Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 9, no. 1, pp. 165–182 (2005)

Simultaneous Graph Drawing: Layout Algorithms and Visualization Schemes

C. Erten *S. G. Kobourov* *V. Le* *A. Navabi*

Department of Computer Science
University of Arizona
{cesim,kobourov,vle,navabia}@cs.arizona.edu

Abstract

In this paper we consider the problem of drawing and displaying a series of related graphs, i.e., graphs that share all, or parts of the same node set. We present three algorithms for simultaneous graph drawing and three visualization schemes. The algorithms are based on a modification of the force-directed algorithm that allows us to take into account node weights and edge weights in order to achieve mental map preservation while obtaining individually readable drawings. The algorithms and visualization schemes have been implemented and the system can be downloaded at `http://simg.cs.arizona.edu/`.

Article Type	Communicated by	Submitted	Revised
regular paper	G. Liotta	February 2004	July 2005

This work is partially supported by the NSF under grant ACR-0222920.
An extended abstract of this paper appeared in GD 2003.

1 Introduction

Consider the problem of drawing a series of graphs that share all, or parts of the same node set. The graphs may represent different relations between the same set of objects. For example, in social networks, graphs are often used to represent different relations between the same set of entities. Alternatively, the graphs may be the result of a single relation that changes through time. For example, in software visualization, the inheritance graph in a Java program changes as the program is being developed. Consider the graphs in Fig. 1. There are two simultaneously displayed graphs that represent two snapshots of a file system structure rooted at the directory `graphs/`. The drawing conveys well both underlying structures and it is easy to identify the changes between the two snapshots.

In this paper, we attempt to address the following problem: Given a series of graphs that share all, or parts of the same node set, what is a natural way to layout and display them? The layout and display of the graphs are different aspects of the problem, but also closely related, as a particular layout algorithm is likely to be matched best with a specific visualization technique. As stated above, however, the problem is too general and it is unlikely that one particular layout algorithm will be best for all possible scenarios. Consider the case where we only have a pair of graphs in the series, and the case where we have hundreds of related graphs. The "best" way to layout and display the two series is likely going to be different. Similarly, if the graphs in the sequence are very closely related or not related at all, different layout and display techniques may be more appropriate. With this in mind, we consider several different algorithms and visualization models.

For the layout of the graphs, there are two important criteria to consider: the *readability* of the individual layouts and the *mental map preservation* in the series of drawings. The readability of individual drawings depends on aesthetic criteria such as display of symmetries, uniform edge lengths, and minimal number of crossings [18]. Preservation of the mental map can be achieved by ensuring that common substructures (nodes, edges, subgraphs etc.) that appear in consecutive graphs in the series, are drawn in a similar fashion [16]. These two criteria are often contradictory. If we individually layout each graph, without regard to other graphs in the series, we may optimize readability at the expense of mental map preservation. Conversely, if we fix the node positions in all graphs and draw common edges the same way, we are optimizing the mental map preservation but the individual layouts may be far from readable.

1.1 Related Work

Classical force-directed methods [1] for graph drawing typically begin with an initial random placement of the nodes inside a drawing window frame of prespecified size. The graph is then treated as a system of interacting physical objects, based on attractive and repulsive forces [10] or graph distances [15]. Force-directed layout algorithms employ an energy function that characterizes

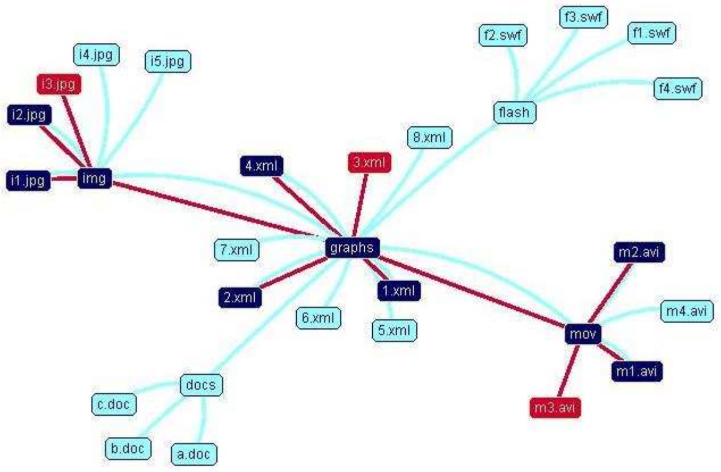


Figure 1: Two snapshots of a file structure rooted at directory `graphs/`. Red nodes and edges belong to the earlier snapshot. Dark blue nodes belong to both snapshots. Light blue nodes and edges belong to the later snapshot. The edges of the later snapshot are curved.

the state of the system. The minimization of suitably chosen energy functions tends to produce aesthetically pleasing graph drawings.

In the algorithm of Fruchterman and Reingold [10], the attractive and repulsive forces are respectively defined as, $f_r(d) = -\kappa^2/d$ and $f_a(d) = d^2/\kappa$, where d is the distance between two nodes. The repulsive forces are calculated for each pair of nodes whereas the attractive forces are calculated for pairs of nodes connected by an edge. The ideal distance between nodes, κ, is defined as $\kappa = C\sqrt{A_{frame}/n}$, where A_{frame} is the area of the drawing window frame, C is a constant determining how the nodes fill the frame, and n is the total number of nodes.

Several variations of force-directed methods for edge-weighted graphs have been proposed. In [8, 12] edge-weighted graphs are drawn so that the length of edges is proportional to their weights. Similarly, layouts for node-weighted graphs have also been considered in the context of focus-nodes that apply repulsive force proportional to their weight, so that the neighborhoods of such nodes will not be too cluttered [14].

