G-Space: A Linear Time Graph Layout

Brian Wylie [a], Jeffrey Baumes [b], and Timothy M. Shead [a*]

[a] Sandia National Laboratories, Albuquerque, NM, USA;
[b] Kitware, Inc., Clifton Park, NY, USA

ABSTRACT

We describe G-Space, a straightforward linear time layout algorithm that draws undirected graphs based purely on their topological features. The algorithm is divided into two phases. The first phase is an embedding of the graph into a 2-D plane using the graph-theoretical distances as coordinates. These coordinates are computed with the same process used by HDE (High-Dimensional Embedding) algorithms. In our case we do a Low-Dimensional Embedding (LDE), and directly map the graph distances into a two dimensional geometric space. The second phase is the resolution of the many-to-one mappings that frequently occur within the low dimensional embedding. The resulting layout appears to have advantages over existing methods: it can be computed rapidly, and it can be used to answer topological questions quickly and intuitively.

Keywords: Graph layout, graph drawing, graph visualization, graphs and networks, information visualization

1. INTRODUCTION

Graph drawing is an active area of research and has been well studied for many years. There are many approaches to graph layout including force-directed, multi-level hierarchies, High Dimensional Embedding (HDE), and topological feature-based methods. Further, combinations of these techniques have been demonstrated, including the excellent visual results of [1].

Our proposed approach to graph layout is primarily motivated by the desire to provide a "real time" layout capability for fluid user interaction and rapid feedback. A specific use case we wish to support is the rapid layout of graphs produced by interactive database queries, such as "display all the e-mail communication in the Enron database between September and December". Terabyte-sized relational databases are now commonplace, and the number of graphs implicitly embedded in the data can be enormous. The user will likely make many iterative database queries and for graph layout to be useful in this context the result set must be laid out and displayed within seconds.

A second motivation is to provide more explicit topological information to the user. Traditional layouts of highly connected graphs often obfuscate the answers to straightforward questions such as:

- Which vertices have a direct connection to a query vertex?
- How many "hops" (topologically) does it take to get from one vertex to another?
- Is a vertex "closer" (graph-theoretical distance) to one vertex or another?
- What is the shortest path between two vertices, or the *N* shortest paths?

Of course, in the real world the questions asked will be more specific: "Show me anyone with either direct or indirect contact with Person *X* in the last 30 days".

We believe our approach successfully satisfies both motivations and conveys clear topological information about complex graphs in linear time. We discuss the specific details of the G-Space layout, and compare both the visual quality and performance of G-Space with other popular layout algorithms. As with any layout algorithm G-Space has its limitations, particularly the many-to-one mappings that occur within the LDE, so we discuss the approaches used to mitigate those limitations.

---

* Further author information:
Brian Wylie: E-mail: bnwylie@sandia.gov
Timothy M. Shead: E-mail: tshead@sandia.gov
Jeffrey Baumes: E-mail: jeff.baumes@kitware.com

The G-Space layout algorithm makes extensive use of the Titan Informatics Toolkit [2]. Titan is a Sandia National Laboratories project that combines information visualization with scientific visualization within the open source VTK framework.

2. RELATED WORK

Force-directed layouts have become the mainstay of graph layout algorithms [3]. The force-directed algorithm simulates the forces of a system where adjacent vertices are connected with springs, and the energy of the system is decreased over time until the system stabilizes. Although it often produces pleasing results on small graphs, simple force-directed placement is often impractical for large graphs due to its algorithmic complexity. Also, given the nature of the vertex placement, force-directed layout can be difficult to interpret. Notably, the areas of greatest interest (high vertex or edge concentration) are often the least interpretable. In spite of its shortcomings, the force-directed approach has been the foundation on which many graph layout algorithms are built.

Graph Drawing with Intelligent Placement (GRIP) is an algorithm based on the force-directed approach [4]. GRIP first produces a sequence of vertex sets with decreasing size from V (the set of all vertices) to V_k, where $|V_k| = 3$. The set V_k is placed deterministically based on the relative positions among the three vertices. GRIP places additional vertices in V_{k-1} based on their positions relative to each other then performs a force-directed layout on the added vertices. This process continues until all vertices have been placed. This algorithm has efficient runtimes in practice.

The high-dimensional embedding (HDE) algorithm is a recent advancement in graph layout strategies [5]. HDE first extracts M graph vertices with high distance from each other. The vertices are given an M-dimensional coordinate, where each coordinate value represents the distance from one vertex to another. The algorithm extracts the principal components of this high-dimensional space in order to reduce the dimensionality to two or three dimensions for placing the vertices. HDE is a very fast algorithm whose core involves only M linear breadth-first searches. A disadvantage of HDE is the tendency to produce long, thin layouts, particularly when applied to trees or tree-like graphs [1].

The Algebraic multigrid Computation of Eigenvectors (ACE) layout algorithm relies heavily on matrix computations to produce the layout [6]. This procedure iteratively coarsens the graph, and then projects the graph using the eigenvectors of the graph's Laplacian representation. The eigenvectors of the coarse graph are used as the initial solutions to the coarser graph, resulting in rapid convergence.

FM3 is another algorithm based on force-directed layout [7]. The algorithm reduces the quadratic nature of the force-directed technique by coarsening the graph into a hierarchy of fixed-diameter graphs. During refinement, the repulsive forces, which normally take quadratic time to compute, instead are computed in O(N log N) time with an efficient quad tree structure.

