Cluster-based Visualization of Dynamic Graphs

Pascal Held, Julia Hempel, Rudolf Kruse

Otto-von-Guericke University of Magdeburg
Faculty of Computer Science
Department of Knowledge Processing and Language Engineering
Universitätsplatz 2, D-39106 Magdeburg
Tel.: +49 391 67 52700
Fax: +49 391 67 12018
E-Mail: {pheld,kruse}@iws.cs.uni-magdeburg.de,
julia.hempel@st.ovgu.de

Abstract

Graph visualizations are applied for describing relations between objects in many application fields, e.g., in social network analysis and software visualization. Several clustering strategies can be used to identify groups of objects automatically. On the one hand, visualizing these clusters is useful to analyse and evaluate clustering algorithms. On the other hand, cluster visualization allows a fast estimation of similarity between objects and provides orientation in the graph. Because objects, relations and clusters might change over time, dynamic graph drawing received significant interest in the last decades. Several algorithms have been proposed enhancing well-known static layout algorithms. However the dynamic drawing of clusters in graphs is less considered. In this work, we propose three layout algorithms for dynamic clustered graphs. While two approaches are based on enhancing a force-directed layout, the third one uses a divide-and-conquer approach. The approaches are evaluated and compared based on different metrics. The results suggest that the divide-and-conquer approach is best suited for the dynamic drawing of clustered graphs since it separates the clusters well and stabilizes the layout.

1 Introduction

Graphs are important data structures for describing relations between objects. In order to visualize those relationships, graph drawing is applied in many application fields, e.g. in social network analysis and software visualization. Therefor, objects are visualized as nodes and relations as edges between nodes. Moreover, the affiliation of objects to a category or cluster can be visualised using graphs. This can be done e.g., by grouping nodes together or drawing a bounding box around nodes of a cluster. Drawing a clustered graph requires laying out nodes and edges while taking into account the cluster affiliation as well as aesthetic criteria. Especially for a large number of nodes, finding a pleasing layout is time consuming and error-prone [1]. Because of that, automatic layout algorithms were extensively studied [2]. Since relationships might change rapidly over time, e.g., in social networks or computer networks, dynamic graph drawing received significant interest in the last two decades. Several algorithms [3, 4, 5] and specialized visualization techniques [6, 7] have been proposed for dynamic graph drawing. Moreover, empirical studies have been conducted to investigate the issue of dynamic graph comprehension [6, 8, 9].

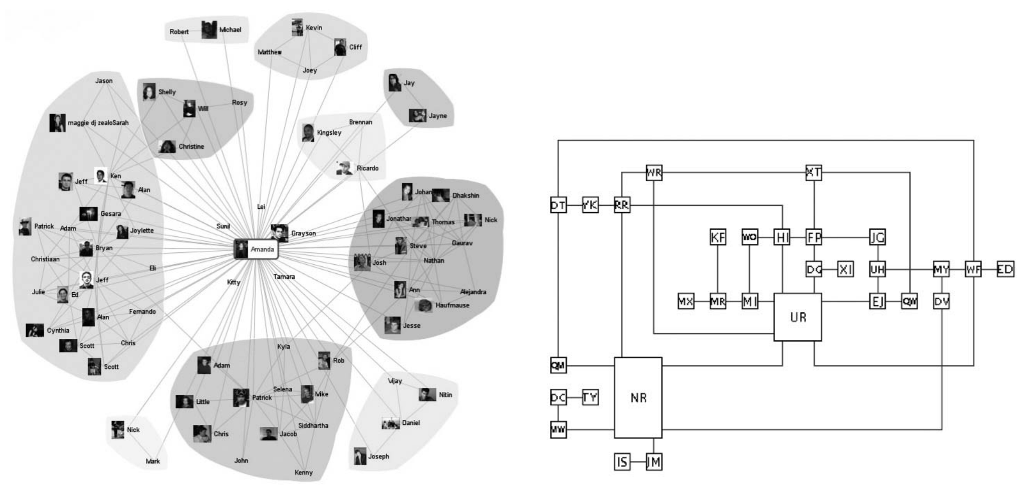
However, algorithms for drawing clustered graphs were less taken into consideration. In this paper, we present three algorithms which aim to group cluster nodes together and evaluate their success for dynamic graphs. In order to provide a frame of reference, we will first discuss requirements on graph layouts and common algorithms. After that, we describe our layout algorithms in detail. Finally, the results of the evaluation are presented and discussed.