In dynamic graph drawing the goal is to maintain a nice layout of a graph that is modified via operations such as insert/delete edge and insert/delete node. Techniques based on static layouts have been used [4, 13, 19]. North [17] studies the incremental graph drawing problem in the DynaDAG system. Brandes and Wagner adapt the force-directed model to dynamic graphs using a Bayesian framework [3]. Diehl and Görg [6] consider graphs in a sequence to create smoother transitions.

Brandes and Corman [2] present a system for visualizing network evolution in which each modification is shown in a separate layer of 3D representation with nodes common to two layers represented as columns connecting the layers. Thus, mental map preservation is achieved by precomputing good locations for the nodes and fixing the position throughout the layers. Dwyer and Eades [7] describe a method for visualizing time dependent flow in a network of objects by representing time as the third dimension, using columns of varying width. Along these lines, Collberg *et al* [5] describe a graph-based system for visualization of software evolution, while preserving the mental map by fixing the locations of all common nodes in the evolving graph.

1.2 Our Contributions

In this paper we describe three approaches for visualizing a series of graphs that share all, or parts of, the same node set; see Fig. 2.

In the *aggregate view model* we show all the graphs in one combined drawing, using different edge (node) styles, to differentiate between the different graphs. The corresponding layout algorithm creates an *aggregate* graph from the given sequence of graphs. The aggregate graph is node-weighted and edge-weighted and the node (edge) weight corresponds to the number of times a particular node (edge) appears in the sequence. A modified force-directed approach is used to layout the aggregate graph, taking into account the weights of the nodes and the edges.

In the *merged view model* we create a 3D drawing, in which each graph is displayed in its own 2D plane, and the planes are arranged on top of each other in the order that the graphs appear in the sequence. In the corresponding layout algorithm, we create a *merged* graph. The merged graph consists of all the graphs in the sequence, together with additional edges connecting the same nodes in all graphs. A modified force-directed layout is used to layout the merged graph by restricting each graph to its own 2D plane.

In the *split view model* each graph is displayed in its own drawing window. The corresponding layout algorithm is designed for a pair of related graphs G_1 and G_2 but can be generalized to larger series of graphs. We use non-random placement of the nodes, based on graph distances, to independently obtain initial drawings D_1 and D_2 for the two graphs. Next the placement of the nodes from $D_1(D_2)$ is used to "seed" an iteration of the force-directed layout for $G_2(G_1)$ and the process is repeated either until the two layouts converge or until the iteration count reaches a fixed constant.

The three view models closely match their corresponding layout algorithms. However, our system allows all possible combinations of the presented view models and the layout algorithms creating nine different visualization schemes.

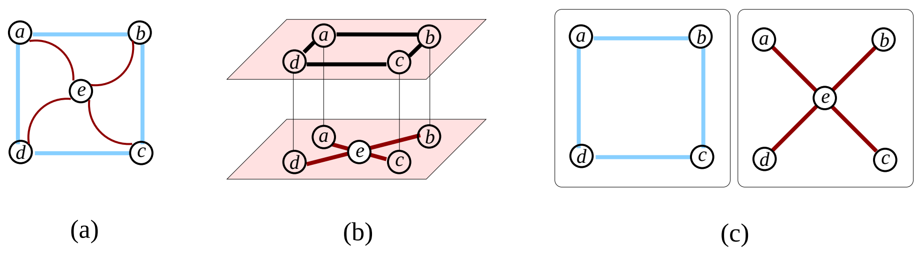


Figure 2: Layout and visualization: (a) aggregate; (b) merged; (c) split.

2 Modified Force-Directed Method

The first two algorithms presented in Sections 3 and 4 rely on a modification of the force-directed method by Fruchterman and Reingold [10]. This modification can be thought of as an extension of the traditional algorithm, that takes into account node weights and edge weights in the force calculations. Thus, we assume that the input graphs in the series are simple unweighted graphs and we are free to assign appropriately chosen weights.

Given a series of graphs $G_1, G_2, \ldots, G_k$, we create a graph $G_{Alg} = (V_{Alg}, E_{Alg})$ such that both the nodes and the edges are assigned weights. Depending on the layout algorithm (either Aggregate or Merged), we provide two different constructions for G_{Alg}. The next two sections provide further details on these constructions. We use the node and edge weights to modify the standard force-directed algorithm as follows. If a node $v \in V_{Alg}$ has large weight then it is placed close to the center in the final layout. If an edge $(u, v) \in E_{Alg}$ has large weight then the nodes u and v are placed close to each other in the final layout. This is a simple heuristic, but it ensures that:

- persistent nodes remain close to the center of the layout, while transient nodes are placed on the periphery;
- nodes that are adjacent in many of the graphs in the series are placed close together.

In order to handle the node weights, we place a dummy node in the center of the frame and ensure that it attracts all the other nodes in proportion to their weights. We formulate this new central attraction force as, $f_{ca}(v) = d^2 \times w(v)/\kappa$, where $w(v)$ is the weight of the node, d is its distance from the center and κ is the ideal distance.

To handle edge weights, we scale the attractive forces by their edge weights and the new formulation of the attractive forces becomes, $f_a(e) = d^2 \times w(e)/\kappa$, where $w(e)$ is the weight of the edge e. If two nodes are connected by an edge with weight 0, then the modified algorithm treats them as if they are not connected at all.