TopoLayout is a new layout approach which combines many of the algorithms described above, along with simple tree and clique layout strategies [1]. Using detection algorithms including connected components, tree, and clique detection, TopoLayout is able to classify regions of the graph and apply the best layout algorithm to each region.

A related problem to graph layout is general data embedding, or dimensionality reduction. Multi-dimensional scaling (MDS) seeks to find the embedding in k dimensions where distances in the space correspond most closely to measured distances. Finding the optimal placement has a high complexity, so researchers have sought ways to efficiently approximate the algorithm. Algorithms termed Landmark MDS and Pivot MDS achieve this by performing MDS on a subset of the data objects [11, 12]. Remaining objects are placed relative to these landmark (or pivot) objects.

FastMap has a similar approach, but iteratively uses pairs of pivots instead of defining all pivots at once [13]. It finds a pair of distant objects (the pivots) and embeds the rest of the objects in relation to the distance between them, placing everything on the line between the pivots. It then repeats this process (picking two new pivots) for each of the k dimensions in the embedding. G-Space is in some ways similar to a one-dimensional FastMap. However, instead of placing all objects along a line in one dimension, it expands this into two dimensions, disambiguating the actual distance to each pivot.

3. APPROACH

Our graph layout approach appears to be suitable for analyzing graph structures on multiple scales under various use cases; it can be used to visualize extremely large graphs representing a database of relationships in its entirety, or it can be used to show the results of a targeted point-to-point query (a "Kevin Bacon" query). The first phase of our approach was originally inspired by the point-to-point case so we'll begin our explanation there.

3.1 LDE (Low Dimensional Embedding)

Clearly, LDE is a play on the term HDE (High Dimensional Embedding) and we adopted it in homage of the work done by [5]. When we started our work, we were not yet familiar with HDE and were only interested in two dimensional embeddings. Our work was motivated by the following use case: given a database containing a large social network, a user wishes to see the "meta-relationship" (shortest path, types of relationships along the shortest path, identified critical links, etc) between two individuals. Thus the LDE process is essentially HDE with two pivot points. Assume the graph shown in Figure 1:

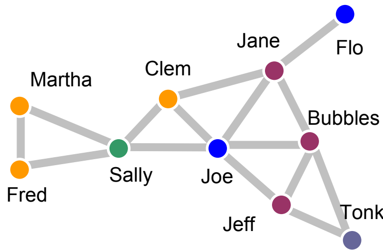

Figure 1. Example email network.

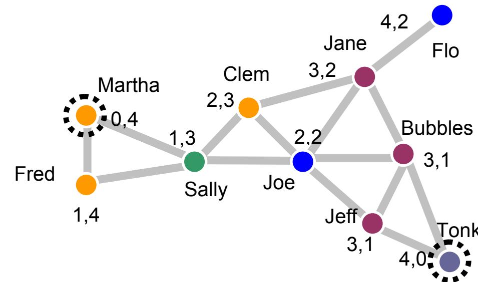

Figure 2. Giving coordinates to the graph vertices based on their shortest path distance from pivot points.

The user wishes to display the meta-relationship between two vertices "Martha" and "Tonk". The LDE process selects those two vertices as pivot points and conducts a breadth-first search from each. Every vertex now has an associated two-tuple containing its shortest-path distance from each of the two pivot points. Using these tuples we simply map each vertex into two dimensional geometric space as shown in Figure 3:

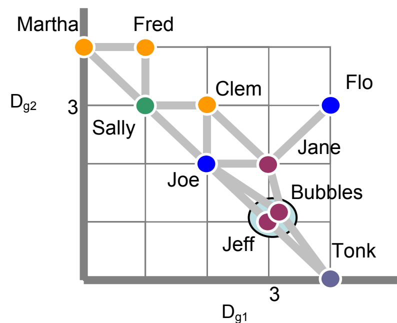

Figure 3. Direct mapping of graph distance (length of the un-weighted shortest-path) as coordinates into two dimensional space.

You can see from Figures 2 and 3 that the "Jeff" and "Bubbles" vertices both have distance coordinates of 3,1 from the pivot points (a many-to-one mapping in the embedding process). Where HDE resolves these many-to-one mappings using higher dimensions and principle components analysis (PCA), LDE accepts the many-to-one mappings and resolves them using a different technique (see section 3.3). The advantage to this approach is that there is no need to run 50 or 100 breadth-first searches (BFS), embed the graph into a 100 dimensional space, do a PCA, compute projections with maximal variance and then project down to two dimensions.

The simplicity of the low dimensional embedding exposes many interesting geometric properties within the layout, as shown in Figures 4 – 7:

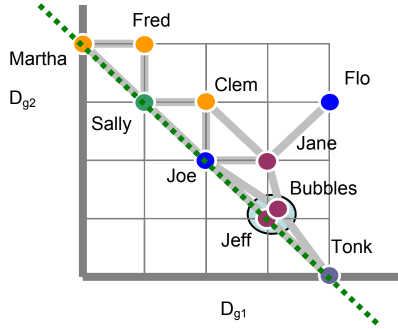

Figure 4. The embedding based on graph-theoretical distance ensures that the shortest path between the pivot points is guaranteed to be along the dashed green line. Longer paths form "arcs" into the positive quadrant.