2 Layout Requirements

Dynamic graph drawings aim to visualize the evolution of relationships between objects over time. Therefore, both the static layout at a certain time and the evolution of the layout need to support the users' comprehension of the graph. Consequently, the challenge is to compute a new layout that is both aesthetically pleasing and fits well into the sequence of drawings of the evolving graph [5]. Below, quality metrics for static layouts and for dynamic layouts are discussed.

2.1 Static Layout

The problem of static graph drawing has been studied extensively [2]. Different conventions exist which are well known for different domains. For example, straight-line drawings are widely used in social network visualization, while orthogonal drawings are common in circuit schematics and software engineering (see Fig. 1).



(a) (b)

Figure 1: Straight-line drawing (a) [10] and orthogonal drawing (b) [11]

To support users in reading and understanding graph drawings, several aesthetic criteria developed over time. The following criteria are listed by Battista et al.[2].

Crossings: Minimization of the total number of crossings between edges

Area: Minimization of the area of the drawing and the aspect-ratio. Ideally, we would like to obtain small area for any aspect-ratio in order to fit drawings in arbitrarily shaped windows.

Edge Length: Minimization of the sum of the edges, the maximum length of an edge and the variance of the length of the edges

Bends: Minimization of the sum of bends, the maximum number of bends on an edge and the variance of the number of bends on an edge (only for orthogonal drawings)

Angular Resolution: Maximization of the smallest angle between two edges incident on the same vertex

Moreover, the classification of vertices into clusters is desirable in some application fields e.g. to visualize circles of friends in a social network. Therefore, vertices are divided into a set of clusters, as presented in Figure 1 (a). Concerning the visualization of clustered graphs, additional aesthetic criteria are named by Frishman et al. [5]:

- The size of each cluster should be proportional to the number of vertices it contains.

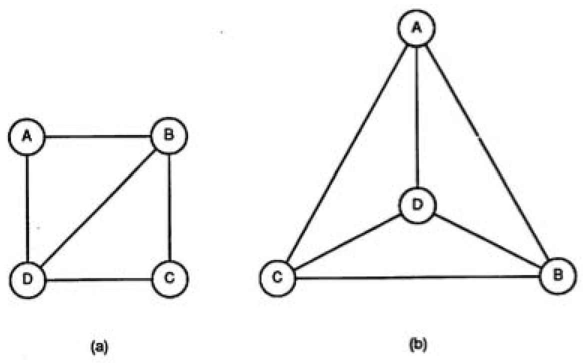


Figure 2: Adding an edge destroys the mental map [13]

- The drawing of each cluster should be compact.
- The overlapping between cluster boundaries should be minimal.

2.2 Dynamic Layout

In interactive applications, several types of modifications, e.g. adding or removing vertices, edges or clusters, change a graph over time. Moreover, the underlying data of a graph might change. Therefore, the graph layout needs to be updated. However, most of the static graph drawing algorithms are not incrementally stable. A small change in the input set may yield unpredictable, instable changes between successive layouts as presented in Figure 2. This might be confusing and annoying for users since they have to spend a lot of time relearning the new graph. To overcome this problem, the users' mental map should be preserved while updating the layout [12, 13, 14]. Commonly, preservation of the mental map is defined as moving as few nodes as possible as little as possible [14, 13]. This shall help users to read and memorize evolving graphs. Moreover, graphical updates should reflect actual changes in the data [14]. That means, adding one edge should result in moving only vertices that are involved in the change.

Additional requirements are named by Frishman et al. [5] for clustered graphs:

- The movement of clusters between successive layouts should be small. Especially, clusters that are not modified should remain in their previous position if possible.

- The change in the size of clusters between successive layouts should be minimal, when the number of vertices in the cluster is similar.
- The movement of vertices inside a cluster should be minimized.

In the following, we will present algorithms for dynamic graph drawing and discuss their strengths and shortcomings based on the presented requirements.