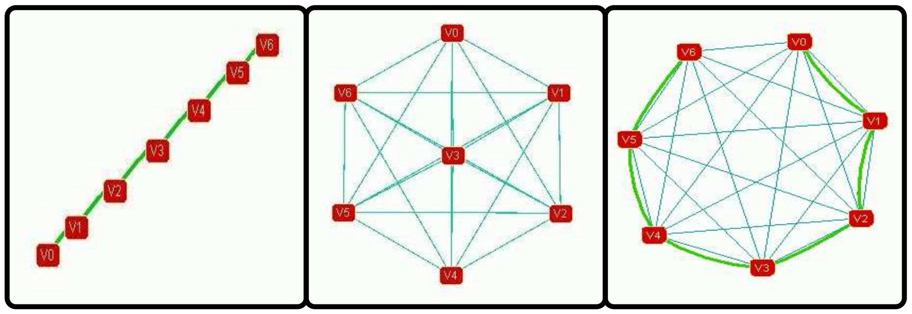


Figure 3: Left: Individual layout of P_7 drawn with curved edges. Middle: Individual layout of K_7 drawn with straight-line segments. Right: Simultaneous embedding of P_7 and K_7 obtained from the aggregate layout method.

3 Aggregate Graph Layout and Visualization

This algorithm and its matching visualization scheme combine all the related graphs into a single drawing. The location of the nodes is fixed for all graphs thus perfectly preserving the mental map (possibly at the expense of readability of the individual graphs). The edges of different graphs are distinguished by employing different edge styles. This algorithm/visualization combination is most suitable when dealing with a small number of graphs or when the graphs are very similar to each other.

3.1 Aggregate View Visualization

In the aggregate view we only display a node once, even though it may appear in multiple graphs and we display all edges from all the graphs in the sequence; see Fig. 3. The graphs can be displayed in 2D or 3D and we employ different edge colors and edge styles to differentiate between the different graphs. Displaying all graphs using a single node set allows the viewer to see multiple graphs at the same time and view the difference in relationships more easily. Different edge colors and edge styles are used to distinguish between the relationships from each graph. For example in Fig. 3 the edges of one graph are drawn with green straight line segments, whereas the other graph is drawn with thicker curved edges in a different shade of green.

3.2 Aggregate Layout Algorithm

In the aggregate graph layout method we begin by creating the node-weighted and edge-weighted graph $G_{Alg} = (V_{Alg}, E_{Alg})$ from the sequence of graphs, $G_1, G_2, \ldots, G_k$, as follows: A node $v \in V_{Alg}$ has weight $w(v)$, where $w(v)$ is the number of times it appears in the series. An edge $e \in E_{Alg}$ has weight $w(e)$, where $w(e)$ is the number of times it appears in the series. We then apply the modified force-directed layout algorithm to obtain a drawing for G_{Alg}. From this drawing we extract the drawings of each individual graph in the series. Thus, nodes and edges that are present more than once in the series are in the same position in all graphs that they appear in. This approach guarantees mental map preservation, possibly at the expense of good readability. Yet, if a graph G_i has many nodes/edges that exist in most of the graphs in the series, then G_i is an important graph and the resulting layout will be similar to that of an independent layout of G_i.

Fig. 3 shows the simultaneous layout of K_7, the complete graph on seven nodes and P_7, the path with seven nodes. The edges that belong to the path are drawn using curved and thick edges. Note that although an individual layout of K_7 would place one of the nodes in the middle, the simultaneous embedding with the aggregate layout method pushes that node out because of the presence of the path. A summary of the aggregate graph layout algorithm is in Fig. 4.

```
Aggregate Graph Layout
1   Construct $G_{Alg} = (V_{Alg}, E_{Alg})$:
        $V_{Alg} = V_1 \cup V_2 \cup \ldots \cup V_k$, $E_{Alg} = E_1 \cup E_2 \cup \ldots \cup E_k$
2   Assign weights to each $v \in V_{Alg}$ and $(u, v) \in E_{Alg}$:
        $w(v)$ = number of appearances of $v$ in $V_1, V_2, \ldots, V_k$
        $w(u, v)$ = number of appearances of $(u, v)$ in $E_1, E_2, \ldots, E_k$
3   Use the modified force-directed layout algorithm on $G_{Alg}$
4   Extract the layout of each $G_i$ from the layout of $G_{Alg}$
```

Figure 4: Aggregate Graph Layout. $G_1, G_2, \ldots, G_k$ are the input graphs.

4 Merged Graph Layout and Visualization

This algorithm and its matching visualization scheme is the most flexible of our algorithm/visualization combinations. All the graphs in the series are drawn, and the drawings are closely related and influence each other via edges connecting the same nodes on different planes. As the most flexible algorithm/visualization combination it is effective in most cases.

4.1 Merged View Visualization

In the merged view each of the graphs is drawn on its separate 2D plane, and the planes are layered in 3D in the order of appearance; see Fig. 5. All the graphs are shown on the same screen such that corresponding nodes in adjacent drawings have approximately the same positions on their planes. This model provides a clear mental mapping between the drawings of each graph. Moreover the balance between mental map preservation and individual graph readability can be interactively controlled. In our implementation, the user has full 3D control over the view, can move and rotate the view and zoom in and out.

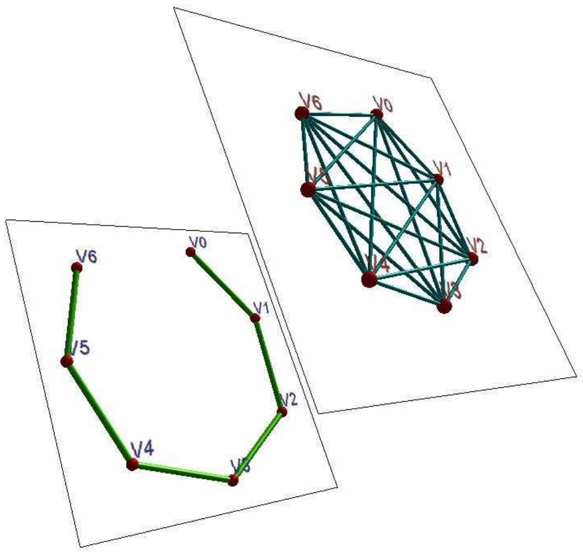


Figure 5: Simultaneous embedding of K_7 and P_7 using the merged graph layout method. The visualization is done in 3D using a separate plane for each graph.