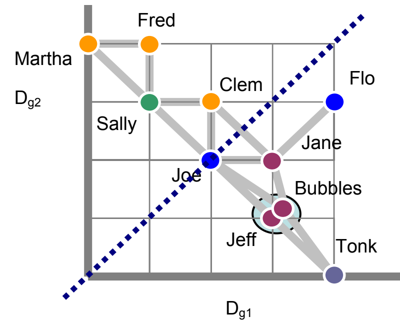

Figure 5. The dark blue line represents shortest path equidistance between the pivot points. "Flo" has a shorter path to "Tonk" than to "Martha" simply based on this geometric property.

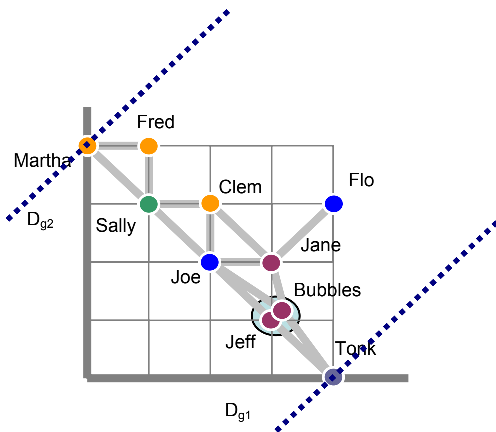

Figure 6. The distance from a vertex to a pivot vertex can never be greater than its distance to the other pivot vertex + the shortest path between the pivots. Thus, the two dark blue lines mark the "boundaries" of the layout.

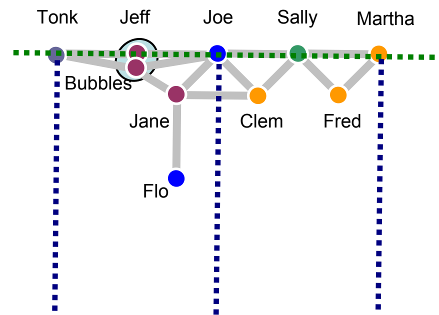

Figure 7. If the diagram is rotated 135 degrees clockwise, the shortest path will be along the top and equidistance is the vertical line in the center.

3.2 Generalizing the LDE Process

The above examples use a point-to-point query where the two pivot points for the LDE were specified by the user. As mentioned in the HDE work, pivot points can be automatically computed. In our case, we run a total of three BFS searches. The first BFS begins at a random vertex within the graph, finds a "pseudo-peripheral"† vertex, and passes that vertex as the starting-point for the second BFS. The second BFS provides vertices with the first component of their graph distance tuple, and identifies a second pseudo-peripheral vertex. This vertex becomes the starting-point for the third BFS, which provides each vertex with the second component of the distance tuple. The running time of this phase is O(V+E).

In our testing the algorithm appeared fairly insensitive to the particular choice of peripheral vertices; initially we had been more formal about the pivot choices. In fact, the normal procedure for finding pseudo-peripheral vertices is to conduct a series of BFS passes until the passes give convergence on the two pivot points. Figure 8 demonstrates the layout resulting from the generalized LDE process with automatic pivot calculation:

† Peripheral vertices have a shortest path equal to the diameter of the graph (the longest shortest path); *pseudo-peripheral* vertices have a long short path but are not guaranteed to have the longest.

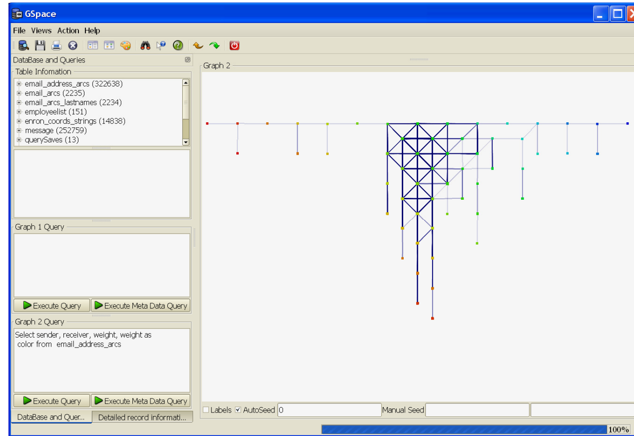

Figure 8: Automatic LDE layout of the Enron email database (322k edges, 75k vertices) [8].

3.3 Resolving the Many-To-One Mappings

As we investigated the resulting LDE layouts there were immediate reservations about their limited "resolution". Although the layouts were intriguing, they contained a high percentage of vertices which had "collided" into the same distance "bins". A large graph (50k vertices) with a graph diameter of 6 will have at most 36 "bins" in which to map the vertices, drastically limiting the value of placing the graph into a two dimensional geometric space. We needed a systematic way to resolve the many-to-one mappings that occur with LDE.

An initial attempt was made to use the LDE mapping as the starting condition for a force-directed layout, which would push/pull the vertices apart to achieve finer resolution. The resulting layouts did appear to benefit slightly from a reduction of large local minima; however the space-filling nature of the repulsive forces made the final layout relatively indistinguishable from more traditional force-directed layouts. Although this particular test bore no fruit, it did lead to the observation that filling all available space is not in-and-of-itself a useful goal. In fact, packing vertices with similar attributes together can provide users with additional insight into their shared natures.

3.3.1 Vertex Bundling

Inspired by the terrific edge bundling technique of [9], we adopted the term "Vertex Bundling" to describe how vertices can be packed together based on their connectivity attributes. Vertices have edge "obligations" to other vertices; vertices that share similar obligations can be bundled together to minimize the space they consume. Looking at the small graph in Figure 9, we see that traditional force-directed layout pushes vertices apart to fill the available space. We suggest the opposite approach, grouping the vertices together into "semantic bundles" which, like the edge bundling technique, simplify the layout and bring clarity to the topological relationships within the graph.