3 Algorithms

Table 1: Overview of widely used dynamic graph drawing algorithms

Author	Offline vs. Online	Graph Type	Static Algorithm	Mental map metric
North et al. [14]	online	directed acyclic graph drawn in a hierarchical manner	Sugiyama's heuristic [15]	node position and order
Brandes et al. [16]	online	undirected graph	generic	node positions
Diehl et al. [17, 18]	offline	undirected graph	generic	node posistions
Erten et al. [4]	offline	weighted graphs (nodes and edges weighted)	force-directed algorithm by Kamada and Kaiwa [19]	node positions
Frishman et al. [5]	online	directed, clustered graph	force-directed algorithm Neato [20]	cluster and node position

In the last section, criteria for dynamic graph layouts were discussed. Commonly, algorithms for dynamic graph layouts are based on static graph drawing algorithms. These static algorithms were augmented in order to preserve the users' mental map. In order to provide a frame of reference for our work, we present static and dynamic layout algorithms especially for clustered graphs.

3.1 Static Layout Algorithms

For the static setting, algorithms are in general well studied. Most common algorithms are force-directed layout algorithms. They are used to create straight-line drawings of undirected graphs by simulating a system of forces and finding a local minimum energy configuration [2]. The force-directed algorithm by Fruchterman and Reingold [21] and the spring-embedder algorithm by Kamada and Kaiwa [19] are widely used for static graph drawing. However, work on clustered graph drawing is less wide spread. An algorithm for straight-line drawing of clustered graphs has been presented in [22]. However, it only applies to planar clustered graphs where every cluster induces a connected planar subgraph. Wang and Miyamoto [1] present a more general algorithm using a divide-and-conquer approach. They first partition the graph into subgraphs. After that, they layout the subgraph using a force-directed layout algorithm and finally they compose the subgraphs together. In [23], an algorithm for drawing clustering hierarchies of a graph using a hierarchical graph drawing algorithm is presented. For a discussion of clustered graph drawing refer to [24].

3.2 Dynamic Layout Algorithms

In Section 2, we discussed requirements for static and dynamic layouts. Unfortunately, these criteria are often contradictory [25]. On the one hand, node and cluster positions might change radically from time-slice to time-slice when optimizing aesthetic criteria. On the other hand, the individual layout might become difficult to understand, when positions are fixed to preserve the mental map. Because of that, finding a suitable trade off is a crucial and also very challenging task in the design of dynamic graph layout algorithms. In general, dynamic algorithms can be grouped in two categories: offline and online algorithms. In the offline scenario, the entire input sequence is known in advance, whereas in the online scenario the sequence is given one graph at a time [9]. An overview of commonly used approaches is presented in Table 1. Below, some of these approaches are discussed in detail.

An early approach for online graph drawing is proposed by North et al. [14]. They developed DynaDAG, a dynamic layout algorithm for hierarchical directed graphs. Their approach is based on the static layout algorithm by Sugiyama [15]. To preserve the users' mental map, they take geometric and topological stability into account. That means, the position and the order of nodes shall be stable between successive layouts. For this purpose, a heuristic is used to move nodes between adjacent ranks, based on median sort.

Brandes et al. [16] introduced a more general approach for graphs. Different static layout algorithms can be used as a baseline for the algorithm, e.g., Eades' spring-embedder [12]. Their approach is based on random field models and Bayesian conditional probabilities. The layouts, which are produced by the static layout algorithm for the respective graph, are updated based on a stochastic estimator. This estimator is composed of the static layout model and an additional stability model. The underlying metric for the stability model (e.g., node movement, relative node movement) is adaptable. Moreover, the trade-off between readability (static quality) and mental map preservation can be changed by adapting the weight of both models.

Erten et al. [4] developed GraphAEL, which is a package to create 2D and 3D graph animations. Their implementation is based on the force-directed algorithm by Kamada and Kaiwa [19]. However, they enhanced the algorithm in order to support both node weights and edge weights. As a result, GraphAEL tries to place heavy nodes well away from each other and to place vertices connected by heavy edges, closer to each other. To preserve the mental map, the algorithm aims to minimize the movement of vertices in the evolving graph. Therefore, the different time-slices are combined to a single graph as shown in Figure 3(a). Between vertices with the same label in adjacent time-slices, edges are created. Because of these edges, attractive forces exist between vertices in different time-slices. Each vertex is attracted towards the vertices associated with it in the adjacent time-slices and consequently, its freedom of movement is limited. As in Brandes' approach [16], the impact of mental map preservation can be easily configured by changing the weight of the inter-time-slice edges. However, no other mental map criteria can be used in GraphAEL since the metric is built into the heuristic for minimizing forces [3].

