purposes of real-time animation, this candidate set would still be far too large to be investigated explicitly.

To avoid paying the cubic cost of generating all candidate transformations of node triples, practical heuristics are needed. In this paper, we use clustering techniques to identify subsets of nodes sharing similar transformations.

3 *k*-Means

The first clustering method we use is based on the well-known k-means hill-climbing heuristic [12]. The general k-means heuristic for point sets begins with an arbitrary partition $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ of the data set into k groups. It then attempts an iterative improvement of the partition, as follows:

- a representative point is computed for each group P_i;
- each element of the data set is assigned to the representative that best suits it, generating a new partition $\mathcal{P}' = \{P'_1, P'_2, \ldots, P'_k\}$.

The process is repeated until an iteration yields no improvement, according to some measure of goodness of a partition.

It is easy to see that this procedure must eventually converge. The great popularity of the k-means method is due to its simplicity and speed. Typically, the number of iterations required is constant, leading to an observed linear-time complexity.

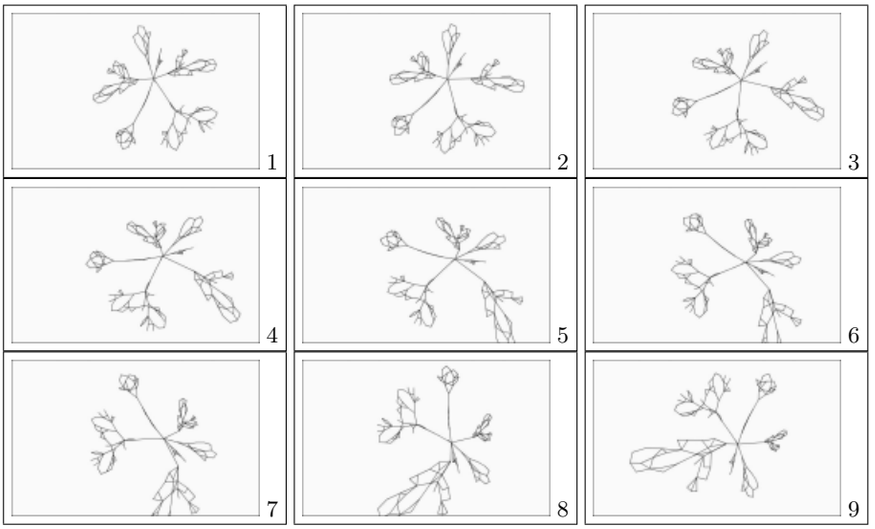
Our adaptation of k-means computes an affine transformation of each group of nodes as the representative of the group, using the linear regression techniques outlined in [7]. In the second step of each iteration, the redistribution of nodes is accomplished by assigning each node to the representative transformation that brings the node closest to its final destination. Improvement can be evaluated according to such measures as the sum of these distances to final destinations, or of squares of these distances. Iteration continues until no further improvement is made.

3.1 Experiments

The following animations show examples of applying the k-means method, where the initial partitions are generated randomly. For all examples, k was set to 10, even though the number of different transformations involved was significantly less. This choice was motivated by two observations. First, as the short-term memory of most users is believed to be able to accommodate roughly 7 items [13], we conclude that computing more than 10 very distinct transformations in one animation would not likely lead to a better preservation of the user's mental map. Second, subclusters derived from a well-associated larger cluster will be associated with very similar transformations, provided that each subcluster contains at least three linearly-independent nodes. This indicates that the method will likely tolerate a choice of k somewhat larger than the minimum.

Figure 3 shows an example where our algorithm successfully identifies three different motions. In addition to an overall rotation of the graph by approximately 180 degrees, two subgraphs also expand in size. The graph is taken from a dataflow analysis application.

Figure 4 refers to the same initial layout as in Figure 3. This time, the animation contains two different motions, a rotation of roughly 180 degrees plus a flip of one connected subgraph. Although the motions are still easy to follow as the animation proceeds, some users may prefer a uniform rotation by the entire graph followed by a flip of a subgraph. As yet, the method does not identify



http://www.cs.usyd.edu.au/ carsten/gd01/c.mpg

Fig. 3. Example of an animation using k-means clustering. The algorithm successfully identifies and displays three separate motions.