4.2 Merged Layout Algorithm

In contrast to the aggregate method, the merged graph layout method does not guarantee perfect mental map preservation. The algorithm begins with the construction of $G_{Alg} = (V_{Alg}, E_{Alg})$. It is obtained by taking $G_1, G_2, \dots G_k$ and inter-connecting all corresponding nodes with a special class of edges, E_{new}. The edges in E_{new} connect consecutive instances of the same vertex. Thus, if a node v appears j times in the sequences, there will be j copies of it in G_{Alg}. Once the merged graph G_{Alg} has been created and the weights assigned, the modified force-directed method is applied.

The positions of corresponding nodes in each layout depend on how we assign weights to the edges in E_{new}. The larger the weight of edges in E_{new}, the closer the corresponding nodes in each separate layout will be. An important property of this layout method is the proximity of corresponding nodes in the final layout. Let $u_1, u_2, \dots u_i$ be all the nodes in G_{Alg} that correspond to the same node u and let $v_1, v_2, \dots v_j$ be all the nodes in G_{Alg} that correspond to the same node v. If $i < j$, then the merged layout algorithm will place the nodes corresponding to v closer to each other than the nodes corresponding to u.

In addition to this automatic assignment of weights to the edges in E_{new}, in our implementation we give further interactive control to the user to increase or decrease the importance of these weights. In particular, the user has control over a slide bar, the two extremes of which correspond to perfect readability (each graph is drawn independently, as the weights of the edges in E_{new} is

0) and perfect mental map preservation (corresponding nodes are fixed in the same position in all graphs in the series). Fig. 5 illustrates the simultaneous embedding resulting from the merged graph layout of K_7 and P_7. Note that although the locations of the corresponding nodes might not be the same, the mental map is still preserved since the relative locations of the corresponding nodes remain the same. A summary of the merged graph layout algorithm is in Fig. 6.

```
Merged Graph Layout
1  Rename the nodes in $V_1, V_2, \ldots, V_k$ so that each node is unique
2  Construct $E_{new}$ by connecting corresponding nodes in $V_1, V_2, \ldots, V_k$
3  Construct $G_{Alg} = (V_{Alg}, E_{Alg})$:
       $V_{Alg} = V_1 \cup V_2 \cup \ldots \cup V_k$, $E_{Alg} = E_1 \cup E_2 \cup \ldots \cup E_k \cup E_{new}$
4  Assign weights for the edges in $E_{new}$
5  Apply the modified force-directed layout algorithm on $G_{Alg}$
```

Figure 6: Merged Graph Layout. $G_1, G_2, \ldots, G_k$ are the input graphs.

5 Independent Iterations Layout and Split-View Visualization

The independent iterations layout algorithm attempts to balance readability and mental map preservation using the idea of seeding the layout of one graph with the layout of another. This approach and its matching split-view visualization are effective when dealing with a small number of graphs.

5.1 Split View Visualization

The two graphs are drawn separately in their own windows in two dimensions and both windows are on the same screen; see Fig. 7. The view model can be generalized to handle many graphs, in which case the screen would be split into many individual panes. Still, as the number of graphs to be visualized increases, the user's ability to read the relations between them will likely decrease.

5.2 Independent Iterations Algorithm

In the preceding two approaches we construct one global graph from the series (aggregate or combined) and extract the individual layout of each graph from the layout of the global graph. Our final layout method is quite different and we describe it here for only two graphs.

The algorithm begins by creating independent layouts for the two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. The layouts are obtained using non-random placement of the nodes, based on the graph distance. For each graph, we first

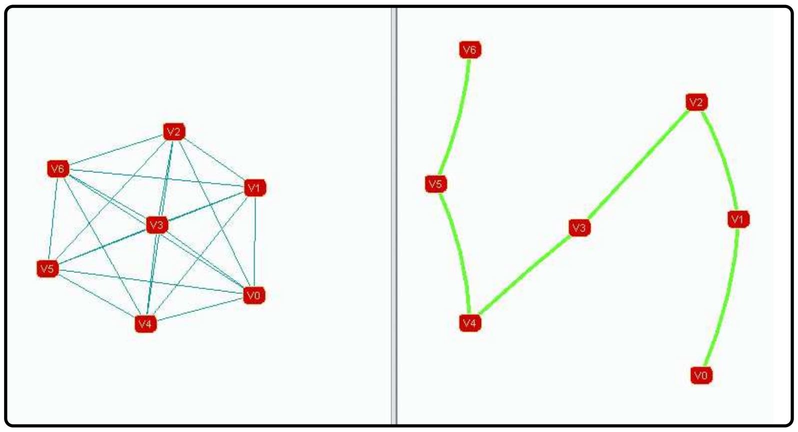


Figure 7: Simultaneous embedding of K_7 and P_7 using independent iterations layout method and split view model for visualization.

place three nodes at the endpoints of a triangle whose sides correspond to the graph distances (number of edges in the shortest path) between the respective nodes. Each following node is placed in a similar manner, using three previously placed nodes. Once all the nodes have been placed, several iterations of the force-directed method are applied. This approach is described in detail in [11].

At the conclusion of this stage, we have "good" layouts for each graph when they are drawn independently. As a result we obtain two different point-sets, P_1 and P_2 specifying the locations of the nodes in G_1 and G_2, respectively.

In the next step G_1 "borrows" the point-set P_2 of G_2 and treats it as an initial placement for the standard force-directed algorithm. Similarly, G_2 uses the point-set P_1 of G_1 and uses it as an initial placement for the standard force-directed algorithm. After applying force-directed iterations to both graphs (again independently) we arrive at two new point-sets P'_1 and P'_2. We repeat the process of point-set swapping and force-directed calculations until the resulting point-sets converge to a given threshold minimum desirable distance between them or until the number of iterations exceed a fixed constant.

