Dynamic Drawing of Clustered Graphs

Yaniv Frishman*
Department of Computer Science
Technion - Israel Institute of Technology

Ayellet Tal†
Department of Electrical Engineering
Technion - Israel Institute of Technology

ABSTRACT

This paper presents an algorithm for drawing a sequence of graphs that contain an inherent grouping of their vertex set into clusters. It differs from previous work on dynamic graph drawing in the emphasis that is put on maintaining the clustered structure of the graph during incremental layout. The algorithm works online and allows arbitrary modifications to the graph. It is generic and can be implemented using a wide range of static force-directed graph layout tools. The paper introduces several metrics for measuring layout quality of dynamic clustered graphs. The performance of our algorithm is analyzed using these metrics. The algorithm has been successfully applied to visualizing mobile object software.

CR Categories: I.3.8 [Computer graphics]: Applications; H.4.3 [Information systems applications]: Communications applications—Information browsers H.5.2 [Information interfaces and presentation]: User Interfaces—Graphical user interfaces C.2.4 [Computer-communication networks]: Distributed systems—Distributed applications

Keywords: graph drawing, dynamic layout, mobile objects, software visualization

1 INTRODUCTION

Graphs are an important data structure for describing relationships between objects (e.g., [7, 10]). The need to draw (i.e. *layout*) graphs appears in many applications and as such, is an active area of research in information visualization [16]. In clustered graphs, the vertices are divided between a set of components called *clusters*, which form a partition of the vertex set. In some applications, the graphs are inherently clustered [4]. In other cases, clustering has been successfully used in order to aid in the visualization of graphs [32].

Many applications require the ability of *dynamic graph drawing*, i.e., the ability of modifying the graph [23, 9, 3]. Different types of graph modifications may be performed: adding vertices and clusters, moving vertices between clusters, removing edges, etc. The challenge in dynamic graph drawing is to compute a new layout that is both aesthetically pleasing as it stands and fits well into the sequence of drawings of the evolving graph. The latter criterion has been termed *preserving the mental map* [22] or *dynamic stability* [23]. A short animation sequence showing incremental layouts of clustered graphs computed by our algorithm is shown in Figure 1. In this dynamic scenario, vertices move between clusters and thus the size of clusters change, edges are added, and clusters are added and removed. Yet, the relative locations of the clusters and the vertices are preserved, while allowing changes in the size of clusters when deemed necessary.

*e-mail: frishman@tx.technion.ac.il
†e-mail:ayellet@ee.technion.ac.il

IEEE Symposium on Information Visualization 2004
October 10-12, Austin, Texas, USA
0-7803-8779-1/04/$20.00 ©2004 IEEE

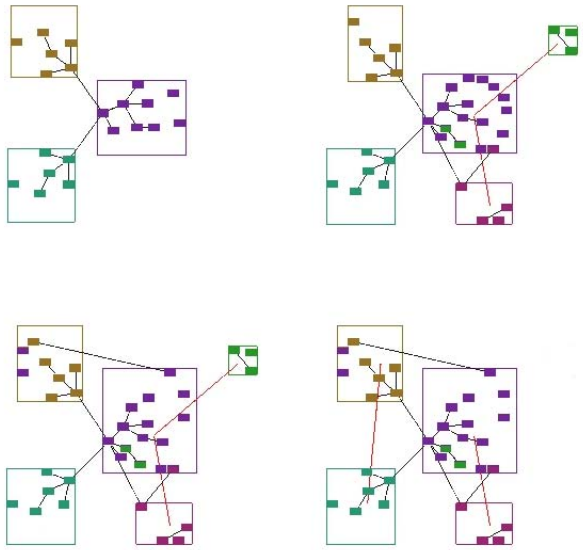


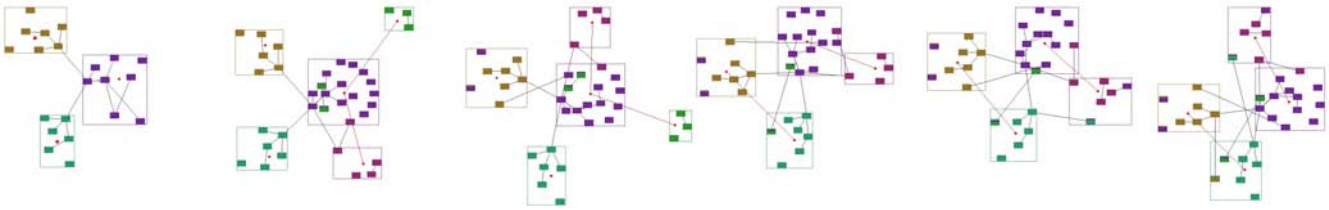
Figure 1: Snapshots from an animation sequence

One field in which clustered graphs arise is software visualization, and in particular, visualization of mobile object frameworks [17, 6, 21]. Such frameworks extend the distributed objects concept [24, 26] in allowing the objects to migrate to remote hosts, along with their state and behavior, while the application is executing (in order to speed up interaction).