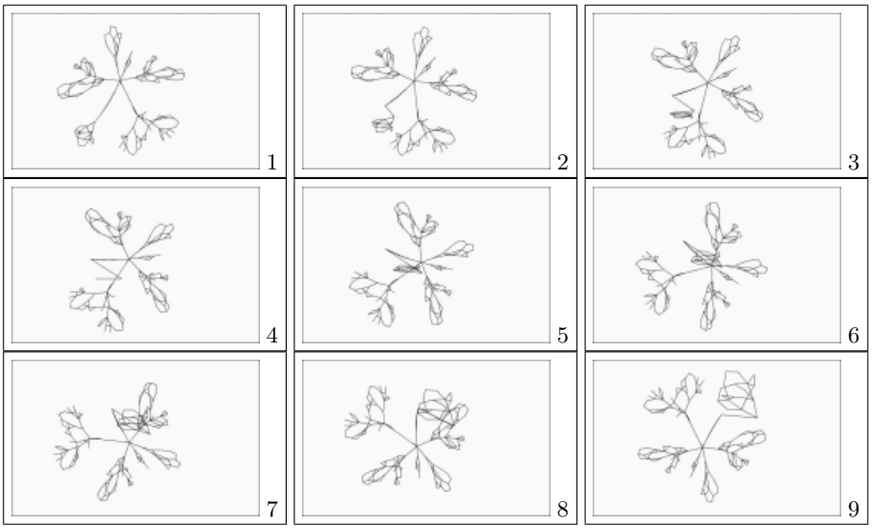
cases in which animation of subgraphs should be performed in sequence rather than in parallel.

Figure 5 shows an animation on an artificially-generated graph with features chosen in an attempt to confound the method. In particular, the wide variation in inter-node distances gives rise to the possibility of an amplification of error, if a transformation generated by collections of nodes that are close to their final destinations happens to be inappropriately applied to nodes that are far from their final destinations. The animation shows a rotation of the complete graph by roughly 180 degrees, plus a scaling of roughly half the graph. Our algorithm successfully identifies both movements.

The initial layout of Figure 6 is identical to that of Figure 5. However, the positions of the nodes of the final layout are those of Figure 5 with a small amount of random scattering. In this case, no ideal transformation exists, and the introduced noise increases the risk of inappropriate transformation of individual nodes. The animation shows that even in this case, the algorithm successfully discovers the main motions. For some nodes which lie very close together, the algorithm produces small additional motions that are not correct.

3.2 Advantages and Disadvantages

Using k-means to compute the animation in many cases successfully identifies different types of transformations, even in the presence of 'local' noise. Convergence is very fast, and the animations may be viewed in real time.



http://www.cs.usyd.edu.au/ carsten/gd01/f.mpg

Fig. 4. Example of an animation using k-means clustering. The algorithm successfully identifies a rotation and a flip.

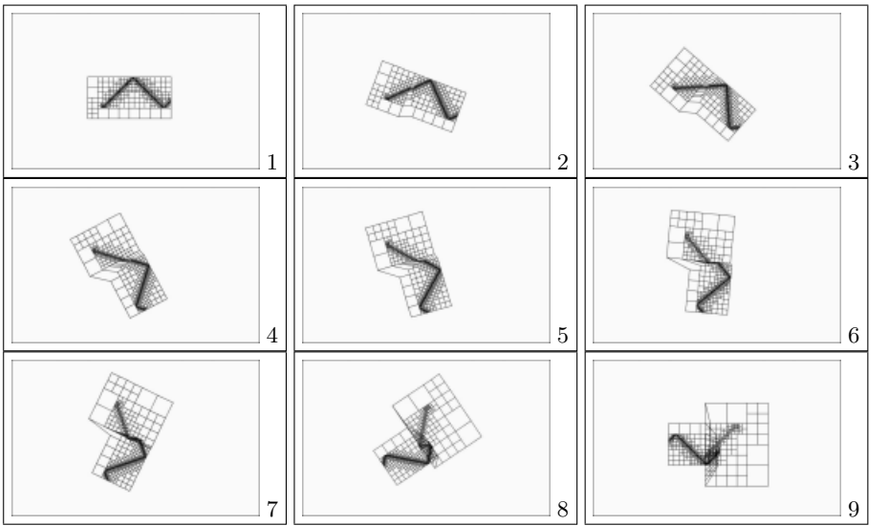
However, the general k-means method is known to be very sensitive to the choice of partition with which it is initialized [11, page 277]. Its variants, including our method, are not well-equipped to discover or escape from poor clusterings that are nevertheless locally optimal. Much effort has been given to the sensitivity of k-means variants to their initial partitions [1,14,15]. In general settings, failure of the k-means heuristic is more likely when clusters are difficult to distinguish, or when a cluster is split among many groups of the initial partition.