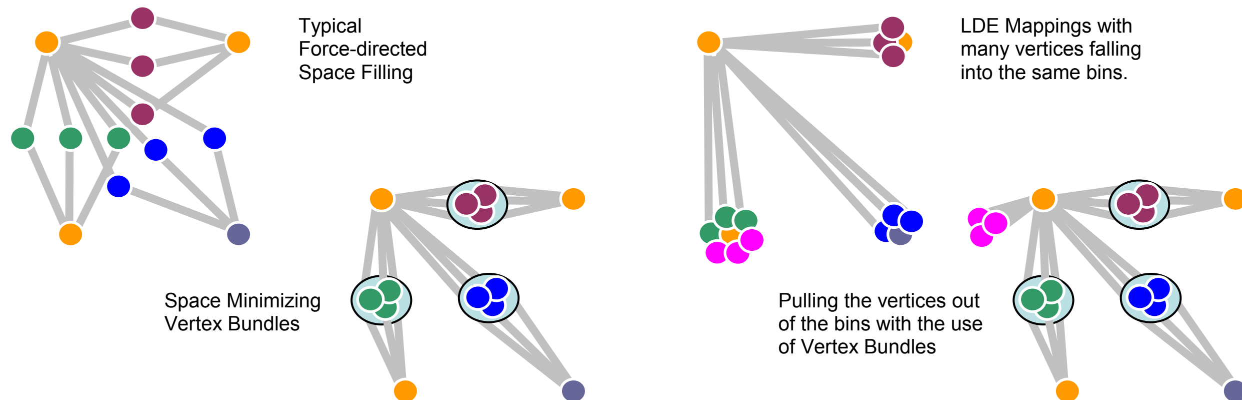

Figure 9. Simplifying graph layout, and clarifying topological relationships with the use of vertex bundles.

Figure 10. Applying the vertex bundling technique to pull vertices out of the many-to-one bins that occur with LDE.

Using the technique of vertex bundling we can now work to resolve the many-to-one mappings that occur in the LDE of the graph into two dimensional space.

3.3.2 Vertex Bundle Maps

On a large scale, vertex bundling can be used to resolve a significant portion of our many-to-one mapping problem. We now use the problematic distance bins as an ally to create a "scaffolding" of control points. Each vertex has an edge to one of more other vertices, at this point all vertices lie within a bin (control point), so we simply traverse the vertex list, determining which vertices have edges to which control points and bundle all vertices with edges to the same control points.

After the graph has been through the LDE phase and looks like the layout shown on the left side of Figure 11, we now pass the graph through the vertex bundling process and in a manner similar to marching cubes, each vertex has its edges tested against a case table of control points to see which vertex bundle it will be a member of. The entire graph is processed from first vertex to last, and depending on which case is "matched" the vertex is offset some relative amount from its current position. The running time of this phase is O(V+E).

Currently we call out 14 different cases (and 11 additional sub-cases) that are split out from the LDE bins (see Figure 12). The 25 defined cases are as follows. For each vertex the following are determined:

1. Edges only go to vertices in one bin: ***case 0***
2. Edges go to vertices of two bins: ***cases 1 – 11***
3. If a vertex falls within case 1-11 and also has edges back to the main bin then they are placed very close to the bundle but biased toward the main bin (see inset Figure 13): ***cases 12 – 22***
4. If a vertex does not meet any of these 23 cases, but has one or more edge connections within the same bin the vertex stays in the bin: ***case 23***
5. If none of these apply then the vertex is marked as an "***Unresolved Vertex***" (see Section 3.4).

The diagram in Figure 12, at first looks oddly biased towards the bottom left corner. When splitting out the vertex bundles an even distribution seems more effective, until the realization that these bundle maps are applied at every bin within the LDE layout. So the diagram will need to mesh well when placed next to, above, and below the neighbour bins who are also applying the diagram to their vertices.

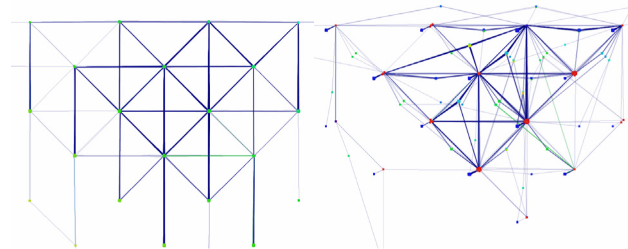

Figure 11. On the left an LDE layout of a subset of the Enron database (1374 Vertices, 2241 Edges). On the right the same layout after the Vertex "Bundling" pass. Vertex Bundling significantly mitigates the many-to-one mappings and helps convey topological information.

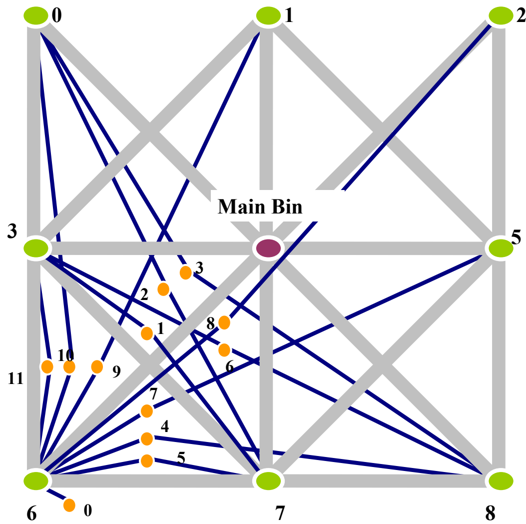

Figure 12. Vertex Bundle Map. For each distance bin created by the LDE, the above map is applied to its vertices.