Given a mapping between two point-sets, the distance between them can be measured as the sum of Euclidean distances between each pair of corresponding points in the point-sets. This simple metric is not well-suited to our problem as the following example shows: Assume layout l_2 is just a 90° rotation of layout l_1. Even though the topology of the layouts is the same, calculating the distance between l_1 and l_2 as the sum of Euclidean distances between points would be misleadingly high. To overcome this problem, we first align the two layouts as best as possible using rigid 3D motion. In particular, we apply an affine linear transformation on l_1 so that the layout of l_1 after the transformation, is as close as possible to l_2. The transformation consists of translation, rotation, scaling, and shearing.

Fig. 7 shows the simultaneous embedding of K_7 and P_7 resulting from independent iterations layout using the split view, described below. Note that the resulting layout for each graph is not the same as an individual layout for that graph. Instead, the independent iterations layout is a compromise between the two individual layouts. A summary of the layout algorithm is in Fig. 8.

```
Independent Iterations Layout
1  Using independent non-random placement obtain layouts $l_1$, $l_2$ for $G_1$, $G_2$
2  Apply a linear transformation on $l_1$ to align it to $l_2$
3  Let $mindist = dist(l_1, l_2)$ and $bestl_1 = l_1$, $bestl_2 = l_2$
4  Repeat until
   $(mindist < threshold)$ OR $(iterationcount > maxiterationcount)$:
         4.1 Apply layout algorithm on $G_1$ to get $l_{1'}$
              (using $l_2$ for initial placement)
         4.2 Apply layout algorithm on $G_2$ to get $l_{2'}$
              (using $l_1$ for initial placement)
         4.3 Apply a linear transformation to align $l_1$ to $l_2$
         4.4 If $dist(l_{1'}, l_{2'}) < mindist$
                 $mindist = dist(l_{1'}, l_{2'})$
                 $bestl_1 = l_{1'}$, $bestl_2 = l_{2'}$
         4.5 $l_1 = l_{1'}$, $l_2 = l_{2'}$, $iterationcount$ ++
```

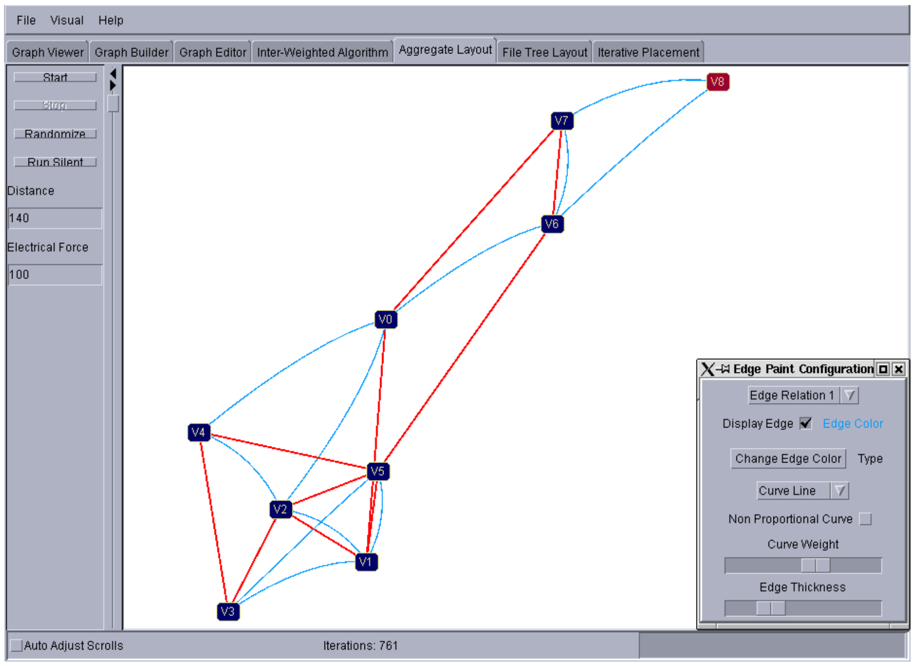
Figure 8: Independent Iterations Layout. G_1 and G_2 are the input graphs

The algorithm is defined for two graphs but can be extended to handle more graphs. The point-set swapping can be extended to swapping the point-sets of neighboring graphs in the sequence and the distance measure between a pair of layouts can be extended to measure distances between multiple point-sets. Our current implementation is for pairs of graphs.

6 Conclusion and Discussion

In this paper we present three algorithms for simultaneous graph drawing as well as three visualization schemes for viewing such drawings. An implementation in Java is available at `http://simg.cs.arizona.edu/`. In addition to the three layout methods and three visualization schemes, the system provides various capabilities such as graph editing and building some common classes of graphs, such as complete graphs, trees, paths, and random graphs. Graphs in GML format can be loaded and stored. All algorithms are animated so that the progress of the layout algorithms can be continuously monitored. In addition, nodes can be manipulated interactively, allowing the user control over the layout. Fig. 9 shows a snapshot of the system interface.

The images in Fig. 10 are obtained using the aggregate layout algorithm and the aggregate view visualization. The left image shows two paths, with 11 and



Distance
140
Electrical Force
100
Edge Relation 1
Display Edge ✓
Curve Line
Non Proportional Curve
Auto Adjust Scrolls
Iterations: 761

Figure 9: A snapshot of our simultaneous graph drawing system.

13 nodes respectively, that have four nodes in common. The right image shows two stars on 5 nodes, with different centers and common leaves.

The image in Fig. 11 is obtained using the merged layout algorithm and the merged view visualization. The data consists of a series of four subtrees of a tree on 13 nodes.