In the context of animation, when clusters are difficult to distinguish, their associated transformations are necessarily similar. Whether two similar clusters are declared to be separate or the same, the resulting animations will not differ greatly.

However, the initial partitioning of a single cluster among many groups is a potentially much more serious problem. This has the effect of a subgraph transformation not being applied in the animation. In our experimentation, we observed a tendency of the method to underutilize rather than overutilize transformations.

4 Distance-Based Clustering

Although popular due to its speed, k-means is not the only clustering strategy that can be applied to graph animation. In various spatial settings, many clustering strategies have been proposed. The criteria they seek to optimize almost



http://www.cs.usyd.edu.au/ carsten/gd01/d.mpg

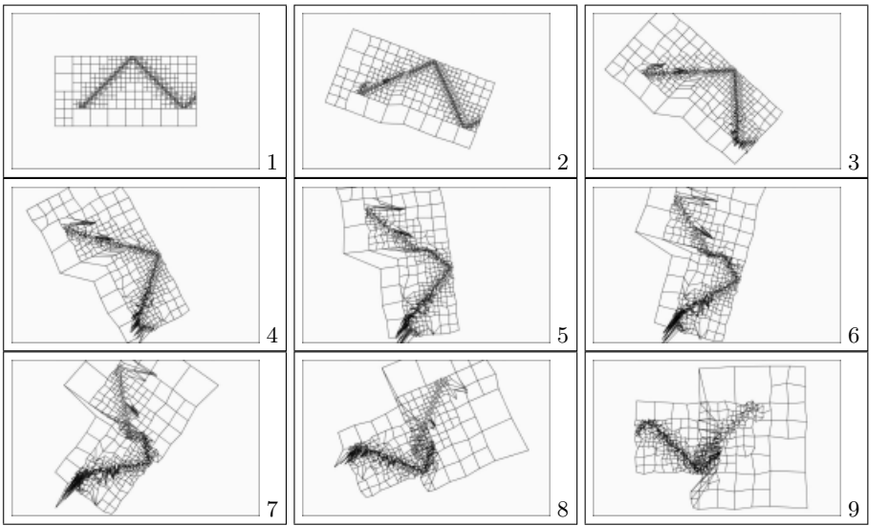
Fig. 5. Example of an animation using k-means clustering. The algorithm successfully identifies a rotation and a scaling.

invariably make use of some notion of distance, usually based on the Euclidean metric, as it captures the essence of spatial autocorrelation and spatial association [8]. Bottom-up approaches, in which clusters are formed by agglomeration of items that are 'close' together, are in accordance with the view that in spatial application areas, nearby items have more influence upon each other. The same can be argued for the graph layout setting, as one would expect a subgraph transformation to apply to nodes appearing in a contiguous region of the layout.

In the context of animation, the stopping conditions proposed for the k-means animation method can serve as the clustering criterion for more general methods. However, as was mentioned earlier, the problem remains of how to limit the number of triples of nodes examined when generating candidate affine transformations. One solution is to take advantage of the tendency for nodes sharing a common transformation to lie close to one another in the initial and final layouts. By restricting the examination of node triples to those lying close in either the initial or final layouts, a large reduction of the number of triples may be achieved.