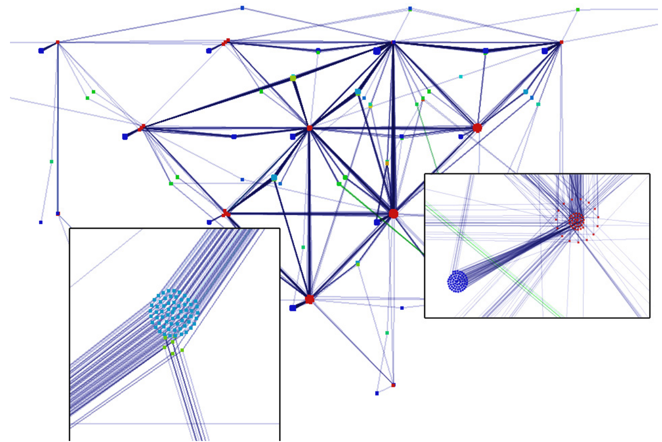

Figure 13. Enlarged image of Figure 10 with insets showing layout detail.

3.4 Unresolved Vertices

As described in section 3.3.2, there are currently 24 cases where the vertices are split out to specific locations based on topological obligations. If a vertex does not match any of the cases in the vertex bundle table, that may indicate a "problem" with the layout. In practice we find that synthetic graphs with regular topology can be difficult for G-Space to resolve. Many example datasets which are variations of 2D grids tend to not fare well in G-Space.

However we do understand the importance of user notification/feedback when potential problems with the layout may exist, so unresolved vertices are specifically marked, accumulated, and displayed to the user as areas of interest they may want to investigate further. All figures in the results section of this paper include the percentage of unresolved vertices within that particular layout.

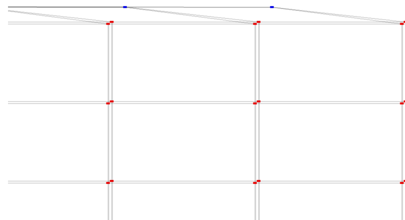

Figure 14: An example of unresolved vertices (in red) on a 2D grid which folds over on itself.

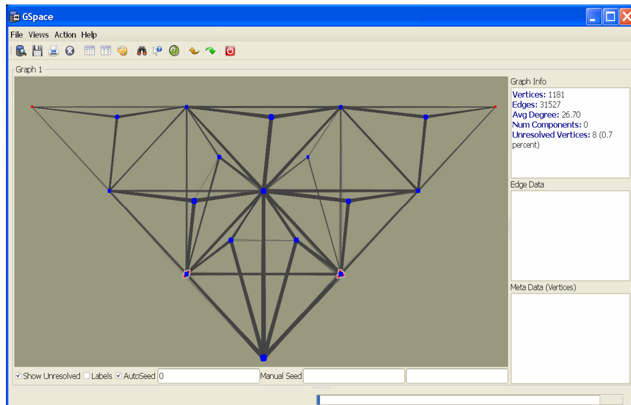

Figure 15: IMDB 1999 (N=1181 E=31,527): In this IMDB dataset only 8 vertices were categorized as "unresolved" out of 1181 total.

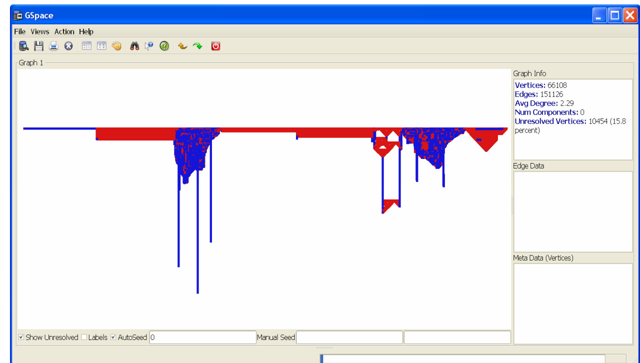

Figure 16: bi_walsh(N=77,251 E=183,945): G-Space has marked areas in red where the 2D grids in this dataset fold over on themselves (10454 unresolved vertices).

3.5 Current Issues with G-Space Layout

As we mentioned G-Space does not fare well when processing input graphs with well structured topology (grids, trees, etc). The BFS searches are trying to impose structure on the layout, and if the graph already has a well defined structure the result may be a layout such as the one in Figure 16.

The issue with unresolved vertices in itself is not that horrible; the vertices are tracked, marked, and highlighted so that the user can see them. By far, the current worst unaddressed issue is the direct descendants of those unresolved vertices. As demonstrated in Figure 17, the unresolved vertices (4 red vertices top right) all have children that are not connected anywhere else so those children are mistakenly placed together and not marked as unresolved. Obviously the layout is misleading to the user and we hope to address this case in the near future.

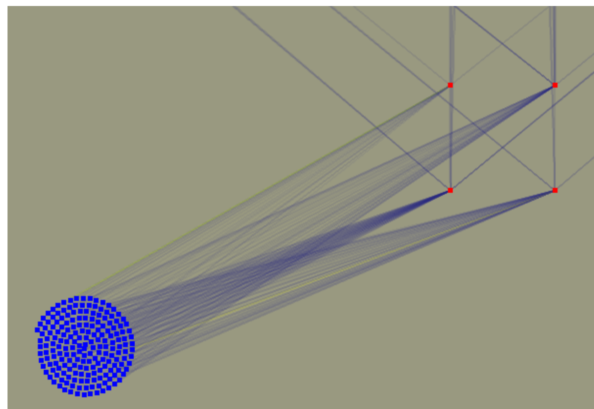

Figure 17. A dark corner in G-Space. Unresolved vertices can have their children mistakenly placed together.

