Simultaneous Graph Drawing: Layout Algorithms and Visualization Schemes*

(System Demo)

C. Erten, S. G. Kobourov, V. Le, and 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 vertex set. We designed and implemented three different algorithms for simultaneous graphs drawing and three different visualization schemes. The algorithms are based on a modification of the force-directed algorithm that allows us to take into account vertex weights and edge weights in order to achieve mental map preservation while obtaining individually readable drawings. The implementation is in Java and the system can be downloaded at `http://simg.cs.arizona.edu/`.

1 Introduction

Consider the problem of drawing a series of graphs that share all, or parts of the same vertex 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 vertex 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.

* This work is partially supported by the NSF under grant ACR-0222920.

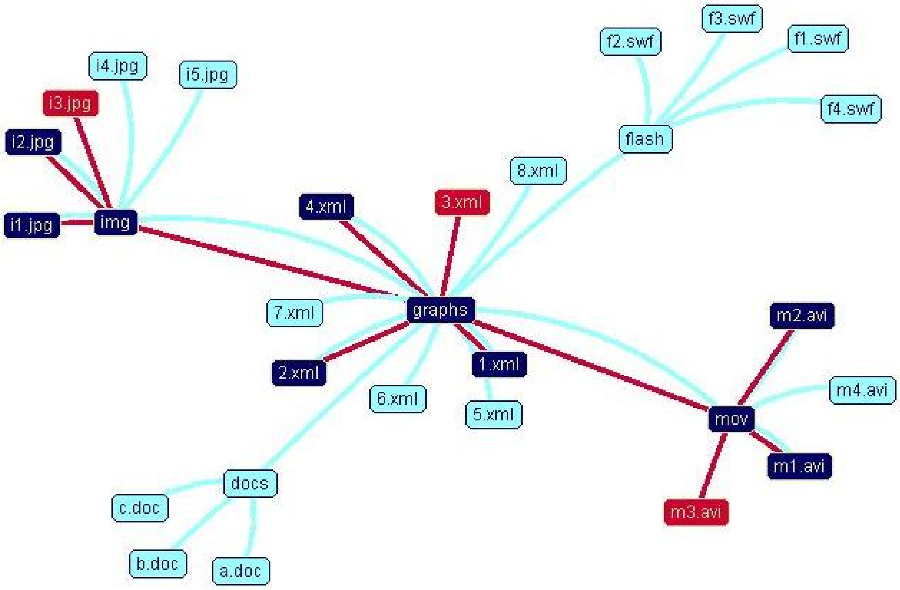


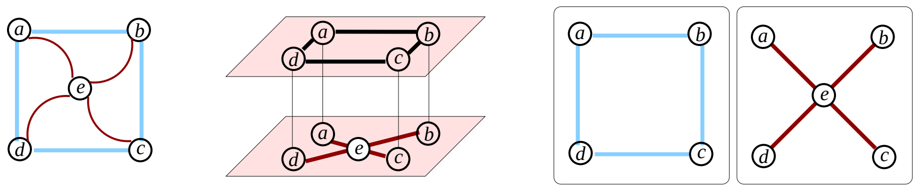
Fig. 1. Two snapshots of the file structure rooted at directory `graphs/`. Red vertices and edges belong to earlier snapshot. Dark blue vertices belong to both snapshots. Light blue vertices and edges belong to the later snapshot. The edges of later snapshot are curved.

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. Preservation of the mental map can be achieved by ensuring that vertices that appear in consecutive graphs in the series, remain in the same positions. 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 vertex positions in all graphs, we are optimizing the mental map preservation but the individual layouts may be far from readable.

For the visualization of the graphs there are numerous different possibilities. We could draw each graph in the series in its own 2D plane, in order of appearance, or we could show one graph at a time, and morph to the next one. If there are only a small number of graphs in the sequence, we could display all of them simultaneously, using different edge styles.

We designed and implemented three layout algorithms and three visualization schemes; see Fig. 2. We summarize the layout algorithms below:

1. In the first layout algorithm we create 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.
2. In the second 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



(a) (b) (c)

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

vertices in all graphs. A modified force-directed layout is used to layout the merged graph by restricting each graph to its own 2D plane.