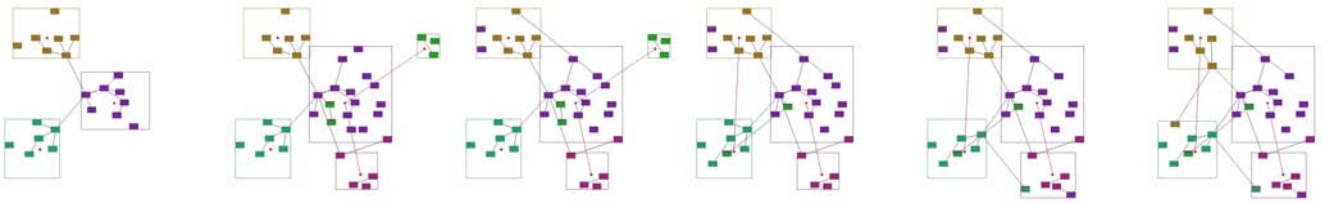
In these frameworks, the notion of a dynamic clustered graph arises quite naturally. Every object is represented by a vertex in the graph. A machine is represented as a cluster that contains the objects currently residing in it. The area occupied by a cluster is used as a visual clue to the user regarding the number of objects located in the machine represented by the cluster. Naturally, the graph being visualized evolves with time, as objects migrate between machines and machines connect and disconnect from the network. Our algorithm has been designed to show these interactions.

The general problem of drawing graphs, e.g., assigning coordinates to graph vertices, edges and other elements, has been extensively studied [27, 20, 8]. One popular technique is force-directed layout, which uses physical analogies in order to converge to an aesthetically pleasing drawing [2, 27, 19]. Drawing non-point vertices using this approach is discussed in [13, 5, 15]. Extending force-directed algorithms for drawing large graphs is discussed in [14, 29].

Work on clustered graph drawing is less widespread. In [30], a divide and conquer approach, in which each cluster is laid out separately and then the clusters are composed to form the graph, is used. In [11], a method of drawing the clustering hierarchies of the graph using different Z coordinates in a 3D view is discussed. See



(a) Force-directed non-incremental layout



(b) Our incremental layout

Figure 2: Incremental vs. non-incremental layout (from left to right)

also [4, 1] for a discussion of clustered and compound graph layout.

Incremental drawing of directed acyclic graphs is discussed in [23], which uses a modification of the Sugiyama algorithm [25] in order to draw ranked digraphs. An algorithm for computing the layout of a sequence of graphs offline is described in [9]. The algorithm is based on using different adjustment strategies in order to compute the new layout. The DA-TU system described in [18] allows navigating and interactively clustering huge graphs. Finally, some commercial graph layout packages such as [28, 31] contain provisions for dynamic layout of graphs. As far as we know, none of the above was designed to handle incremental drawing of clustered graphs.

In this paper we propose a new algorithm for online incremental layout of clustered graphs. The algorithm does not impose restrictions on the structure of the graph. It allows drawing of edges not only between vertices but also between clusters, which is used to convey information to the user. The algorithm provides a means of separating the set of vertices in each cluster to a subset of vertices that stay in the same cluster and a subset of vertices that might move to a different cluster. The layout of the vertices inside the cluster is influenced by this separation.

The major design consideration of our algorithm is preserving the mental map while the graph is being updated. We show that force directed layout techniques [2, 27, 19] can be used as a basic building block. However, they cannot be used as is, as demonstrated in Figure 2(a), where clusters and vertices move considerably between successive drawings. We propose a few enhancements to existing algorithms in order to preserve the mental map, as shown in Figure 2(b), where only small variations in cluster location and size are exhibited. Also note the stability of the vertices inside the clusters as opposed to the non-incremental layout.

A key consideration in designing algorithms is the desirable properties of the results. This paper proposes several criteria for evaluating the quality of dynamic clustered graphs. They include space compactness, minimization of the changes between frames and run-time efficiency. We demonstrate that our algorithm performs well according to these properties. Moreover, we show that this is the case when considering a software visualization application.

The rest of this paper is structured as follows. Section 2 defines the problem. Section 3 describes the algorithm. A software visualization application is presented in Section 4. Finally, Section 5 concludes and discusses future directions.

2 Problem Statement

This section defines clustered graphs and possible graph updates. It also discusses criteria by which the quality and stability of the layout is evaluated.

Definition 2.1 ***Partition:*** *A k-way partition of a set C is a family of subsets $(C_1, C_2, \ldots, C_k)$ such that $\bigcup_{i=1}^{k} C_i = C$ and $C_i \cap C_j = \emptyset$ for $i \neq j$.*

Definition 2.2 ***Clustered Graph:*** *A clustered graph is an ordered quadruple $G = (V, C, E_v, E_c)$, where V is the vertex set, C is a set of clusters which form a partition of the vertex set V, E_v is the set of vertex-vertex edges $E_v \subseteq \{ (v_i, v_j) | i \neq j, v_i, v_j \in V \}$ and E_c is the set of cluster-cluster edges $E_c \subseteq \{ (C_i, C_j) | i \neq j, C_i, C_j \in C \}$.*

Given a series of clustered graphs $G_1, G_2, \ldots, G_n$, the goal of the algorithm is to produce a sequence of layouts $L_1, L_2, \ldots, L_n$, where L_i is a drawing of G_i, such that the sets V_i, C_i, E_{v_i}, E_{c_i} are assigned coordinates. Since the sequence of graphs G_i is not known in advance, the algorithm is an online algorithm. The updates U_i that can be performed between successive elements G_{i-1} and G_i are: Adding or removing vertices, edges or clusters, and modifying the partition of vertices into clusters (i.e. moving vertices between clusters).

A key issue in incremental graph drawing is the stability of the layouts [22, 23]. This is important since a user looking at a graph drawing gradually becomes familiar with the structure of the graph. We propose the following criteria for evaluating the quality of the layout:

1. The movement of clusters between successive drawings should be small. Specifically, clusters that are not modified should remain in their previous position if possible.

2. The change in the size of clusters between successive drawings should be minimal, when the number of vertices in the cluster is similar.