The image in Fig. 12 is obtained using the independent iterations layout and split-view visualization. The data consists of two graphs obtained from a file system (also shown using the aggregate layout algorithm and aggregate view in Fig. 1).

Finally, we show several graphs obtained from the Graph Drawing Proceedings database [9]. In Fig. 13 we show a piece of the Graph Drawing *topic graph*. A topic graph for a given time interval has as nodes the dominant keywords/phrases used in publications during that period. The edges connect the keywords/phrases that co-occur. We show two pieces of the topic graphs for the time periods 1996-1998 and 1998-2000. The images from top to bottom are obtained using the aggregate layout and aggregate view, the merged layout and merged view, and the independent iterations layout and split-view, respectively. In the top image, the nodes corresponding to the first time period are gray, the nodes corresponding to the second time period are white, and common nodes are black. In the bottom two images, the graph with the red edges represents

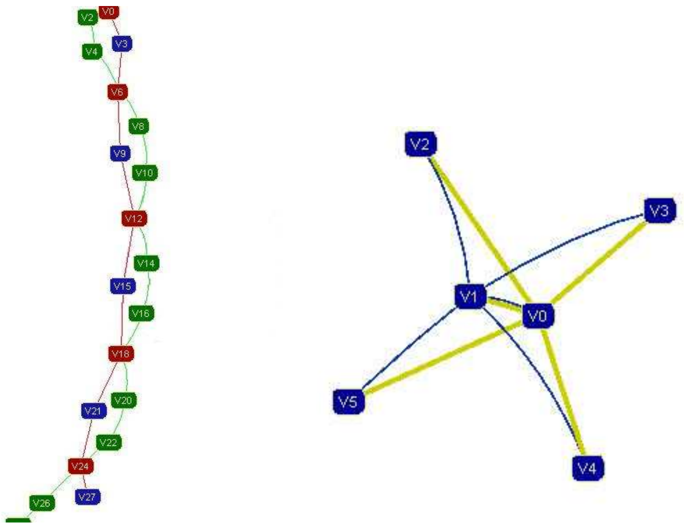


Figure 10: Two paths and two stars using aggregate layout and aggregate view.

the first period and the graph with the green edges represents the second.

In Fig. 14 we show a piece of the *collaboration graph* from the Graph Drawing Proceedings database. The nodes in a collaboration graph are authors and the edges represent collaborations between the authors in the given time period. We show two pieces of the collaboration graphs for the time periods 1995-1999 and 1999-2003.

7 Acknowledgments

We would like to thank the anonymous referees for their insightful comments and suggestions.

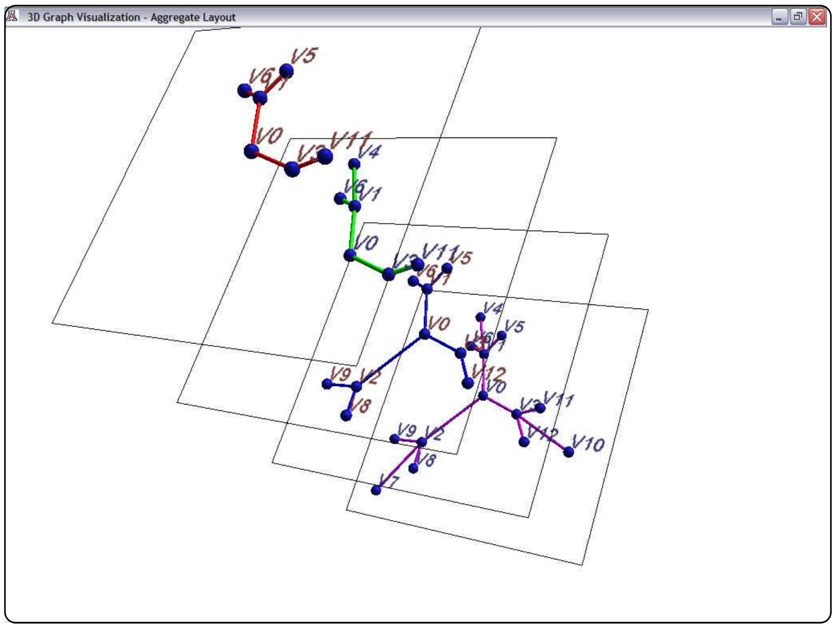


Figure 11: Four subtrees using merged layout and merged view.

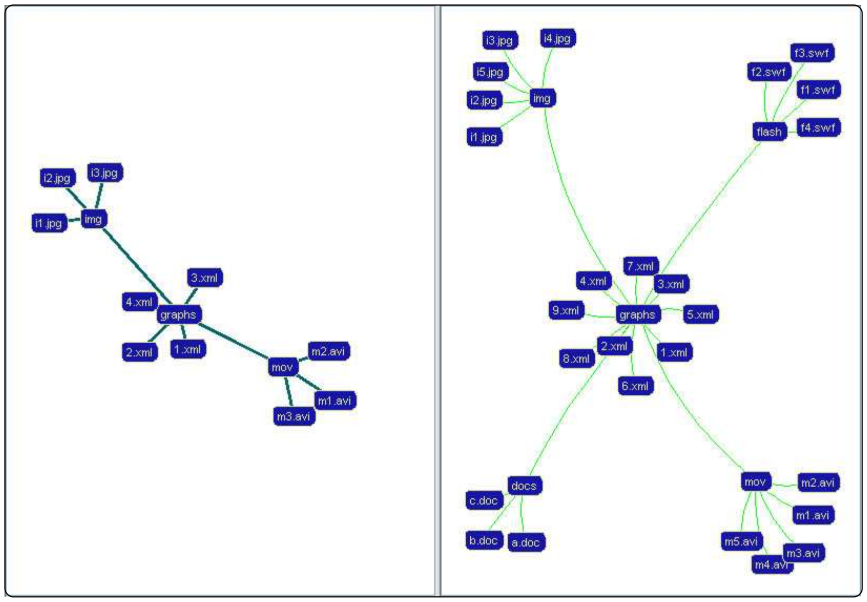


Figure 12: File structure at two different times, using independent iterations and split view.