3. The third 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 intelligent (rather than random) placement of the vertices, based on graph distances, to independently obtain initial drawings D_1 and D_2 for the two graphs. Next the placement of the vertices 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 until the two layouts converge.

The three visualization schemes closely correspond to the algorithms above. However, different combinations of layout algorithms and visualization schemes can also be used. We summarize the layout models below:

1. In the *aggregate view model* we use the aggregate graph to show all the graphs in one combined drawing, using different edge(node) styles, to differentiate between the different graphs.
2. 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.
3. In the *split view model* each graph is displayed in its own drawing window.

2 Previous Work

Classical force-directed methods [9, 11, 18] for graph drawing use a random initial embedding of the graph and treat the graph as a system of interacting physical objects. Force-directed layout algorithms typically employ an energy function that characterizes the state of the system. The minimization of suitably chosen energy functions tends to produce aesthetically pleasing graph drawings. Several variations of force-directed methods for edge-weighted graphs have been proposed. In [1, 15, 14] edge-weighted graphs are drawn so that the length of edges is proportional to their weights. Similarly, layouts for vertex-weighted graphs have also been considered in the context of focus-vertices that apply repulsive force proportional to their weight, so that the neighborhoods of such vertices will not be too cluttered [17].

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 vertex. Techniques based on static layouts have been used [4, 16, 21]. North [20] 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 [8] consider graphs in a sequence to create smoother transitions. Special classes of graphs such as trees, series-parallel graphs and st-graphs have also been studied in dynamic models [6, 19]. 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 vertices common to two layers represented as columns connecting the layers. Thus, mental map preservation is achieved by precomputing good locations for the vertices and fixing the position throughout the layers.

Simultaneous planar graph embedding is a related problem that asks whether there exist locations for the vertices of two different planar graphs such that each of the graphs can be drawn with straight lines and no crossings. Recent theoretical results [5, 10] imply that simultaneous embeddings exist only for special cases and relaxations of the problem (such as the one we address in this paper) should be considered. Along these lines, Collberg *et al* [7] describe a graph-based system for visualization of software evolution, which uses a modification of our algorithm for visualization of large graphs [12], while preserving the mental map by fixing the locations of all common vertices in the evolving graph.

3 Modified Force-Directed Method

We first review the basic force-directed graph layout algorithm and then describe the modifications for node-weighted and edge-weighted graphs. The modified force-directed algorithm is used in all three layout algorithms.

A standard force-directed layout algorithmbegins with an initial random placement of the vertices. Then it iteratively computes the effect of repulsive/attractive forces on vertices and updates the temperature. The temperature controls the scale of each iteration. At the beginning the temperature is high and vertices move significant distances and with time, the temperature is decreased. The attractive and repulsive forces are defined as follows:

$$f_r(d) = -\kappa^2/d \qquad f_a(d) = d^2/\kappa,$$

where d is the distance between two vertices. The repulsive forces are calculated for each pair of vertices whereas the attractive forces are calculated for pairs of vertices connected by an edge. The ideal distance between vertices, κ, is is defined as follows:

$$\kappa = C\sqrt{A_{frame}/n},$$

where A_{frame} is the area of the frame, C is a constant determining how the vertices fill the frame, and n is the total number of vertices.

Given a series of graphs $G_1, G_2, \ldots, G_k$, we create one node-weighted and edge-weighted *aggregate graph* $G_A = (V_A, E_A)$. A node $v \in V_A$ has weight w if it appears in w of the graphs in the series. Similarly, an edge $(u, v) \in E_A$ has weight proportional to the number of times edge (u, v) appears in the series. We use the node and edge weights to modify the standard force-directed algorithm as follows. If vertex v has large weight (it appears in many graphs) then it should to be placed close to the center in the final layout. If an edge (u, v) has large weight then the vertices u and v should be placed very close to each other in the final layout. This is a simple heuristic, but it ensures that:

– persistent vertices remain close to the center of the layout, while fleeting vertices appear and disappear on the periphery;

– vertices that are adjacent in many of the graphs in the series are placed close together.