Frishman et al. [5] developed a dynamic layout algorithm for clustered graphs. Their approach is based on the force-directed layout component Neato which is available in the GraphVis package [20]. In order to compute a new graph layout, Frishman et al. compute the force-directed layout of this graph using the previous layout as initial layout. Concerning mental map preservation, they ranked cluster stability to be more important than vertex stability and vertex stability to be more important than edge stability. To implement these criteria, they added a dummy node to each cluster and connected it to all vertices in the cluster as presented in Figure 3(b). The position of the dummy node is fixed in order to minimize cluster movements. Moreover, spacer nodes are added to the cluster as placeholders for new nodes. If a node is added to the cluster, the closest spacer is

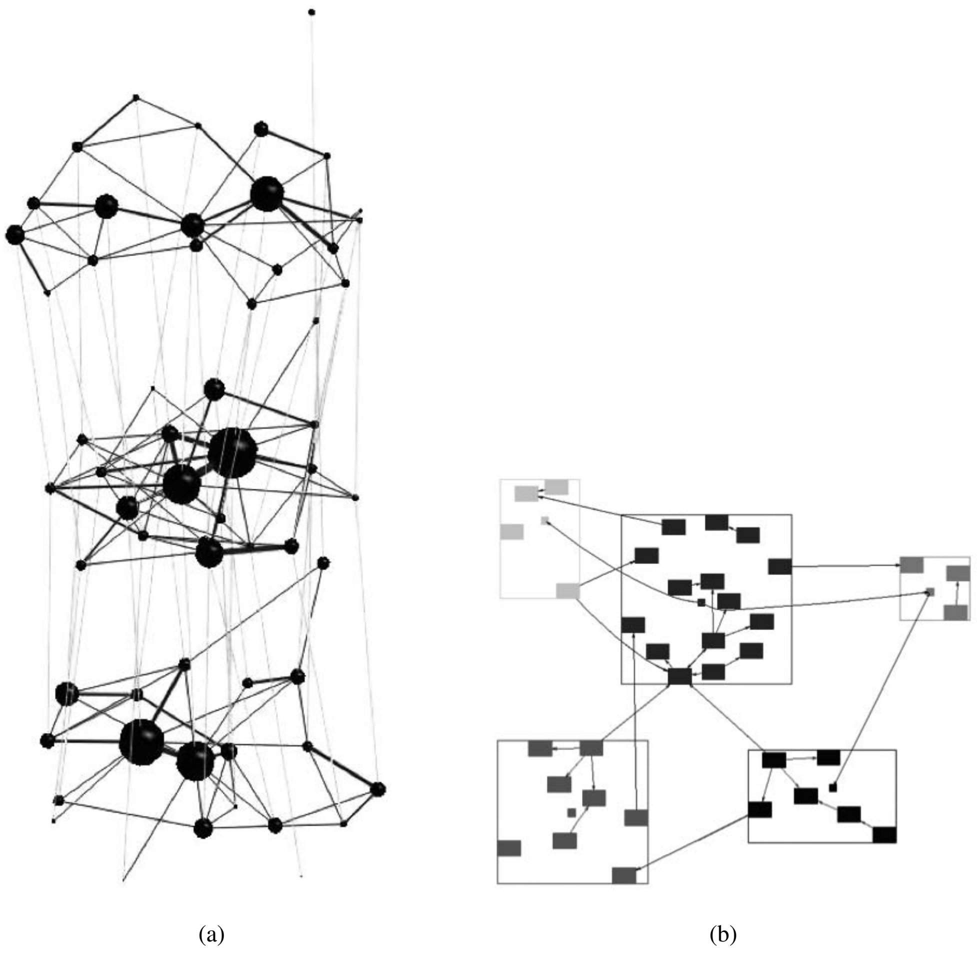


Figure 3: Mental map preservation by attractive forces between (a) nodes in adjacent time-slices [4] and (b) cluster nodes and the cluster center [5]

replaced by this node. Hence, the size of a cluster is maintained with the cost of extra screen space needed for the invisible spacers.

4 Implementation

In the previous section, we presented several algorithms for graph drawing. However, only the algorithm by Frishman et al. [5] considers the visualization of dynamic clustered graphs (to the best of our knowledge). In order to provide a broader comparison and discussion, we implemented three cluster-based graph drawing algorithms. Therefore, we used the open-source framework GraphStream[1]. Moreover, the SpringBox layout was used as a starting point for our implementation. The SpringBox is a force-

[1]http://graphstream-project.org

directed layout which is available in the GraphStream project. It is based on the algorithm by Fruchterman and Reingold [21]. However, it is modified on the attraction. In order to stabilize the evolving layout, the degree of nodes is taken into account. Below, our algorithms are described in detail.