3. Movement of vertices inside a cluster should be minimized.

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

5. The drawing of each cluster C_i should be compact.

6. Overlapping between vertices should be avoided and overlapping between cluster boundaries should be minimal.

Our application to software visualization adds an additional requirement. The vertices in each cluster are divided into two subsets, static objects that remain at the same cluster throughout the animation and movable objects. This should become visually apparent.

Note that there are classical aesthetic criteria such as the number of edge crossings, the total edge length, etc. which we ignore here. However, the underlying static algorithm used addresses these criteria.

3 The Algorithm

Given a sequence of clustered graphs $G_1, G_2, \ldots, G_n$, our goal is to compute a sequence of graph layouts $L_1, L_2, \ldots, L_n$, so as to adhere as much as possible to the criteria discussed in Section 2. A possible approach is to develop an incremental algorithm for drawing clustered graphs from the ground up. A different approach, which we have pursued, is to use an existing non-incremental graph layout algorithm as a basic block, and build the incremental layout capability on top.

Among the different classes of graph drawing algorithms, the force directed algorithm class seems to be the natural choice in our case [2, 27, 19, 7, 10]. Roughly speaking, this approach simulates a system of forces defined on the input graph and outputs a local minimum energy configuration. An edge is simulated by a spring connecting its endpoint vertices. Edge length influences the optimal spring length and edge weight determines its stiffness. The algorithm converges towards a minimum energy position, starting from an initial placement of the vertices. In our case, the previous layout, L_{i-1}, can be used as a starting position for the new layout, L_i. Extending a force directed algorithm to perform a layout of clustered graphs is discussed in Section 3.2.

Our algorithm's requirements from the underlying force-directed static layout algorithm are that there exist ways to assign initial coordinates to vertices, to restrict their movement, to set edge lengths and to add support for drawing clusters. Since little assumptions are made regarding the underlying layout algorithm, a wide variety of existing layout tools can be used. As such, our algorithm can add incremental layout capabilities to most existing packages.

In our implementation we use the GraphViz graph drawing package [12] and its force directed layout component, Neato [13, 19]. Neato avoids overlaps between vertices and allows setting preferred edge lengths and weights. It also allows pinning down vertices. Pinned vertices are not moved while the algorithm converges by moving vertices according to the forces acting on them. However, Neato neither supports clustered graphs nor does it support controlling the repulsive forces between vertices. These deficiencies are addressed by our algorithm, as will be described next.

We adopt the proposition made in [23] that vertex stability is more crucial than edge stability. Specifically, we prefer changing edge lengths rather than moving vertices. Moreover, in our case, cluster stability is more significant than vertex stability. Thus, our algorithm utilizes the following key ideas.

First, dummy vertices and edges are used in order to create a clustered structure. Since clusters are treated as vertices, their motion can be controlled. Second, invisible place-holder vertices are used in order to minimize the movement of clusters and of vertices within clusters. This is done while maintaining compactness and keeping the size of the clusters proportional to the number of vertices they contain. Third, edge length and weight are used as a means of controlling the changes made to the layout. Fourth, to achieve both dynamic stability and distinguish between stable and movable vertices, the set of vertices is partitioned into two sub-sets – stable and movable. The subsets are laid out in a structure that approximates two concentric circles around the center of the cluster. Static objects are placed in the inner circle and movable objects in the outer one.

These ideas are elaborated in this section. After outlining the algorithm, various phases and aspects of the algorithm are discussed in detail, including cluster support, minimization of visual changes, and animations of graph updates.

3.1 Overview

To compute layout L_i, only the last layout, L_{i-1}, and the new graph that needs to be laid out, G_i, are used. This is a fast and simple approach that fits well with the view that incremental layout performs some local changes in the graph. In other words, the previous layout is considered as a good starting point for the new layout, with some adjustments made according to the changes that occurred.

The first step in computing the new layout, described in Section 3.4, is a merge stage, which merges layout L_{i-1} and graph G_i. In the second stage, an actual layout, L_i^1, is computed using a static force directed layout algorithm with the modifications described in Sections 3.2–3.3. In the third stage, the quality of this layout is checked, as described in Section 3.5. If the layout is deemed satisfactory, it is accepted and $L_i = L_i^1$. Otherwise, a second layout attempt is performed, producing layout L_i^2. During this attempt, more freedom is given to the layout algorithm in terms of moving vertices, at the expense of weakening the connection between the old and the new layouts. The better of L_i^1 and L_i^2 is selected as the final drawing L_i. The final stage of the algorithm, described in Section 3.6, animates the change between the drawings L_{i-1} and L_i in a smooth manner. The algorithm is summarized in Figure 3.

```
procedure incremental_drawing ( L_{i-1}, G_i ) {
    G_i^m = merge_graphs ( L_{i-1}, G_i )
    L_i^1 = layout_graph ( G_i^m )
    if ( L_i^1 is good enough )
        L_i = L_i^1
    else {
        L_i^2 = layout_graph ( modify_graph ( L_i^1 ) )
        L_i = better ( L_i^2, L_i^1 )
    }
    animate_change ( L_{i-1}, L_i )
}
```

Figure 3: Algorithm overview in pseudo-code

3.2 Supporting Clusters

Adding an invisible dummy attractor vertex to each cluster, to which all of the vertices in the cluster are connected with invisible edges, is proposed in [4], where repulsive forces are also used, in order to increase cluster separation. One of the approaches discussed is a divide and conquer algorithm, in which the clusters are first laid out separately and then the different layouts are composed together. A hybrid approach that solves the problem of neglecting inter-cluster edges, caused by this algorithm, is discussed in [30].

We follow the approach of adding a dummy vertex to each cluster. However, separation between the clusters and meeting the other requirements described in Section 2, is achieved differently. It is accomplished through proper settings of edge lengths and weights, as described below.