In order to handle the vertex weights we place a dummy vertex in the center of the frame and ensure that it attracts all the other vertices in proportion to their weights. We formulate this new central attraction force as:

$$f_{ca}(d) = d^2 \times w/\kappa,$$

where w is the weight of the vertex and d is its distance from the center. To handle edge weights we scale the attractive forces by their edge weights and the new formulation of the attractive forces becomes:

$$f_a(d) = d^2 \times w_e/\kappa,$$

where w_e is the weight of the edge e.

4 Layout Algorithms and Visualization Schemes

Depending on mainly two factors, the number of graphs to be embedded simultaneously and how similar the individual graphs are, different layout methods and visualization techniques arise. If there are not too many graphs to be embedded and the graphs share a reasonably large common substructure, then a layout method that embeds common vertices of each individual graph at exactly the same locations and the common edges in a similar manner is preferable. In terms of visualization, it might be more advantageous to view the graphs on the same plane. However, if there are many graphs to be embedded, or if the individual graphs do not share many common substructures, then more flexible embeddings might be more visually appealing. In such cases, we do not insist on exactly the same locations for shared vertices of different graphs but rather try to locate them in close proximity, so that the mental map of the viewer is somewhat preserved. Not insisting on the exact same location for same vertices, allows for more freedom to draw each graph with higher readability. In terms of visualization, having each graph laid out on a separate 2D plane or morphing between consecutive 3D drawings seems most suitable.

Based on these observations we describe three different layout methods: *aggregate graph layout*, *merged graph layout* and *converging iterations layout*. After describing each layout method, we present a matching visualization scheme that seems most appropriate for it: *aggregate view*, *merged view* and *split view*. While the three visualization schemes closely correspond to their matching algorithms, different combinations of layout and visualization algorithms can also be used.

In the aggregate graph layout method we begin by creating the node-weighted and edge weighted graph $G_A = (V_A, E_A)$ from the graph sequence, $G_1, G_2, \ldots, G_k$, as described in the previous section. We then apply the modified force-directed layout algorithm to obtain a drawing for G_A. From this drawing we extract the drawings of each individual graph in the series. Thus, vertices 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, since the vertex/edge weights are taken into account in the layout of the aggregate graph, the final layout will be close to an individual layout of a graph proportional to

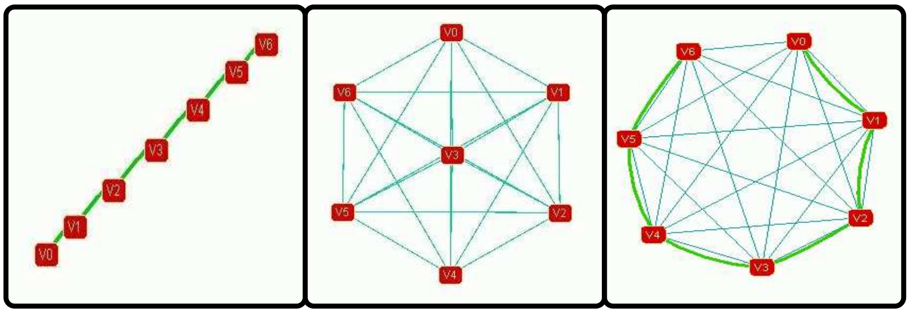


Fig. 3. Left: Individual layout of P_7 drawn with curved edges. Middle: Individual layout of K_7 drawn with straight-line segments. Right: Simultaneous embedding of P_7 and K_7 obtained from the aggregate layout method.

the importance of that one graph. In other words, if a graph G_i has many vertices/edges that exist in most of the graphs, then G_i is an important graph and the resulting layout will be similar to that of an independent layout of G_i.