4. RESULTS

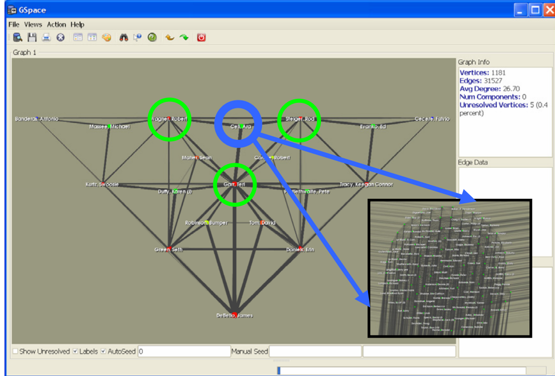

Figure 18: IMDB 1999 (N=1181 E=31,527): The dataset consists of all actors who are at most 2 hops from the actor Jake Gyllenhaal in the movie *October Sky*. Jake is in the cluster surrounded by the blue circle.

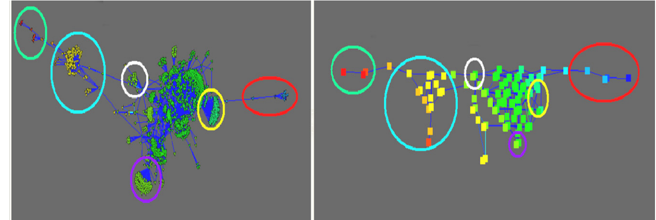

Figure 19: Correlation of a traditional force-directed layout (left) to a G-Space layout (right). Color matched circles represent similar sets of vertices between the two layouts. Data: subset of Enron database (1374 Vertices, 2241 Edges).

The IMDB 1999 dataset provides an excellent demonstration of the G-Space layout. This graph consists of all actors who are at most 2 hops from the actor Jake Gyllenhaal in the movie *October Sky*. Two actors are linked if they were in a movie or TV program together that was released in 1999. G-Space places Jake in the blue cluster, and we can readily see that the clusters circled in green are one "hop" away, while every other cluster is one hop away from those (feel free to count "hops" in Figure 18 to confirm this). The small diameter of this graph is obvious in the G-Space layout, unlike other layout algorithms. Vertex bundles make it easy to see the groups of actors, count the hops between them, and quickly understand the global topological features of a graph.

In Figure 19, we compare the correlation between a traditional force-directed layout (left) and G-Space (right), for a subset of the Enron email database (1374 Vertices, 2241 Edges). The angular lines and extremely tight clusters of the G-Space layout can give the appearance that the layout is displaying a small graph, so we used color to provide a visual "registration" with the force-directed layout. The color matched circles in Figure 19 enclose the same vertex sets in both layouts. Note that the force-directed layout was rotated and scaled to match the orientation and aspect ratio of the G-Space layout.

On a qualitative level, when exploring the two layouts side by side, with linked selection, the G-Space layout conveys better global graph structure and allows more detailed inspection of topological relationship.

Figures 20 and 21 show the results of the G-Space algorithm on a number of real-world and synthetic graphs. For a description of these graphs, see [1] and [10]. All runtimes were under a third of a second, and the layout was performed more quickly than the other algorithms in all cases tested. Note that the G-Space experiments were run on a different machine than that used in the TopoLayout paper, so G-Space times should be only roughly compared against the other runtimes.

Both the Spider and Flower layouts suffer from the limitation of only using distance information from two vertices. Since these datasets contain many long tendrils, all but two of the tendrils will have correlated distance coordinates and be placed in similar locations. In the Flower dataset, two of the "petals" of the flower protrude from the center to the left and right, while the other petals are merged into a vertical line. A similar situation occurs in the Spider dataset.

All layout experiments were run on a laptop with an Intel Dual Core 2.0GHz processor, 2GB RAM, running Windows XP.

5. CONCLUSIONS AND FUTURE WORK

We have shown that the G-Space algorithm is an efficient method for laying out a graph without the use of force-directed placement. G-Space is a useful technique where the topological relationships between vertices can be seen quickly and intuitively. Through the use of vertex bundles, we are able to resolve many of the many-to-one mapping issues inherent

in the low dimensional embedding. The speed and clarity of G-Space is particularly useful in scenarios where an analyst is conducting interactive queries of a large database, requiring rapid, interactive layout.

There are a number of areas in which G-Space can be improved. We would like to decrease the number of unresolved vertices in the diagram, to separate more vertices from the main bins. This would involve determining the common topological structures that cause vertices to be placed in the main bin, and pulling these vertices out into separate bundles in a meaningful way.

Sub-trees are a particular nuisance to the algorithm since they fall along vertical lines, instead of being spread out. By identifying treelike structures within the graph, we may layout trees using a standard tree layout algorithm. These tree views could be accessed by either zooming in on the tree, or by clicking on specialized glyphs which brings up a separate tree view.