Five kinds of edge lengths are utilized and indicate the expected level of proximity between their adjacent vertices. The shortest length is assigned to the invisible edges connecting static vertices

to the dummy vertex of the cluster they belong to. The edges connecting movable vertices and the dummy vertex are assigned longer lengths. This creates a layout that resembles two concentric circles. The next type of edges is the edges between vertices. If both vertices at the endpoints of the edge are contained in the same cluster, a shorter length is set than if the vertices are in different clusters. This increases the separation between clusters. The last kind of edges are cluster-cluster edges. The length of these edges is variable and depends on the requested proximity between the different clusters, which is determined by the application, e.g., by the amount of interaction between clusters.

Edge weights are also used in our algorithm. Higher edge weights instruct the underlying force-directed algorithm to try harder to generate edges with lengths close to the optimal lengths supplied to the algorithm (as discussed above). Inter-cluster edges are assigned lower weights than intra-cluster edges. This is done in an attempt to give inter-cluster edges less influence on the layout. This is important when vertices move between clusters. In such cases, it is preferable to stretch or shorten the length of the edges somewhat, rather than displace vertices.

In our implementation, the lengths assigned to the edges connecting a static vertex to a dummy vertex, a movable vertex to a dummy vertex, two regular vertices in the same cluster and two regular vertices located in different clusters, are 1, 2, 1.5 and 4 units of length, respectively. The lengths assigned to cluster-cluster edges vary between 5 and 6 units, where the dummy vertices are used as endpoints for cluster-cluster edges. The weight of intra-cluster edges is set to 1 unit and the weight of inter-cluster edges is set to 2.5 units.

3.3 Minimizing Visual Changes

Invisible vertices, called *spacer* vertices, are added to each cluster, in an attempt to reduce the change in clusters' outlines and minimize the movement of clusters between successive layouts.

The spacer vertices are used as place-holders for regular vertices in a cluster. They are connected with invisible edges to the dummy vertex of the cluster to which they belong, like any other vertex in the cluster. When a vertex is removed from a cluster, a spacer vertex is added to the cluster instead of it. The initial location of the spacer vertex is set to be the location of the vertex that left the cluster. This is done in order to keep the size of the cluster constant and in order to reserve space for a new vertex that might be added to the cluster in the future. When a vertex moves (or is added) to a cluster, the spacer vertex that is closest to its previous location is replaced by this new vertex.

However, when adding or removing spacers, the algorithm keeps the number of spacers in a cluster between an upper and a lower fraction of the number of vertices in the cluster. This is done in order to give the algorithm breathing room when modifying clusters. Moreover, the limits are set so as to avoid a case in which a cluster with a very small number of regular, visible vertices occupies a large area due to the many spacer vertices it contains.

When calculating the outline of each cluster, which is often simply the bounding box, the spacer vertices are taken into account as if they were regular visible vertices. Obviously, this minimization of the movements comes at the expense of extra screen space, which is occupied by the spacers.

3.4 Merging Graphs

The first step in performing the incremental layout is merging the new graph to be drawn, G_i, and the previous graph drawing, L_{i-1}. The result of the merge stage is a partially laid out graph, G_i^m, in which some of the vertices are assigned initial coordinates. After merging, the graph G_i^m is laid-out by the static layout algorithm. The quality of the resulting incremental layout depends on the initial conditions computed by the merging algorithm.

Merging is performed in several steps. Unchanged and dummy vertices are assigned initial coordinates from L_{i-1}. Then, clusters to which vertices were both added and removed are handled. The added and removed vertices of a cluster are paired-up, and the initial coordinates of an added vertex is set to the coordinates of a removed vertex.

Then, vertices that were added to a cluster or removed from it, but cannot be paired-up, are handled, as discussed in Section 3.3. Next, the vertices in new clusters, that is clusters that exist in G_i but not in L_{i-1}, are inserted into the graph without initial coordinates, along with new spacer vertices. The number of the latter is set to a constant fraction of the number of vertices in the cluster.

The last stage of merging involves vertex pinning, which restricts vertex movement, allowing it to move only as an indirect result of the movement of an unpinned vertex. We have experimented with several strategies for computing the set of vertices to be pinned. Our conclusion is that pinning all vertices that were assigned coordinates achieves good results in terms of the dynamic stability of the layout. We have also observed that in most cases the resulting layouts are aesthetically pleasing.

3.5 Improving the Layout

After computing the graph layout L_i^1, a *cluster density metric* determines whether the layout is of satisfactory quality. For a cluster C_i, we define

$$density\ metric(C_i) = \frac{area(bounding\ box(C_i))}{number\ of\ vertices(C_i)}.$$

That is, the density metric of a cluster is the ratio between the area of its bounding box and the number of vertices it contains. Higher values imply that the vertices in the cluster are spaced further apart, which is not desirable. For the entire graph G we define

$$density\ metric(G) = max_{C_i \in G}\{density\ metric(C_i)\}.$$

Experience has shown that a correlation exists between high density metric values and overlaps between clusters.

A second layout, L_i^2, is computed if the value of the graph density metric exceeds a threshold. To improve the layout, the restrictions on vertex movement are relaxed. The layout algorithm is re-run with the positions of the vertices in L_i^1 as the initial condition. This time the vertices are not pinned down. This gives the layout algorithm more freedom and allows it to converge to a better result. The new layout L_i^2 still resembles L_i^1 because of the supplied initial condition. The final layout is selected as the layout with the lower density metric between L_i^1 and L_i^2. Clearly, the choice between L_i^1 and L_i^2 demonstrates the tradeoff between preserving the mental map and creating an aesthetically pleasing layout. It should be noted that initial attempts to use more relaxed constraints when computing L_i^2, such as removing some of the assigned vertex coordinates, were counterproductive.