4.1 Aggregate Graph Layout

Fig 3 shows the simultaneous layout of K_7, the complete graph on seven vertices and P_7, the path with seven vertices. 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 vertices in the middle, the simultaneous embedding with the aggregate layout method pushes that vertex 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 $G_A = (V_A, E_A)$:
  $V_A = V_1 \cup V_2 \cup \ldots \cup V_k,\ E_A = E_1 \cup E_2 \cup \ldots \cup E_k$
2 Assign weights to each $u \in V_A$ and $(u, v) \in E_A$:
  $w(u)$ = number of appearances of u 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_A
4 Extract the layout of each G_i from the layout of G_A

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

Aggregate View: The matching visualization scheme for the aggregate graph layout is the aggregate view. In this scheme we only display a vertex once, even though it may be 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 vertex set allows the viewer to see multiple graphs at the same time and view the difference in relationships more easily. Different edge colors ands 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 tone of green.

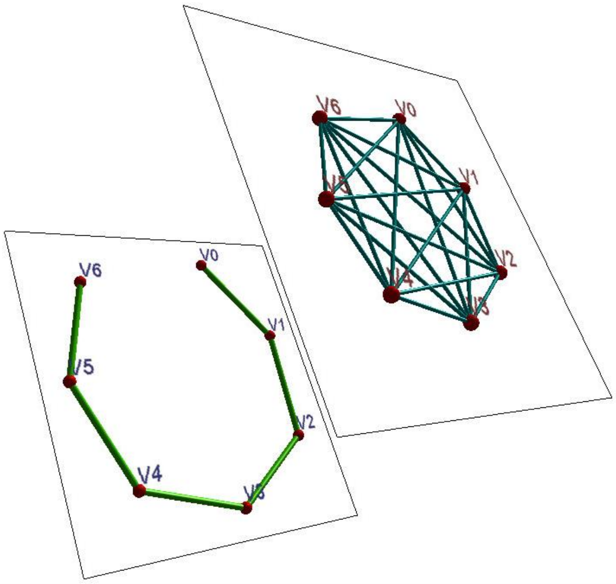


Fig. 5. Simultaneous embedding of K_7 and P_7 using the merged graph layout method. The visualization is done in 3D using a separate plane for each graph.

4.2 Merged Graph Layout

In contrast to the aggregate method, the merged graph layout method does not guarantee perfect mental map preservation. The algorithm begins with the creation of a merged graph $G_M = (V_M, E_M)$ from the given sequence of graphs $G_1, G_2, \ldots, G_k$. The merged graph is obtained by taking $G_1, G_2, \ldots G_k$ and inter-connecting all corresponding vertices with a special class of edges, E_{new}. Thus, if a vertex v appears m times in the sequences, there will be m copies of it in G_k.

The positions of corresponding vertices 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 vertices in each separate layout will be. An important property of this layout method is the proximity of corresponding vertices in the final layout. Let $u_1, u_2, \ldots u_j$ be all the vertices corresponding to u in the merged graph and $v_1, v_2, \ldots v_m$ be the ones corresponding to v. If $j > m$, then the u vertices get placed closer to each other than the v vertices do in the final layout. Once the merged graph has been created and the weights assigned, the modified force-directed method is applied.

In our implementation we allow the user to interactively assign a weight for the edges in E_{new}, so that the user has an overall control on the relative distances of corresponding vertices in different layers. Thus, in effect, the user has overall control over the extent of mental map preservation. 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 vertices might not be the same, the mental map is still preserved since the relative locations of the corresponding vertices remain the same. A summary of the merged graph layout algorithm is in Fig 6.

Merged View: The matching visualization scheme for the merged graph layout is the merged view. In this scheme each of the graphs is drawn on its separate 2D plane, and

Merged Graph Layout

1 Rename the vertices in $V_1, V_2, \ldots, V_k$ so that each vertex is unique
2 Construct E_{new} by connecting corresponding vertices in $V_1, V_2, \ldots, V_k$
3 Construct $G_M = (V_M, E_M)$:
    $V_M = V_1 \cup V_2 \cup \ldots \cup V_k$, $E_M = E_1 \cup E_2 \cup \ldots \cup E_k \cup E_{new}$
4 Assign weights for the edges in E_{new}
5 Apply the modified force-directed layout algorithm on G_M

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

the planes are layrerd in 3D in the order of appearance; see Fig. 5. At the same time, all the graph layouts are shown on the same screen and since corresponding vertices from any two planes have the same approximate positions on their planes, this model provides a clear mental mapping between the two relationships represented by each graph.