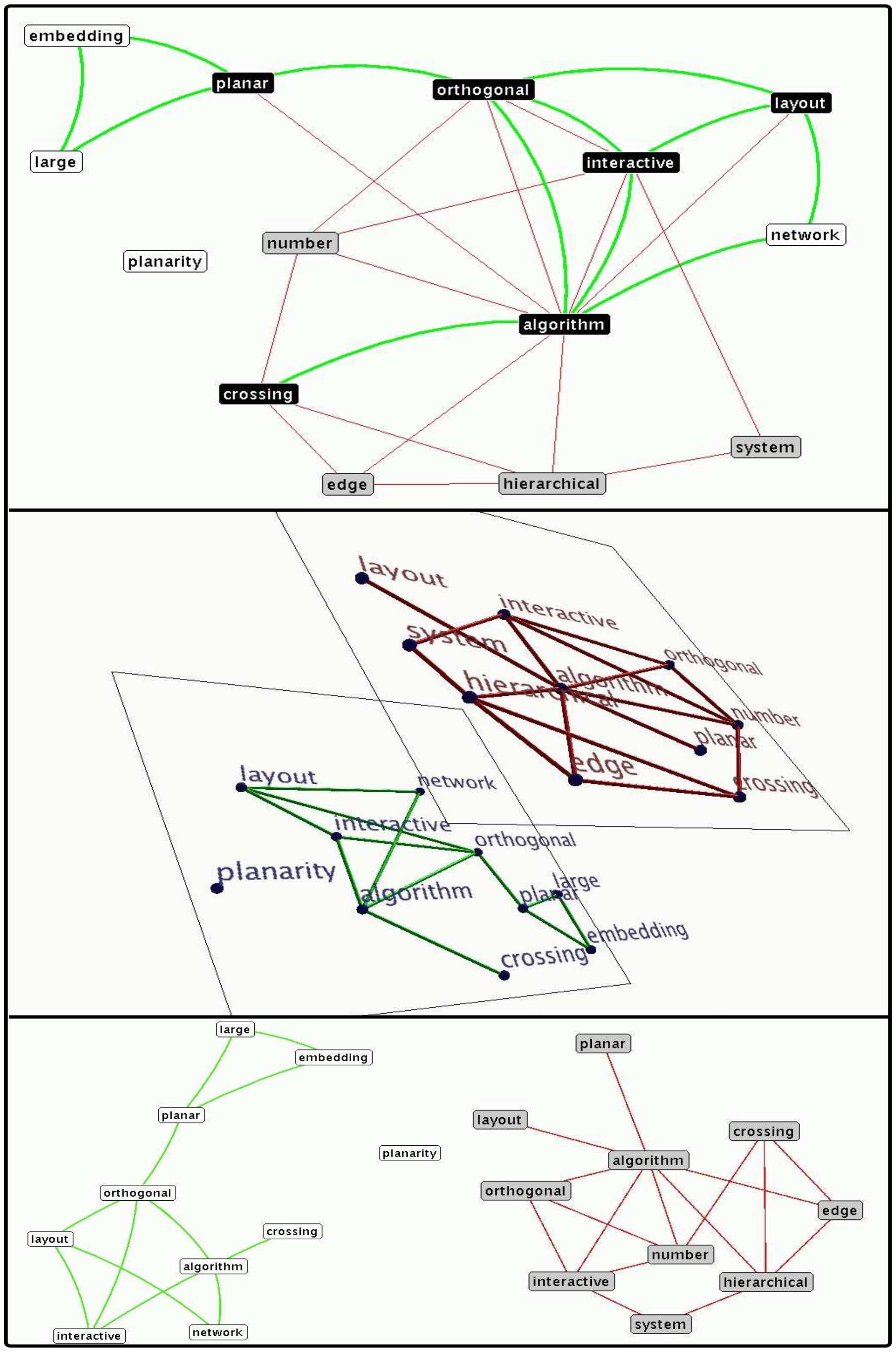


Figure 13: Matching layout algorithms and views applied to a topic graph.