3.6 Display and Animation

We have investigated display in three dimensions, as illustrated in Figure 4, in order to distinguish between vertex types and edge types. Vertex-vertex edges are drawn on the lower plane, while cluster-cluster edges are drawn on the upper plane. In 3D, a cluster is drawn as a semi-transparent pyramid with the cluster's dummy vertex, which is the endpoint of cluster-cluster edges, drawn at the apex of the pyramid. One of our guidelines in creating this visualization is being able to collapse the 3D view into a 2D view in a natural and comprehensible way, as illustrated in Figure 5, which

shows a 2D drawing of the graph from Figure 4. Color is also employed in order to help the user comprehend the image – each cluster has a different color.

The transition between L_{i-1} and L_i is performed using a sequence of intermediate drawings generated by a linear interpolation of the coordinates of vertices, edges and cluster boundaries. (See the attached movie.)

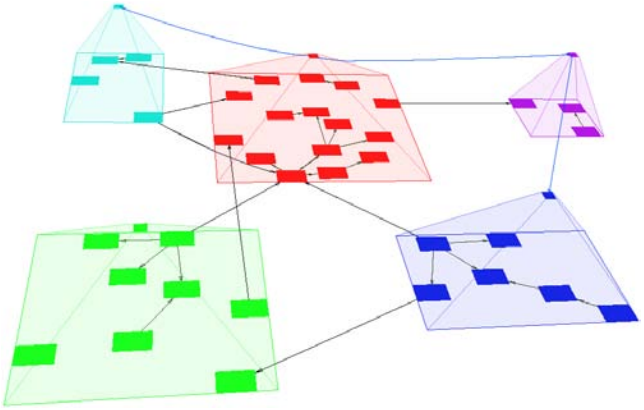


Figure 4: 3D view of a clustered graph

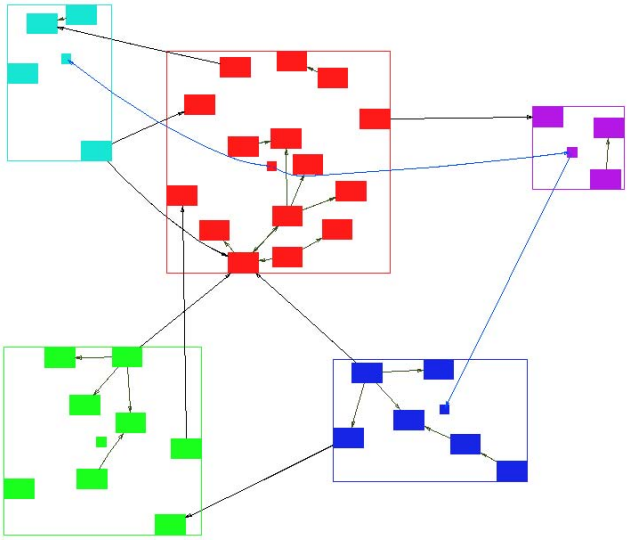


Figure 5: 2D view of a clustered graph

4 Visualizing Mobile Object Software

Our layout algorithm has been used in the visualization of mobile object applications [17, 6, 21]. This framework extends the distributed objects concept, where objects can migrate to remote hosts, along with their state and behavior, during the execution of the application. The visualization should expose the connections, interactions and movements of the objects that are distributed throughout a computer network.

In our visualization, every object is depicted by a vertex. Connections between objects are drawn as vertex–vertex edges. Each machine is represented by a cluster that contains all of the objects currently residing on that machine. The set of cluster–cluster edges is used to display physical connections between machines, as opposed to logical relations that exist between objects.

Our algorithm is demonstrated in Figure 6 as well as in Figures 1-2 and in the accompanying movie. It was tested on several graph sequences. Some of them represent executions of real mobile object applications and others represent simulated data.

To measure the quality of the resulting layouts, we identify several criteria. The first is the density metric discussed in Section 3.5, which is used to measure the compactness of the layout. The second is the sum of displacement of clusters between each pair of successive layouts, which is used to measure the stability of the layout. The third is the percentage of clusters with the same size between successive layouts, which helps to demonstrate the effectiveness of using spacer vertices in minimizing visual changes to the graph.

Figures 7-10 compare the performance of our algorithm to two other algorithms. The first is a non-incremental algorithm and the second is an incremental algorithm in which vertices are assigned initial coordinates computed in the merge stage, but vertex pinning and spacer vertices are not used.

The density metric is plotted in Figure 7. Higher values in the graph represent sparse clusters, which should be avoided. All three algorithms produce similar results, which means that the incremental algorithm manages to compute compact layouts of the graph. Figure 8 shows the sum of the displacements of clusters between each pair of successive layouts. Lower values imply higher stability in the location of clusters. As can be seen, our algorithm outperforms the other algorithms. Figure 9 depicts the number of clusters that maintain their size between each pair of successive layouts. Higher values imply that there are less modifications to cluster outlines. It is clear from the graphs that our algorithm produces much better results than the other algorithms. Finally, Figure 10 depicts the running times of the algorithms. Both incremental algorithms take more time to compute than the non-incremental algorithm. This is mostly due to the extra processing done in the merge stage.

Table 1 summarizes the average values of each of the above metrics. All algorithms produce similar cluster densities. The cluster displacement of our algorithm is by far superior to the non-incremental algorithm, averaging about one twelfth of the non-incremental algorithm. Reducing the movement of clusters has indeed been one of the main design goals of the algorithm. The average percentage of clusters that remain with the same size in our algorithm is about four times as much as the non-incremental algorithm. This is facilitated by the spacer vertices that are used to minimize visual changes to the graph. Finally, the running times of both incremental algorithms is about twice the running time of the non-incremental algorithm, which is reasonable.