G-Space should also be improved in order to make better use of screen space. Currently, vertex bundles are small in order to ensure that they will not overlap with other bundles, assuming that all types of bundles may exist. It would be reasonable to spread out vertex bundles when we know there are no other bundles in close proximity. Further, the roughly-triangular layouts produced by G-Space could be modified to more evenly fill a rectangular viewing area.

ACKNOWLEDGMENTS

We would like to thank Tamara Munzner and Daniel Archambault who developed the TopoLayout algorithm and have run extensive layout experiments. The TopoLayout team have been extremely helpful in making their algorithms, graph data sources, and results available, allowing us to do much of our comparative analysis.

Funding was provided by the Accelerated Strategic Computing Initiative's Visual Interactive Environment for Weapons Simulations (ASCI/VIEWS) program. The work was performed at Sandia National Laboratories. Sandia is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

REFERENCES

1. D. Archambault, T. Munzner, D. Auber. "TopoLayout: multi-level graph layout by topological features." IEEE Transactions on Visualization and Computer Graphics, 13(2):305-317, 2006.
2. B. Wylie and J. Baumes. "The Titan Informatics Toolkit." Not yet published.
3. T. Fruchterman and E. Reingold. "Graph drawing by force-directed placement." Softw. Pract. Exp., 21(11):1129-1164, 1991.
4. P. Gajer and S. G. Kobourov. "GRIP: Graph drawing with intelligent placement." Journal of Graph Algorithms and Applications, 6(3):203-224, 2002.
5. Y. Koren and D. Harel. "Graph drawing by high-dimensional embedding." In Proc. Graph Drawing (GD'02), volume 2528 of LNCS, pages 207-219, 2002.
6. Y. Koren, L. Carmel, and D. Harel. "Drawing huge graphs by algebraic multigrid optimization." Multiscale Modeling and Simulation, 1(4):645-673, 2003.
7. S. Hachul and M. Jünger. "Drawing large graphs with a potential-field-based multilevel algorithm." In Proc. 12th Int. Symp. on Graph Drawing, volume 3383 of LNCS, pages 285-295. Springer-Verlag, 2004.
8. J. Shetty and J. Adibi. The Enron Email Dataset Database Schema and Brief Statistical Report. Available online at http://www.isi.edu/~adibi/Enron/Enron_Dataset_Report.pdf. Accessed March 30, 2007.
9. D. Holten. "Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data." IEEE Transactions on Visualization and Computer Graphics, 12(5):741-748, 2006.
10. S. Hachul and M. Jünger. "An experimental comparison of fast algorithms for drawing general large graphs." In Proc. 13th Int. Symp. on Graph Drawing. Springer-Verlag, 2005.
11. V. de Silva, J. B. Tenenbaum. "Sparse multidimensional scaling using landmark points" (Technical Report). Stanford University.
12. A. Morrison, G. Ross, M. Chalmers. "Fast multidimensional scaling through sampling, springs and interpolation". Information Visualization 2(1) March 2003, pp. 68-77.
13. C. Faloutsos, K.-I. Lin. FastMap: "A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets." In Proceedings of 1995 ACM SIGMOD, SIGMOD RECORD (June 1995), vol.24, no.2, p 163-174.