4.1 Delaunay Triangulation

A classical data structure from computational geometry, the *Delaunay triangulation*, ideally captures proximity relationships among points in the plane, in a very compact form. The Delaunay triangulation $\mathcal{D}(S)$ of S is a planar graph embedding defined as follows: the nodes of $\mathcal{D}(S)$ consist of the data points of



http://www.cs.usyd.edu.au/ carsten/gd01/e.mpg

Fig. 6. Example of an animation using k-means clustering. In addition to a pure rotation and scaling, the final node positions are randomly perturbed by a small offset.

S, and two nodes s_i, s_j are joined by an edge if and only if there exists a circle passing through s_i, s_j having empty interior. The Delaunay triangulation is the graph dual of the well-known Voronoi diagram of S. Some of the interesting features of Delaunay triangulations are listed below; more details can be found in virtually any textbook in computational geometry (for example, [4]).

1. If s_i is the nearest neighbor of s_j from among the data points of S, then (s_i, s_j) is an edge in $\mathcal{D}(S)$.
2. The number of edges in $\mathcal{D}(S)$ is at most $3n-6$, and thus the average number of neighbors of a site s_i in $\mathcal{D}(S)$ is less than 6.
3. The Delaunay triangulation is the most well-proportioned over all triangulations of S, in that the size of the minimum angle over all its triangles is the maximum possible.
4. The triangulation $\mathcal{D}(S)$ can be robustly computed in $O(n \log n)$ time.
5. The minimum spanning tree is a subgraph of the Delaunay triangulation, and in fact, a single-linkage clustering (or dendrogram) can be found in $O(n \log n)$ time from $\mathcal{D}(S)$.

Figure 7 shows a graph (1) and its Delaunay triangulation (2).

For each triangle of the Delaunay triangulation of the initial layout of nodes, we can compute in constant time the affine transformation between the nodes of the triangle to their target positions, by solving a set of six linear equations with six variables. We now can cluster these transformations using clustering techniques suited for the use of generic spatial distance measures.

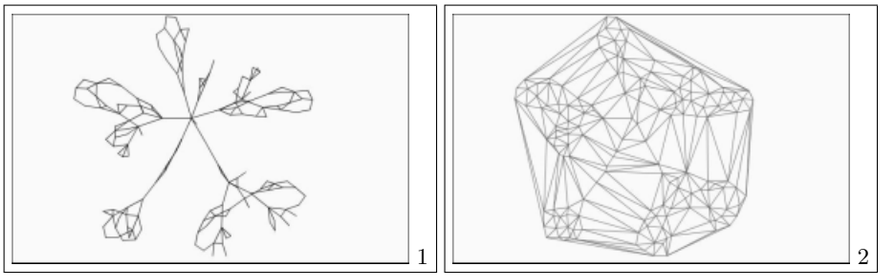


Fig. 7. Example of a graph (1) and its Delaunay Triangulation (2).

4.2 Edge Elimination

An intuitive way to use the transformations of the triangles to compute a clustering is to merge triangles for which the difference of their transformations is smaller than a given threshold value. The elimination of edges results in a small number of large connected regions, the vertices of which can be interpreted as a cluster set. The clusters determined using this method can be animated directly or used to seed other clustering methods, such as k-means, that are sensitive to the quality of their initial partitions.

Figure 8 shows a typical example of the application of the edge elimination method. Figure 8 (1) and (2) show the initial and final drawing of the graph. One part of the graph has been rotated, another part has been scaled and the remaining nodes remained at their initial positions. The graph (3) shows the number of edges (y-axis) separating triangles with differences in their transformations smaller than the threshold (x-axis). Figure 8 (4) shows the clustering obtained by setting the threshold to the point marked by the circle in the graph (3). The degree of darkness of a triangle indicates the number of neighbors with which it has been merged. The distance function is based on the values obtained by decomposing the computed transformation into rotation, scaling, translation, skew, and flip [7]. We can observe that the clusters separate very clearly, and it it easy to produce an appropriate animation from this partition.