5 Conclusion and Future Work

We have presented an online algorithm for incremental layout of clustered graphs. The algorithm uses a force directed static layout tool as a basic building block. The key idea of the algorithm is to establish priorities of avoiding changes. First and foremost, movement of clusters should be avoided, because clusters give insight into the basic structure of the graph. Then, movement of vertices should be avoided, since vertices convey information regarding the size of the clusters and aid in navigating the graph. Movement of edges is considered the least critical.

To achieve this, our algorithm incorporates a few novel concepts. First, crucial vertices (dummy and old) are pinned down. Second, invisible place-holders are used to minimize changes. Finally, lengths and weights of edges are used to control both vertex placement and graph modifications.

It has been demonstrated that the algorithm computes a compact and space efficient graph layout, while minimizing the displace-

ment and changes to clusters between layout iterations.

The algorithm has been applied to the visualization of mobile object environments, where both real and simulated data has been tested. Good results have been achieved at the expense of higher running times. This is due both to the added complexity of the algorithm and to the fact that our implementation is only loosely coupled to the underlying static layout tool.

In future research, we plan to investigate enhancements to our 3D display mode. We would also like to extend the spacer vertices concept to drawing the cluster boundaries. Allowing some flexibility in fitting the boundary around the vertices in the cluster might improve the layout. An additional layout stage where each cluster is modeled as a non-uniform node could help improve cluster separation [5]. Finally, using stronger constraints when a second layout is necessary might further improve the dynamic stability of the algorithm.

Acknowledgements

This work was partially supported by European FP6 NoE grant 506766 (AIM@SHAPE) and by the Israeli Ministry of Science, grant 01-01-01509.

REFERENCES

[1] F. Bertault and M. Miller. An algorithm for drawing compound graphs. In J. Kratochvíl, editor, *Proc. 7th Int. Symp. Graph Drawing (GD 1999)*, number 1731 in Lecture Notes in Computer Science, LNCS, pages 197–204. Springer-Verlag, 2000.

[2] U. Brandes. 4. drawing on physical analogies. *Lecture Notes in Computer Science, LNCS*, 2025:71–86, 2001.

[3] J. Branke. 9. dynamic graph drawing. *Lecture Notes in Computer Science, LNCS*, 2025:228–246, 2001.

[4] R. Brockenauer and S. Cornelsen. 8. drawing clusters and hierarchies. *Lecture Notes in Computer Science, LNCS*, 2025:193–227, 2001.

[5] J. H. Chuang, C. C. Lin, and H. C. Yen. Drawing graphs with nonuniform nodes using potential fields. In G. Liotta, editor, *Proc. 11th Int. Symp. Graph Drawing (GD 2003)*, number 2912 in Lecture Notes in Computer Science, LNCS, pages 460–465. Springer-Verlag, 2004.

[6] W. R. Cockayne and M. Zyda, editors. *Mobile Agents*. Prentice Hall, 1998.

[7] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In S. Diehl, J. T. Stasko, and S. N. Spencer, editors, *Proceedings ACM 2003 Symposium on Software Visualization*, pages 77–86. ACM, 2003.

[8] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Computational Geometry: Theory and Applications*, 4(5):235–282, 1994.

[9] S. Diehl and C. Gorg. Graphs, They Are Changing - Dynamic Graph Drawing for a Sequence of Graphs. In M. T. Goodrich and S. G. Kobourov, editors, *Proc. 10th Int. Symp. Graph Drawing (GD 2002)*, number 2528 in Lecture Notes in Computer Science, LNCS, pages 23–31. Springer-Verlag, 2002.

[10] T. Dwyer. Three dimensional UML using force directed layout. In P. Eades and T. Pattison, editors, *Australian Symposium on Information Visualisation, (invis.au 2001)*, volume 9 of *Conferences in Research and Practice in Information Technology*, pages 77–85, Sydney, Australia, 2001. ACS.

[11] P. Eades and Q. W. Feng. Multilevel visualization of clustered graphs. In S. C. North, editor, *Proc. 4th Int. Symp. Graph Drawing (GD 1996)*, number 1190 in Lecture Notes in Computer Science, LNCS, pages 101–112. Springer-Verlag, 18–20 September 1996.

[12] J. Ellson, E. R. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. Graphviz — open source graph drawing tools. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. 9th Int. Symp. Graph Drawing (GD 2001)*, number 2265 in Lecture Notes in Computer Science, LNCS, pages 483–484. Springer-Verlag, 2002.

[13] E. R. Gansner and S. C. North. Improved force-directed layouts. In S. Whitesides, editor, *Proc. 6th Int. Symp. Graph Drawing (GD 1998)*, number 1547 in Lecture Notes in Computer Science, LNCS, pages 364–373. Springer-Verlag, 1998.

[14] D. Harel and Y. Koren. A Fast Multi-Scale Algorithm for Drawing Large Graphs. *J. Graph Algorithms Appl.*, 6(3):179–202, 2002.

[15] D. Harel and Y. Koren. Drawing graphs with non-uniform vertices. In *Proc. Working Conference on Advanced Visual Interfaces (AVI'02)*, pages 157–166. ACM Press, 2002.

[16] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[17] O. Holder, I. Ben-Shaul, and H. Gazit. Dynamic layout of distributed applications in fargo. In *Proceedings of the 1999 International Conference on Software Engineering*, pages 163–173. IEEE Computer Society Press / ACM Press, 1999.