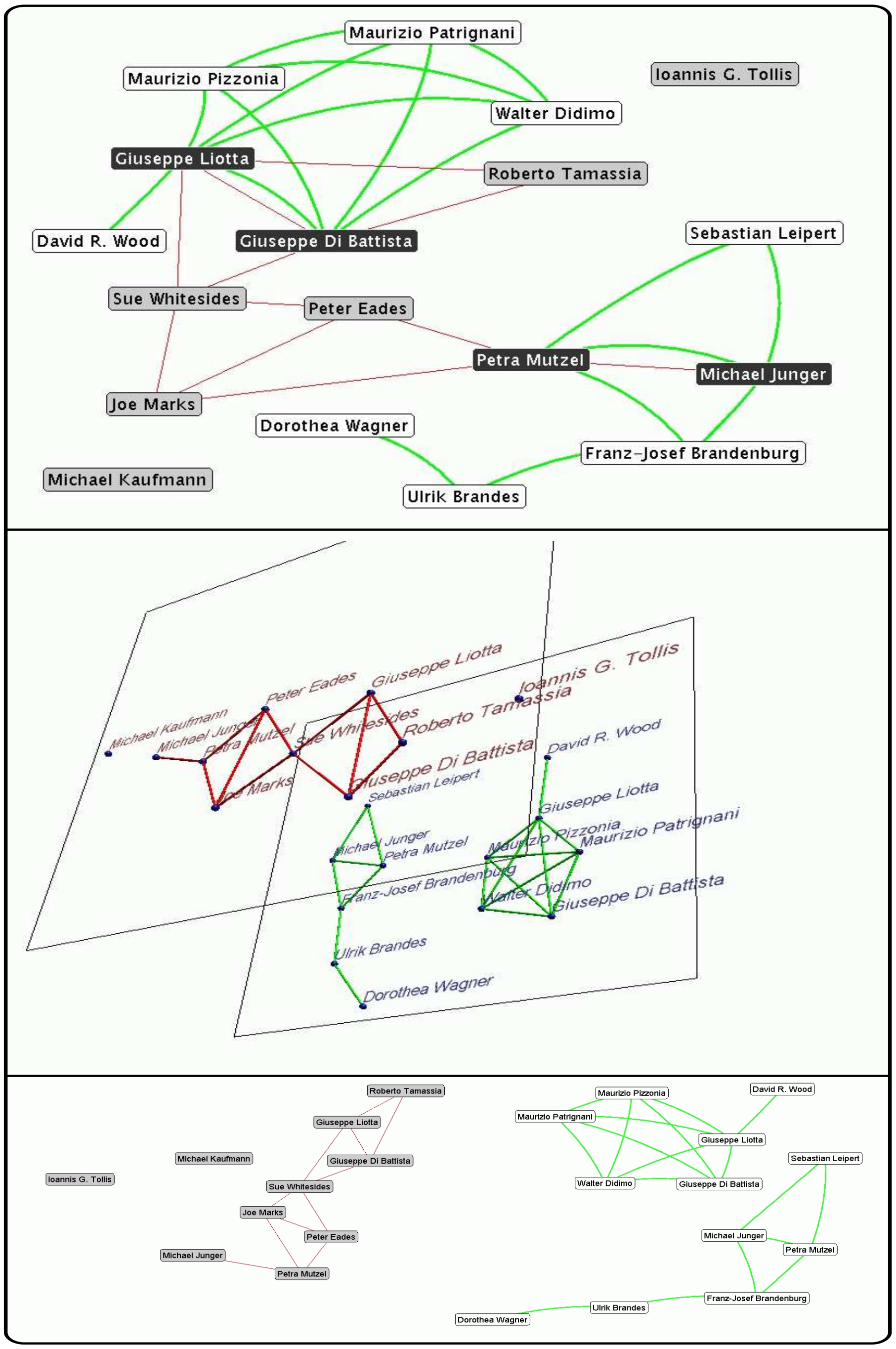


Figure 14: Matching layout algorithms and views applied to a collaboration graph.

References

[1] U. Brandes. Drawing on physical analogies. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs*, volume 2025 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2001.

[2] U. Brandes and S. R. Corman. Visual unrolling of network evolution and the analysis of dynamic discourse. In *IEEE Symposium on Information Visualization (INFOVIS '02)*, pages 145–151, 2002.

[3] U. Brandes and D. Wagner. A bayesian paradigm for dynamic graph layout. In *Proceedings of the 5th Symposium on Graph Drawing (GD)*, volume 1353 of *LNCS*, pages 236–247, 1998.

[4] J. Branke. Dynamic graph drawing. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs: Methods and Models*, number 2025 in LNCS, chapter 9, pages 228–246. Springer-Verlag, Berlin, Germany, 2001.

[5] C. Collberg, S. G. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In *ACM Symposium on Software Visualization*, pages 77–86, 2003.

[6] S. Diehl and C. Görg. Graphs, they are changing. In *Proceedings of the 10th Symposium on Graph Drawing (GD)*, pages 23–30, 2002.

[7] T. Dwyer and P. Eades. Visualising a fund manager flow graph with columns and worms. In *Proceedings of the 6th International Conference on Information Visualisation, IV02*, pages 147–158. IEEE Computer Society, 2002.

[8] P. Eades and C. F. X. de Mendonça Neto. Vertex splitting and tension-free layout. In *Graph Drawing*, volume 1027 of *Lecture Notes in Computer Science*, pages 202–211. Springer, 1996.

[9] C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. Yee. GraphAEL: Graph animations with evolving layouts. In *11th Symposium on Graph Drawing*, pages 98–110, 2003.

[10] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Softw. – Pract. Exp.*, 21(11):1129–1164, 1991.

[11] P. Gajer and S. G. Kobourov. GRIP: Graph dRawing with Intelligent Placement. *Journal of Graph Algorithms and Applications*, 6(3):203–224, 2002.

[12] D. Harel and Y. Koren. Drawing graphs with non-uniform vertices. In *Proceedings of Working Conference on Advanced Visual Interfaces*, pages 157–166, 2002.

[13] 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.

[14] M. L. Huang, P. Eades, and J. Wang. On-line animated visualization of huge graphs using a modified spring algorithm. *Journal of Visual Languages and Computing*, 9:623–645, 1998.

[15] T. Kamada and S. Kawai. Automatic display of network structures for human understanding. Technical Report 88-007, Dept. of Inf. Science, University of Tokyo, 1988.

[16] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *J. Visual Lang. Comput.*, 6(2):183–210, 1995.

[17] S. C. North. Incremental layout in DynaDAG. In *Proceedings of the 4th Symposium on Graph Drawing (GD)*, pages 409–418, 1996.

[18] H. Purchase. Which aesthetic has the greatest effect on human understanding? In G. Di Battista, editor, *Proceedings of the 5th Symposium on Graph Drawing (GD)*, pages 248–261, 1998.

[19] K.-P. Yee, D. Fisher, R. Dhamija, and M. Hearst. Animated exploration of dynamic graphs with radial layout. In *IEEE Symposium on Information Visualization (INFOVIS '01)*, pages 43–50, 2001.