Unfortunately, when a significant amount of noise is introduced, in the form of random displacements of nodes, the quality of the result drops sharply. Figure 9 shows the same example with a random displacement of the nodes in the final drawing. Figure 9 (1) and (2) again show the initial and final drawing of the graph, (3) the threshold graph and (4) a clustering obtained by setting the threshold to the point marked in (3). We can see that it is much harder to determine a suitable threshold value in this case. It is however still possible to generate a good seeding for other clustering algorithms by using a small threshold value.

The effect of noise on the quality of the clustering can be explained in terms of the nature of the Delaunay triangulation and of clustering itself. The assumption on which the use of Delaunay clustering was based is that nodes which lie close to each other are more likely to share the same motion than nodes which

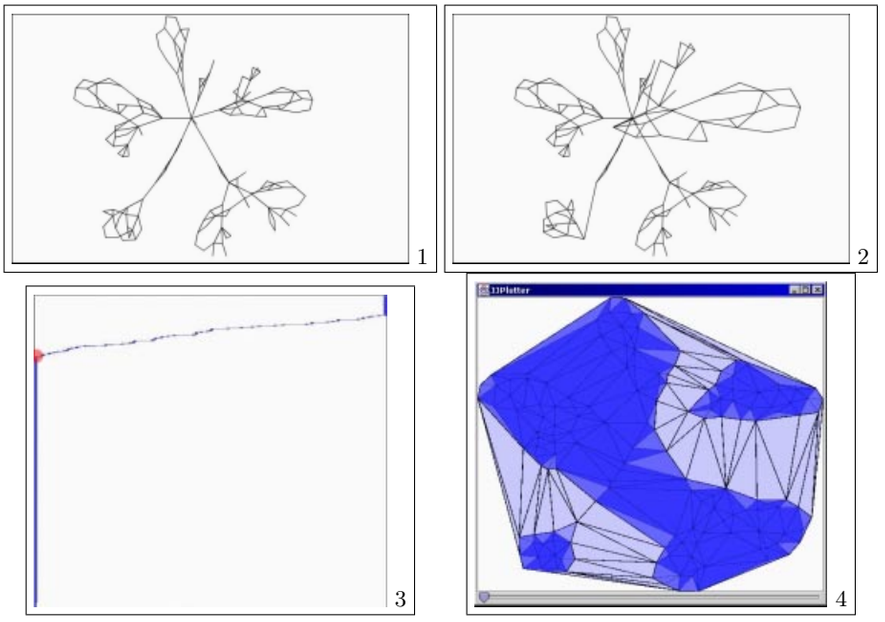


Fig. 8. Example of a clustering obtained by using the edge elimination method. (1) and (2) display the initial and final drawings of the graph. The graph of edges per threshold is displayed in (3), and a clustering obtained by setting the threshold to the point marked in (3) is displayed in (4).

lie further apart. In the Delaunay triangulation, nodes that lie close together form smaller triangles than nodes lying further apart. The displacement of a vertex belonging to a smaller triangle has a greater effect on the transformation generated by the vertices of that triangle, compared to the same displacement of a vertex of a large triangle. Thus displacements tend to reduce the quality of meaningful transformations while having less effect on spurious transformations. This hypothesis agrees with the results of experiments where the graph was designed in such a way that all triangles had approximately the same size. In these cases, useful clusterings were easily generated even in the presence of significant amounts of noise.

5 Conclusion

We have investigated the application of clustering techniques on computing animations of graphs in cases where subgraphs are transformed in different ways. We have seen that in many cases standard techniques such as k-means or edge elimination are able to produce good results. We also discussed the circumstances in which these methods tend to produce poor results. A worthwhile topic for future research would be to apply other clustering techniques to graph animation.