4.1 Force-directed Clustered Layout

Based on the SpringBox layout, this layout algorithms aims to visually separate clusters from each other. Therefore, we manipulate the length of edges using the edge weights. Edges between two nodes within a cluster get a higher edge weight, while edges between two nodes of different clusters get lower weights. Hence, clusters are visually separated since the edges between inter-cluster nodes are longer. This approach potentially provides an easy way to enhance a force-directed layout in order to support clustered graphs. Moreover, the cohesion of the clusters can be easily adapted.

4.2 ClusterNode Layout

The second approach is based on the algorithm by Frishman et al. [5]. For each cluster in the graph, one invisible dummy node is added to the graph. All nodes which belong to the cluster are connected to the dummy node through an edge. After that, the graph layout is computed using the SpringBox algorithm. Based on the dummy node, the nodes of a cluster are arranged closer together because of the additional attractive forces between them. The cohesion of the cluster can be configured using the weight of the edges connecting the dummy node. In contrast to [5], we do not fix the position of the dummy node in order to allow a higher flexibility.

4.3 Divide-and-Conquer

The third algorithm is based on the divide-and-conquer principle. It was proposed by [1] for drawing clustered graphs. In our algorithm, we combined this principle with a force-directed layout. Therefore, we divided the graph in subgraphs. Every subgraph contains the nodes of one cluster. We calculate a force-directed layout for each subgraph using the SpringBox. These layouts are combined using a meta-graph. This graph consists of the clusters in the graph as nodes and the number of nodes within the cluster as node weights. By calculating the layout for this meta-graph, the position of the cluster's centre was determined. Based on this position, the nodes of the subgraphs were placed using their relative position.

5 Evaluation

In order to evaluate and compare the layout algorithms, which we presented in the previous section, we analysed the resulting layouts. The goal of the analysis was to asses both static and dynamic quality criteria.

5.1 Criteria

To assess the quality of the cluster representation, we used the following measures:

Overlap describes the area used by two clusters at the same time. Therefor, the area of a cluster is defined as the convex hull of all nodes. If the same area is used by more than two clusters, the number of overlaps is incremented pairwise. This measure is used to asses how well the clusters are separated.

Minimum Cluster Distance is the minimal distance between pairwise two clusters for the whole graph. If two clusters overlap, this value is 0.

Average Cluster Distance is similar to the Minimum Cluster Distance, but it is the average of all cluster distances. Both Cluster Distance values indicate how good clusters are separated.

Area used by Clusters describes the accumulated area of all clusters. A high value means that the available drawing area is well exploited, while a small value means that the clusters are to compact.

Cluster Crossing Edges describes the number of edges which cross other clusters. Edge crossing should be avoided because they lead to an unclear layout.

Moreover, we analysed the node movements to assess whether the algorithms are well suited for dynamic graph drawings. Therefore, the displacement of each node between adjacent time-slices is calculated and accumulated.

5.2 Graph Generation

We simulated six different graphs, which vary in their number of nodes, connectivity, number of clusters, and number of cluster change events. An overview of the test setting is presented in Table 2. To create a dynamic

Table 2: Overview of the test cases

Test Case	#Nodes	#Cluster	#Noise-Nodes	Intra-Cluster-Connectivity	Inter-Cluster-Connectivity
small	100	3	10	0.2 . . . 0.5	0.01 . . . 0.03
medium	260	5	10	0.2 . . . 0.5	0.01 . . . 0.05
huge	1010	10	10	0.05 . . . 0.3	0.001 . . . 0.05
many edges	260	5	10	0.3 . . . 0.6	0.01 . . . 0.05
less edges	260	5	10	0.075 . . . 0.2	0.0025 . . . 0.01
more dynamic	260	5	10	0.2 . . . 0.5	0.01 . . . 0.05

graph, we simulated 1000 time steps for each graph. In the steps from 1 to 100, the graph is growing due to the creation of nodes and edges. Moreover, the initial cluster assignments are done in this time range. After that, we simulated cluster change events. This means, that groups of nodes are assigned to another cluster. For each of these events, we set the time range, the number of changing nodes, source, and destination of the movement. The exact time step of the reassignment is chosen randomly. During the whole time, edges are inserted and removed randomly to get the expected connectivity. This generation procedure yields in high dynamic graphs.