This view model also allows the user to move and rotate the planes in 3D. This feature is useful in case the user wants to see a particular graph in more detail, in which case it is sufficient to rotate the view around a particular axis. In addition, to enhance the user's 3D view, the vertices are drawn as spheres and the edges as cylindrical pipes.

4.3 Converging Iterations Layout

The two approaches defined above construct a global graph and extract individual layout of each graph from this global layout. 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 intelligent (rather than random) placement of the vertices, based on the graph distance, as described in [13]. At this stage, we have the best drawings 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 vertices 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 graph (again independently) we arrive at two new point-set 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.

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, shearing and given a point $p = (x, y)$ on the plane it can be defined as:

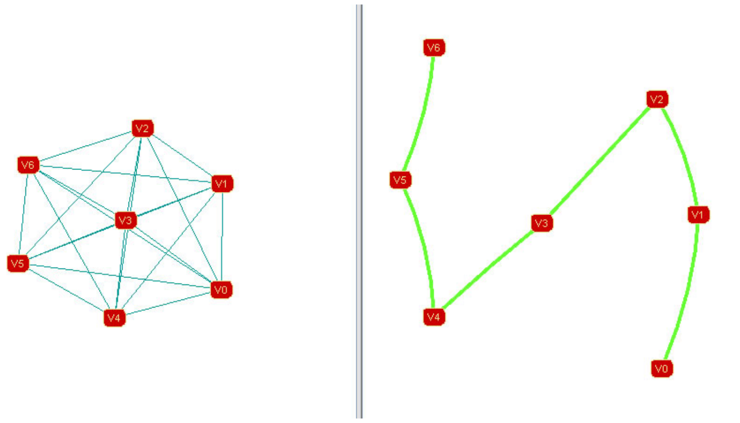


Fig. 7. Simultaneous embedding of K_7 and P_7 using converging iterations layout method and split view model for visualization.

$$f(p) = \begin{pmatrix} c_{x1} & c_{y1} \\ c_{x2} & c_{y2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} c_{x3} \\ c_{y3} \end{pmatrix}$$

We would like to find the function $f(p)$, (i.e. all the constants c_{x1} etc.) that minimizes the distance between the transformed layout of l_1 and l_2, which is then equivalent to minimizing:

$$\sum_{p \in l_1} dist(f(p), p'),$$

where p' is the point in l_2 corresponding to p and $dist(f(p), p')$ is the Euclidean distance between $f(p)$ and p'. Then the minimization can easily be achieved by taking the derivative with respect to c_{xi} and solving for the resulting linear equations. Fig 7 shows the simultaneous embedding of K_7 and P_7 resulting from converging 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 converging iterations layout is a compromise between the two individual layouts. A summary of the converging iterations graph layout algorithm is in Fig. 8.

The algorithm is well 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. Currently our implementation works for pairs of graphs only.

Split View: The two graphs are drawn separately in their own windows in 2-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 greatly decreases in this case which makes the model more suitable for visualization of small number of graphs.

Converging Iterations Layout

1 Using independent intelligent placement obtain layouts l_1 and l_2 for G_1 and 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 $mindist < threshold$:

  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 $l_1 = l_{1'}$, $l_2 = l_{2'}$

Fig. 8. Converging Iterations Layout. G_1 and G_2 are the input graphs

5 Implementation

We have implemented our layout methods and visualization schemes using Java and the system can be downloaded at `http://simg.cs.arizona.edu/`. Fig. 9 shows a snapshot of our system. In addition to the three layout methods and three visualization schemes, the system provides various capabilities such as graph editing, building some common classes of graphs (complete graphs, trees, paths), building random graphs, etc. Graphs in GML format can be loaded and stored. All images in the paper (except that in Fig. 2) are from our system. In Fig. 10 we show more layouts obtained with our system.