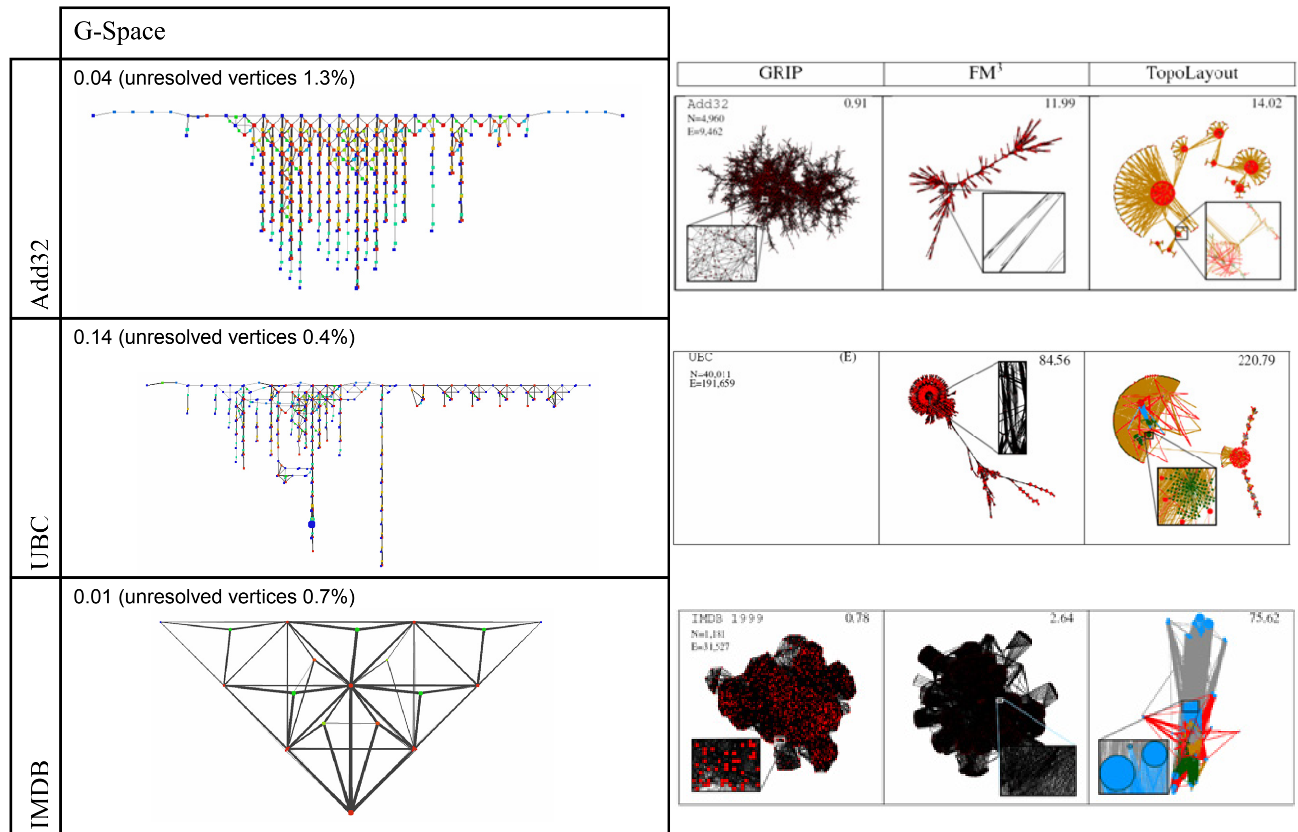

Figure 20: Tests on real-world data sets. The runtime is given for each case in seconds. Note that the runtimes between G-Space and the other algorithms are only a rough comparison since they were run on different machines (see [1]). The unresolved vertex percentage is the percentage of vertices which had no type, and had no connections to other vertices in its group.

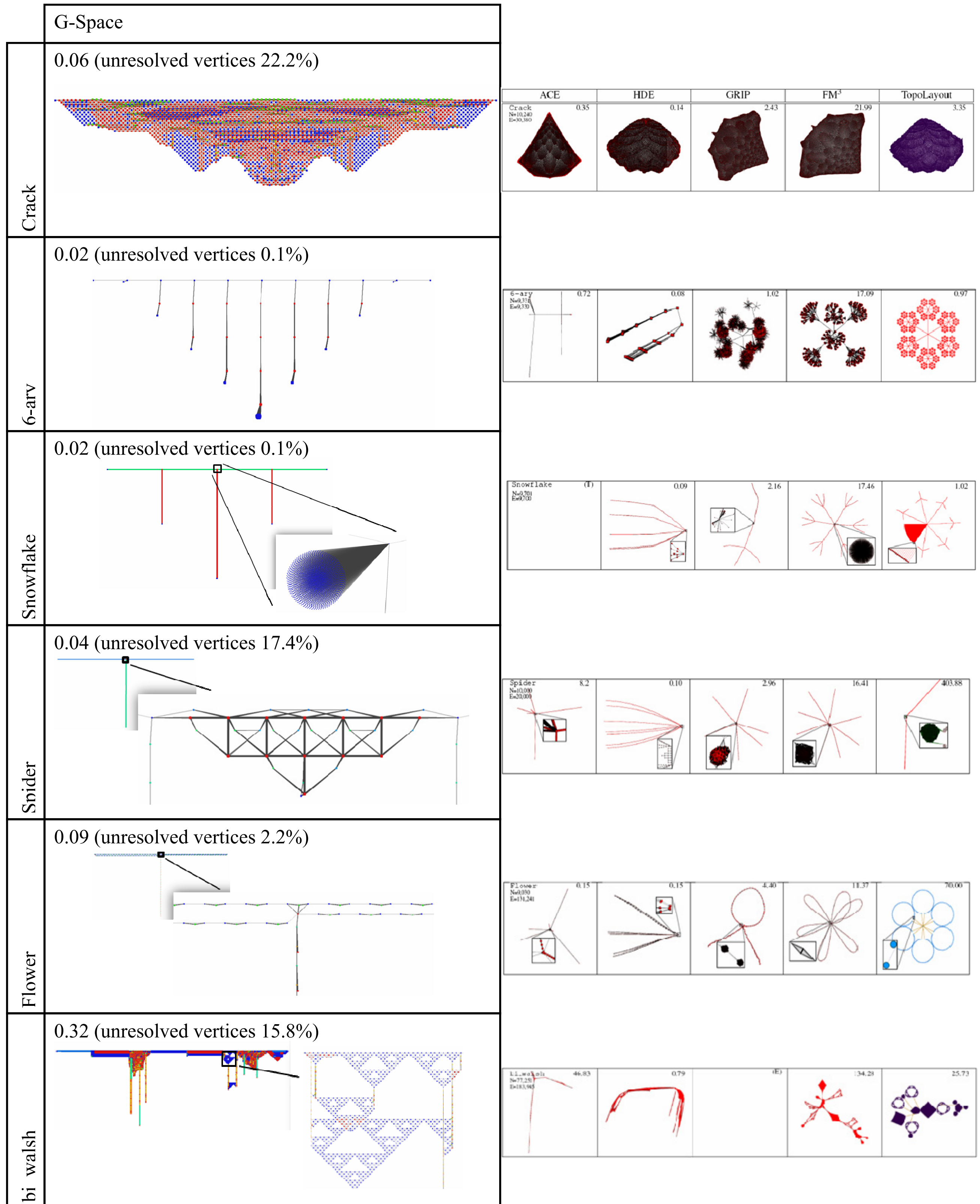

Figure 21: Tests on synthetic data sets. The runtime is given for each case in seconds. All layouts using algorithms other than G-Space were conducted in [1]. Note that the runtimes are only a rough comparison since G-Space was run on a different machine. The unresolved vertex percentage represents those vertices that were not bundled and did not have connections to other vertices within the group.