At `http://goo.gl/KFkOW2` you can download all the test cases as DGS files.

5.3 Results

In the following, we present the results of the different test cases and layout algorithms. In the tables, we present the mean value of the selected measures for the whole time range. The best values is marked **bold** and the worst one *italic*.

5.3.1 Some Impressions

In Figure 4 we showed the same graph with the four different layouts. Some steps before this layout a huge change event has been happen. In

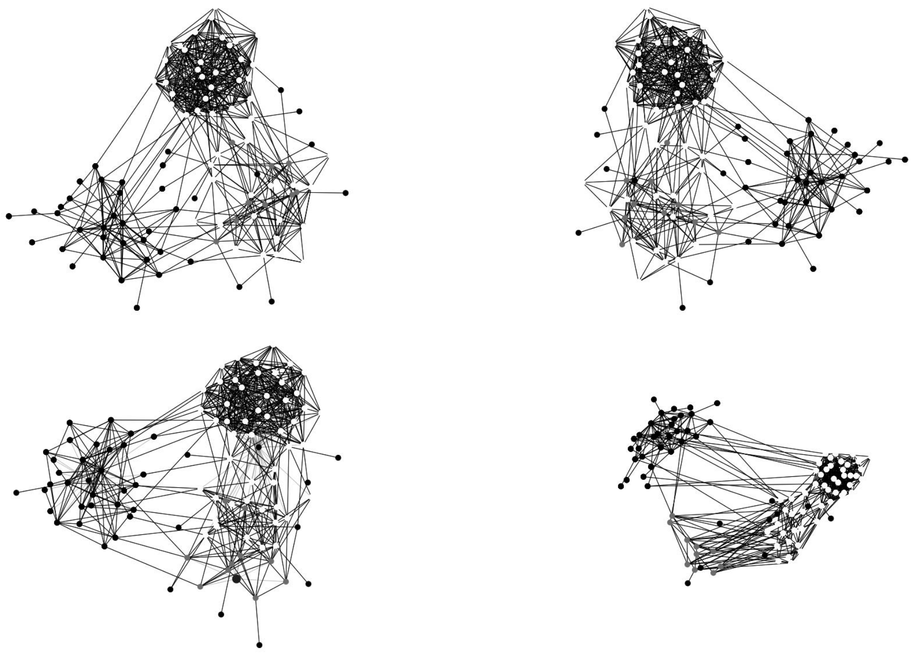


Figure 4: Sample Layouts, top left: SpringBox, top right: Force-directed Clustered Layout, bottom left: ClusterNode Layout, bottom right: Divide-and-Conquer Algorithm

the SpringBox Layout the nodes from the two clusters are totally mixed, the same for the Force-directive Clustered Layout. With the ClusterNode Layout a lot of the nodes are already on the way to the other cluster. In the divide-and-conquer-approach the changing nodes are seperated and next to the target cluster.

5.3.2 Overview Measures

In the following we present the measures for the different test cases and layout algorithms. In the tables we show the mean value of the selected measures for the whole time range. The best values is marked **bold** and the worst one *italic*.

In Table 3, the average node movements are shown. For most test cases, the Divide-and-Conquer algorithm performs best. Only the high connectivity example is a problem for this algorithm. Especially in small or less connected graphs the standard SpringBox algorithm performs worst.

For the measure Overlap, see Table 4, the Divide-and-Conquer algorithm outperforms all other algorithms. Since the algorithm is designed to layout the graph in a way, that every cluster has its own subspace, this result

Table 3: Node Movement

Test Case	SpringBox	Force-directed	ClusterNode	Divide and Conquer
small	*1.512*	1.427	1.451	**0.871**
medium	*6.611*	3.944	3.617	**1.596**
huge	13.858	13.038	*29.908*	**11.890**
many edges	**7.065**	7.544	7.649	*8.588*
less edges	*6.083*	5.895	5.398	**2.418**
more dynamic	*5.475*	5.176	4.744	**2.025**

Table 4: Overlap

Test Case	SpringBox	Force-directed	ClusterNode	Divide and Conquer
small	0.097	*0.098*	0.001	**0.000**
medium	*0.328*	0.190	0.051	**0.000**
huge	1.752	*1.860*	0.393	**0.000**
many edges	*0.946*	0.917	0.281	**0.001**
less edges	*0.359*	0.350	0.017	**0.000**
more dynamic	*0.856*	0.843	0.200	**0.000**

is not surprising. However, the Clustered Node algorithm performs well, too. The SpringBox-based algorithms have a higher number of cluster overlappings, because the cluster has less influence on the node positions.