References

1. *Vertex Splitting and Tension-Free Layout*, volume 1027 of *Lecture Notes in Computer Science*. Springer, 1996.
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. P. Brass, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. B. Mitchell. On simultaneous graph embedding. In *8th Workshop on Algorithms and Data Structures*. To appear in 2003.
6. R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis. Dynamic graph drawings: Trees, series-parallel digraphs, and planar *ST*-digraphs. *SIAM J. Comput.*, 24(5):970–1001, 1995.
7. 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*. To appear in 2003.
8. S. Diehl and C. Görg. Graphs, they are changing. In *Proceedings of the 10th Symposium on Graph Drawing (GD)*, pages 23–30, 2002.
9. P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
10. C. Erten and S. G. Kobourov. Simultaneous embedding of a planar graph and its dual on the grid. In *13th Intl. Symp. on Algorithms and Computation (ISAAC)*, pages 575–587, 2002.

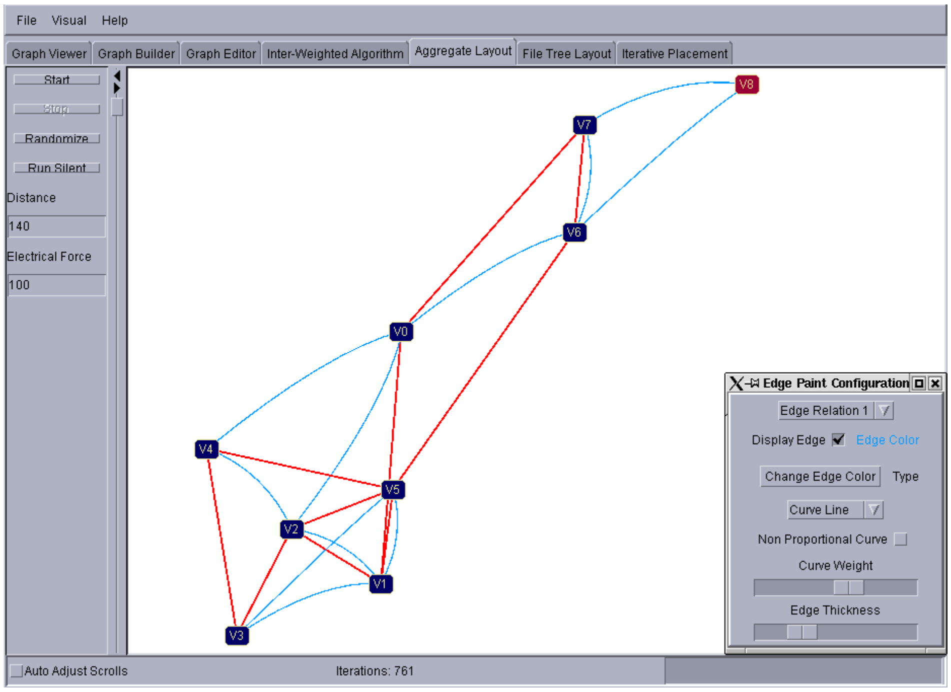


Fig. 9. Snapshot of our simultaneous graph drawing system.

11. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Softw. – Pract. Exp.*, 21(11):1129–1164, 1991.
12. P. Gajer, M. T. Goodrich, and S. G. Kobourov. A multi-dimensional approach to force-directed layouts. In *Proceedings of the 8th Symposium on Graph Drawing (GD)*, pages 211–221, 2000.
13. P. Gajer and S. G. Kobourov. GRIP: Graph dRawing with Intelligent Placement. *Journal of Graph Algorithms and Applications*, 6(3):203–224, 2002.
14. D. Harel and Y. Koren. Drawing graphs with non-uniform vertices. pages 157–166, 2002.
15. D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *Journal of graph algorithms and applications*, 6:179–202, 2002.
16. 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. 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.
18. 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.
19. S. Moen. Drawing dynamic trees. *IEEE Software*, 7(4):21–28, July 1990.
20. S. C. North. Incremental layout in DynaDAG. In *Proceedings of the 4th Symposium on Graph Drawing (GD)*, pages 409–418, 1996.
21. 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.