[18] M. L. Huang and P. Eades. A fully animated interactive system for clustering and navigating huge graphs. In S. Whitesides, editor, *Proc. 6th Int. Symp. Graph Drawing (GD 1998)*, number 1547 in Lecture Notes in Computer Science, LNCS, pages 374–383. Springer-Verlag, 1998.

[19] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, April 1989.

[20] M. Kaufmann and D. Wagner, editors. *Drawing Graphs: Methods and Models*. Number 2025 in Lecture Notes in Computer Science, LNCS. Springer-Verlag, 2001.

[21] D. Lange and M. Oshima. Seven Good Reasons for Mobile Agents. *Communications of the ACM*, 42(3):88–89, 1999.

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

[23] S. C. North. Incremental layout in dynadag. In F. J. Brandenburg, editor, *Proc. 3rd Int. Symp. Graph Drawing (GD 1995)*, number 1027 in Lecture Notes in Computer Science, LNCS, pages 409–418. Springer-Verlag, 1995.

[24] Object Management Group. *The Common Object Request Broker: Architecture and Specification. Revision 2.2*, February 1998.

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

[26] Sun Microsystems, Inc. *Java Remote Method Invocation (RMI) Specification*, December 1997.

[27] I. G. Tollis, G. Di Battista, P. Eades, and R. Tamassia. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[28] Tom sawyer graph layout toolkit, 2004. Currently Available at http://www.tomsawyer.com.

[29] C. Walshaw. A Multilevel Algorithm for Force-Directed Graph Drawing. *J. Graph Algorithms Appl.*, 7(3):253–285, 2003.

[30] X. Wang and I. Miyamoto. Generating customized layouts. In F. J. Brandenburg, editor, *Proc. 3rd Int. Symp. Graph Drawing (GD 1995)*, number 1027 in Lecture Notes in Computer Science, LNCS, pages 504–515. Springer-Verlag, 1996.

[31] R. Wiese, M. Eiglsperger, and M. Kaufmann. yfiles: Visualization and automatic layout of graphs. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. 9th Int. Symp. Graph Drawing (GD 2001)*, number 2265 in Lecture Notes in Computer Science, LNCS, pages 453–454. Springer-Verlag, 2001.

[32] R. Wilson and R. Bergeron. Dynamic hierarchy specification and visualization. In *Proc. IEEE Symp. Information Visualization, InfoVis*, pages 65–72, 1999.

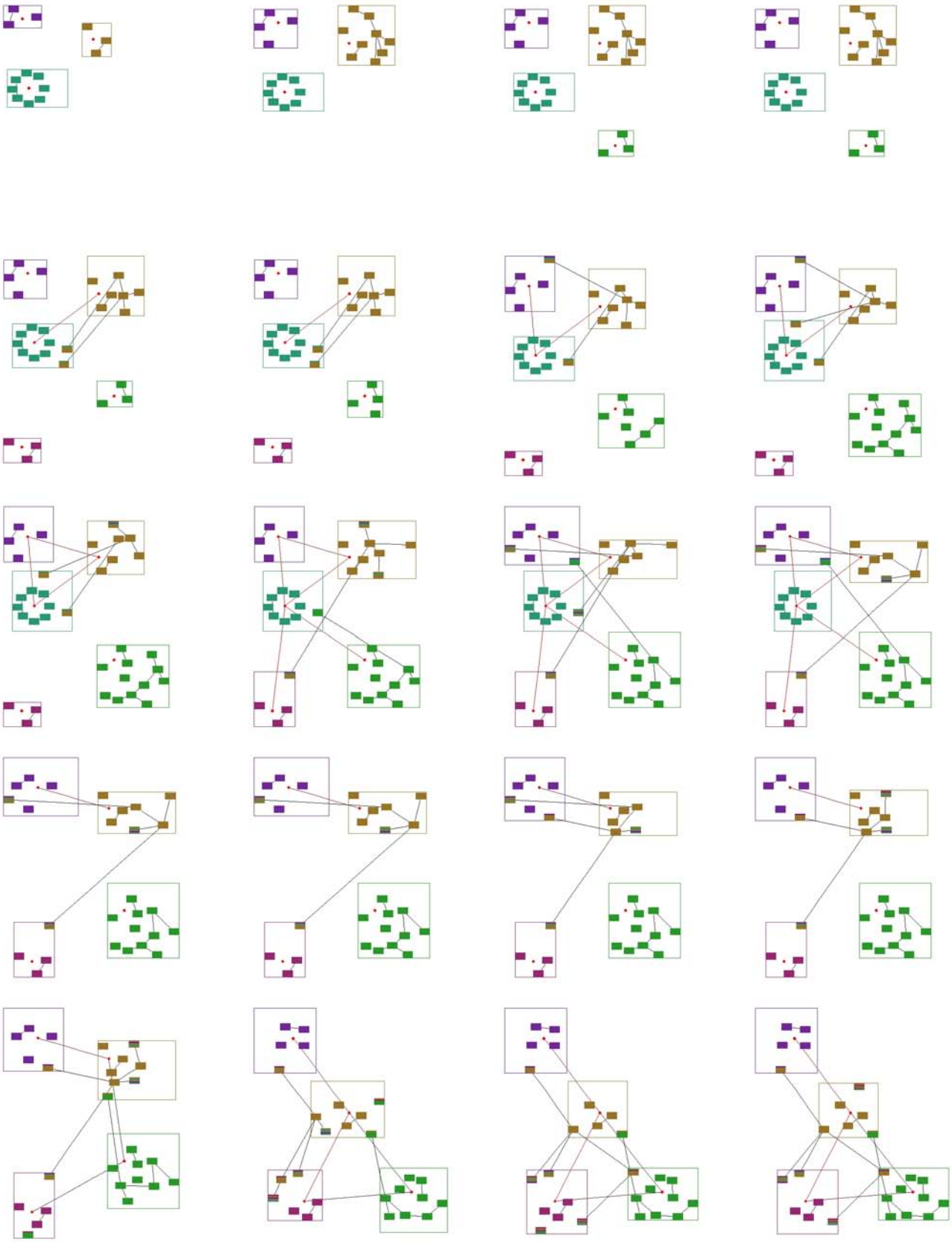


Figure 6: Sample animation sequence (from left to right and top to bottom)