Table 5: Area used by Clusters

Test Case	SpringBox	Force-directed	ClusterNode	Divide and Conquer
small	**0.568**	0.547	0.331	*0.275*
medium	**0.647**	0.525	0.525	*0.206*
huge	1.231	**1.277**	0.683	*0.162*
many edges	**1.319**	1.300	0.889	*0.278*
less edges	**0.732**	0.722	0.405	*0.147*
more dynamic	**1.131**	1.108	0.700	*0.195*

The area used by the cluster, Table 5, describes how good the drawing space is used. Concerning this measure, the SpringBox algorithms performed best. Most of the space is used by multiple clusters at the same time, so the clusters seems to be mixed. The divide-and-Conquer approach uses less of the available space. The clusters are more compact compared to the other algorithms.

The Minimal Cluster Distance, see Table 6, and the Average Cluster Distance, see Table 7, are measures which show how optical separable the clusters are. In this category, the Divide-and-Conquer approach outperform the other ones. It is by definition able to perfectly separate the clusters. Also the ClusterNode Approach is able to separate the clusters well.

Table 6: Minimum Cluster Distance

Test Case	SpringBox	Force-directed	ClusterNode	Divide and Conquer
small	*0.013*	*0.013*	0.068	**0.120**
medium	*0.000*	*0.000*	0.002	**0.102**
huge	*0.000*	*0.000*	0.003	**0.058**
many edges	*0.000*	*0.000*	0.003	**0.088**
less edges	*0.000*	*0.000*	0.009	**0.089**
more dynamic	*0.000*	*0.000*	0.002	**0.111**

Table 7: Average Cluster Distance

Test Case	SpringBox	Force-directed	ClusterNode	Divide and Conquer
small	0.065	*0.062*	0.145	**0.221**
medium	*0.007*	0.010	0.106	**0.289**
huge	0.011	*0.010*	0.070	**0.349**
many edges	0.013	*0.011*	0.052	**0.231**
less edges	*0.006*	*0.006*	0.094	**0.279**
more dynamic	0.016	*0.015*	0.069	**0.298**

The SpringBox approaches produce layouts with a high number of overlapping clusters.

Table 8: Cluster Crossing Edges

Test Case	SpringBox	Force-directed	ClusterNode	Divide and Conquer
small	59.8	*60.0*	2.8	**0.8**
medium	*3435.5*	2661.1	584.5	**160.3**
huge	*39587.0*	39307.4	16893.4	**1758.8**
many edges	15367.0	*15517.5*	8066.9	**969.9**
less edges	953.7	*957.5*	61.0	**20.0**
more dynamic	*2776.5*	2718.1	1025.7	**114.1**

In Table 8, we show the average crossing edges. A crossing edge is an edge which crosses other clusters then the clusters of the corresponding nodes. Such crossing are confusing for the reader. The smaller cluster areas from the ClusterNode and divide-and-conquer approach are an advantage for this measure. The smaller the area is, the smaller the probability of an crossing edge. The divide-and-conquer algorithm outperforms all the other ones but also the ClusterNode approch produces good results.

6 Discussion

Overall, the results of the evaluation suggest that the divide-and-conquer approach is best suited for drawing dynamic clustered graphs. The clusters are well separated since the layout algorithm guaranties that clusters do not overlap. Moreover, the divide-and-conquer approach supports layout stability. Because the layouts of the subgraphs are computed independently from each other, the resulting node movements are locally restricted. This characteristic might help readers' to preserve their mental map of the graph. However, the good partitioning of the graph comes with the cost of inefficiently used screen space. If the available screen space is a critical issue, the ClusterNode algorithm might provide a good trade off. It separates the clusters better than the SpringBox or the enhanced SpringBox. However, since it allows overlaps, it requires less space.

7 Conclusion and Future Work

In this paper, we investigate algorithms for drawing clusters in dynamic graphs. Based on an extensive literature review. we present three different approaches. We implemented these approaches based on a force-directed layout and evaluated their success using several measures.