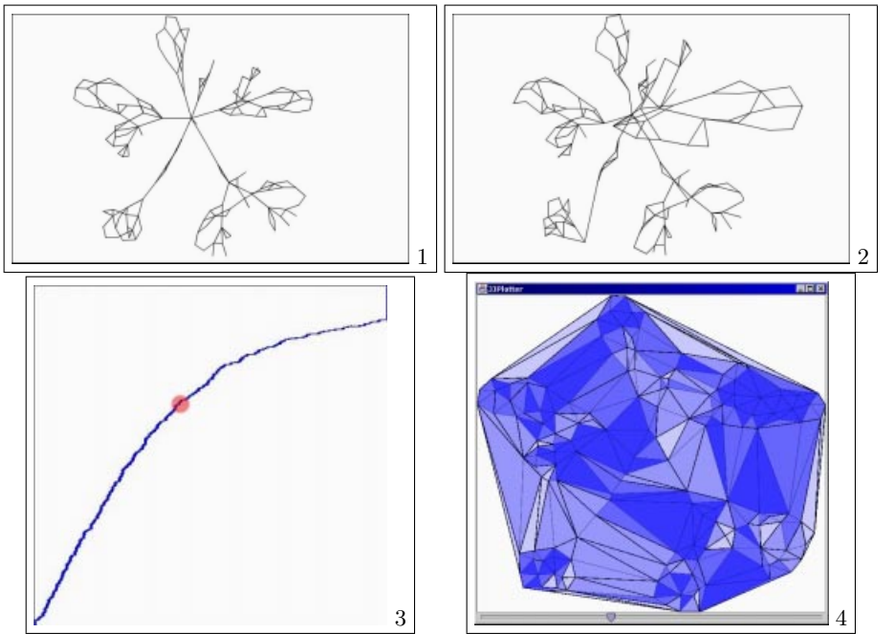


Fig. 9. Example of a clustering obtained by using the edge elimination method. (1) and (2) display the initial and final drawings of the graph. The edges per threshold graph is displayed in (3) and a clustering obtained by setting the threshold to the point marked in (3) is displayed in (4).

References

1. M. S. Aldenderfer and R. K. Blashfield. *Cluster Analysis*. Sage Publications, Beverly Hills, USA, 1984.
2. Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph drawing: algorithms for the visualization of graphs*. Prentice-Hall Inc., 1999.
3. F. Bertault. A force-directed algorithm that preserves edge-crossing properties. *Information Processing Letters*, 74(1–2):7–13, 2000.
4. Mark de Berg, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*, chapter 9, pages 188–200. Springer Verlag, 2nd edition, 1998.
5. C. Friedrich. The ffGraph library. Technical Report 9520, Universität Passau, Dezember 1995.
6. Carsten Friedrich and Peter Eades. The marey graph animation tool demo. In *Proc. of the 8th Internat. Symposium on Graph Drawing (GD'2000)*, pages 396–406, 2000.
7. Carsten Friedrich and Peter Eades. Graph drawing in motion. *Submitted to Journal of Graph Algorithms and Applications*, 2001.
8. R. P. Haining. *Spatial Data Analysis in the Social and Enviromental Sciences*. Cambridge University Press, UK, 1990.

9. http://www.mpi sb.mpg.de/AGD/. *The AGD-Library User Manual Version 1.1.2.* Max-Planck-Institut für Informatik.
10. Mao Lin Huang and Peter Eades. A fully animated interactive system for clustering and navigating huge graphs. In Sue H. Whitesides, editor, *Proc. of the 6th Internat. Symposium on Graph Drawing (GD'98)*, pages 374–383, 1998.
11. L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, NY, USA, 1990.
12. J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. Le Cam, and J. Neyman, editor, *5th Berkley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.
13. G. A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *The Psychological Review*, pages 63:81–97, 1956.
14. U. Fayyad P. S. Bradley and C. Reina. Scaling clustering algorithms to large databases. In R. Agrawal and P. Stolorz, editor, *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 9–15, 1998.
15. C. Reina U. Fayyad and P. S. Bradley. Initialization of iterative refinement clustering algorithms. In R. Agrawal and P. Stolorz, editor, *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 194–198, 1998.