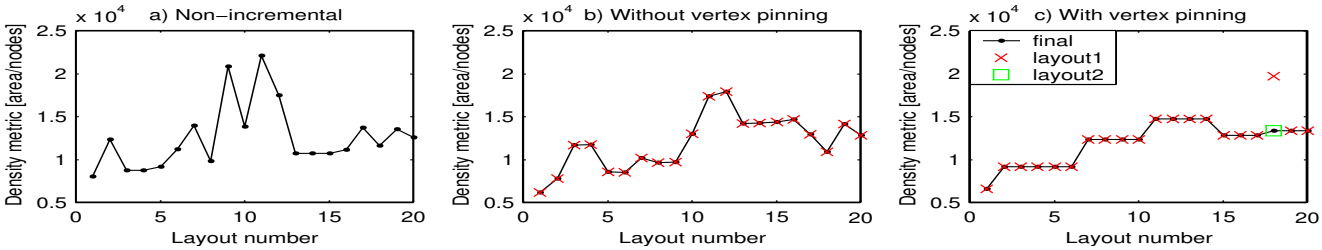


Figure 7: Density metric

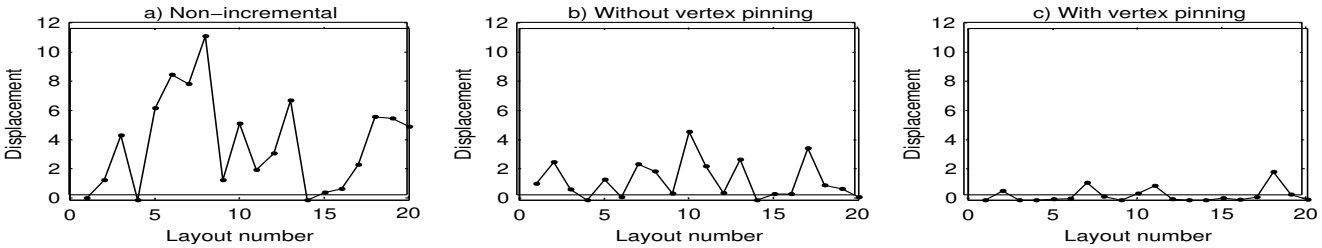


Figure 8: Sum of cluster displacements

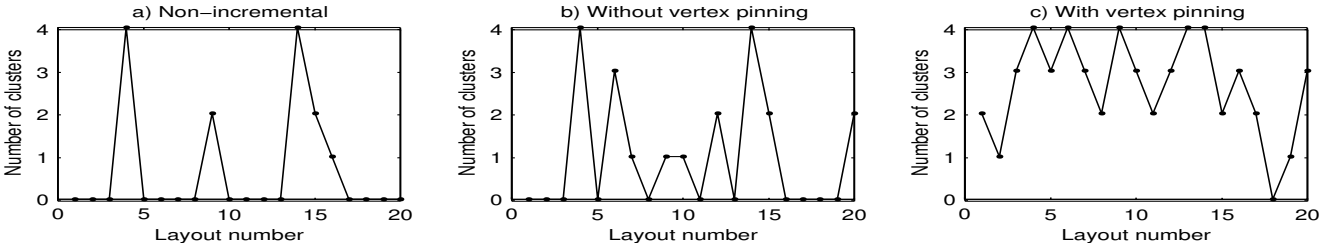


Figure 9: Number of clusters with the same size

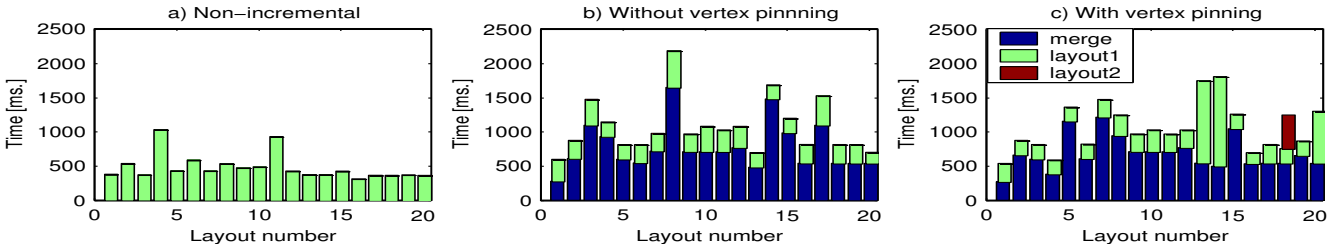


Figure 10: Running times

Average \ Algorithm	non-incremental	without vertex pinning	with vertex pinning
density metric [*area**vertices*]	$1.2516x10^4$	$1.1994x10^4$	$1.1936x10^4$
cluster displacement [distance]	4.0193	1.4118	0.3311
fraction of clusters with the same size	0.1575	0.23	0.615
running time [ms.]	492	1076	1084

Table 1: Average results of an animation sequence