In a nutshell, the results suggest that a divide-and-conquer approach is best suited for the dynamic drawing of clustered graphs since it 1) well separates the clusters and 2) stabilizes the layout. Future work involves further evaluation of the algorithm with real world data as well as a user study in order to test the readability of the resulting layouts. Moreover, the performance needs to be improved to allow the usage in interactive applications.

References

[1] Wang, X.; Miyamoto, I.: Generating customized layouts. In: *Graph Drawing*, S. 504–515. Springer. 1996.

[2] Battista, G. D.; Eades, P.; Tamassia, R.; Tollis, I. G.: *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR. 1998.

[3] Görg, C.; Birke, P.; Pohl, M.; Diehl, S.: Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In: *Graph Drawing*, S. 228–238. Springer. 2005.

[4] Erten, C.; Harding, P. J.; Kobourov, S. G.; Wampler, K.; Yee, G.: GraphAEL: Graph animations with evolving layouts. In: *Graph Drawing*, S. 98–110. Springer. 2004.

[5] Frishman, Y.; Tal, A.: Dynamic drawing of clustered graphs. In: *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, S. 191–198. IEEE. 2004.

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

[7] Federico, P.; Aigner, W.; Miksch, S.; Windhager, F.; Zenk, L.: A visual analytics approach to dynamic social networks. In: *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies*, S. 47. ACM. 2011.

[8] Purchase, H. C.; Samra, A.: Extremes are better: Investigating mental map preservation in dynamic graphs. In: *Diagrammatic Representation and Inference*, S. 60–73. Springer. 2008.

[9] Brandes, U.; Mader, M.: A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In: *Graph Drawing*, S. 99–110. Springer. 2012.

[10] Heer, J.; Boyd, D.: Vizster: Visualizing online social networks. In: *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, S. 32–39. IEEE. 2005.

[11] Bridgeman, S. S.; Tamassia, R.: A user study in similarity measures for graph drawing. *J. Graph Algorithms Appl.* 6 (2002) 3, S. 225–254.

[12] Eades, P.; Lai, W.; Misue, K.; Sugiyama, K.: *Preserving the mental map of a diagram*. International Institute for Advanced Study of Social Information Science, Fujitsu Limited. 1991.

[13] Misue, K.; Eades, P.; Lai, W.; Sugiyama, K.: Layout adjustment and the mental map. *Journal of visual languages and computing* 6 (1995) 2, S. 183–210.

[14] North, S. C.: Incremental layout in DynaDAG. In: *Graph Drawing*, S. 409–418. Springer. 1996.

[15] Sugiyama, K.; Tagawa, S.; Toda, M.: Methods for visual understanding of hierarchical system structures. *Systems, Man and Cybernetics, IEEE Transactions on* 11 (1981) 2, S. 109–125.

[16] Brandes, U.; Wagner, D.: A Bayesian paradigm for dynamic graph layout. In: *Graph Drawing*, S. 236–247. Springer. 1997.

[17] Diehl, S.; Görg, C.; Kerren, A.: Preserving the mental map using foresighted layout. In: *Proceedings of the 3rd Joint Eurographics-IEEE TCVG conference on Visualization*, S. 175–184. Eurographics Association. 2001.

[18] Diehl, S.; Görg, C.: Graphs, they are changing. In: *Graph Drawing*, S. 23–31. Springer. 2002.

[19] Kamada, T.; Kawai, S.: An algorithm for drawing general undirected graphs. *Information processing letters* 31 (1989) 1, S. 7–15.

[20] Ellson, J.; Gansner, E.; Koutsofios, L.; North, S. C.; Woodhull, G.: Graphviz open source graph drawing tools. In: *Graph Drawing*, S. 483–484. Springer. 2002.

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

[22] Feng, Q.-W.; Cohen, R. F.; Eades, P.: How to draw a planar clustered graph. In: *Computing and Combinatorics*, S. 21–30. Springer. 1995.

[23] Eades, P.; Feng, Q.-W.; Lin, X.: Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In: *Graph drawing*, S. 113–128. Springer. 1997.

[24] Brockenauer, R.; Cornelsen, S.: Drawing clusters and hierarchies. In: *Drawing graphs*, S. 193–227. Springer. 2001.

[25] Purchase, H. C.; Hoggan, E.; Görg, C.: How important is the mental map? – an empirical investigation of a dynamic graph layout algorithm. In: *Graph drawing*, S. 184–195. Springer. 2007.