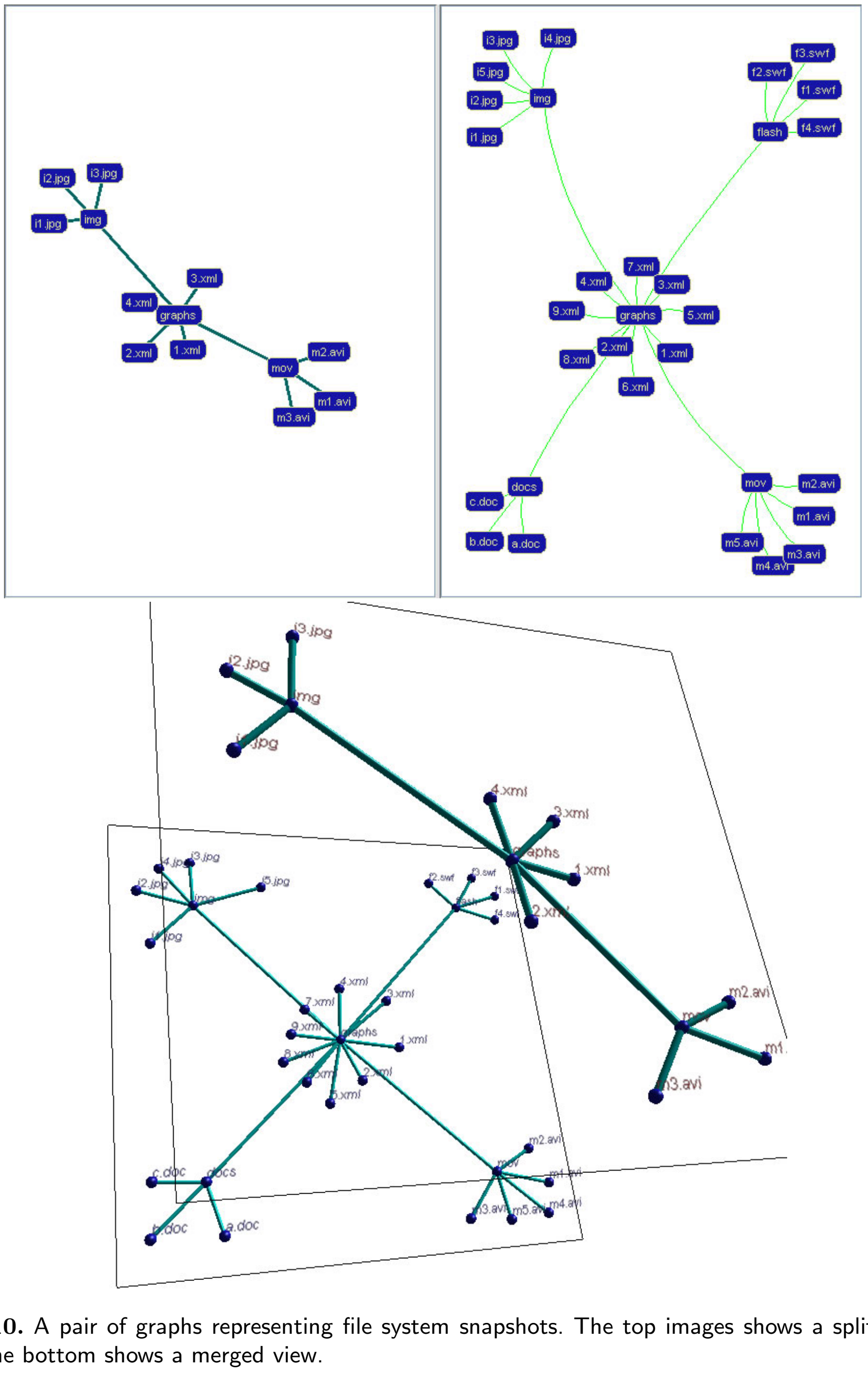


Fig. 10. A pair of graphs representing file system snapshots. The top images shows a split view and the bottom shows a merged view.