TopoLayout: Graph Layout by Topological Features

Daniel Archambault* University of British Columbia
Tamara Munzner* University of British Columbia
David Auber† Université de Bordeaux I

ABSTRACT

We describe a new multi-level algorithm to draw graphs based on the topological features they contain. Topological features are recursively detected and their subgraphs are collapsed into single nodes, forming a graph hierarchy. Once the hierarchy is computed, we draw the subgraphs of the hierarchy, using an appropriate algorithm for each topological feature. Our layout algorithms are area-aware: the space required to draw a topological feature is taken into account when the node representing that feature is drawn at a higher level of the hierarchy. Unlike previous work, TopoLayout can be geared to graphs that contain specific topological features to produce layouts that emphasize those features without asymptotic or empirical runtime penalty.

1 INTRODUCTION

In this work, we address the problem of finding good two dimensional drawings of undirected graphs. Our approach is **multi-level**: we decompose the graph into a hierarchy of subgraphs of decreasing size, and lay out the graph using the hierarchy. Like all multi-level algorithms, our approach allows us to use layout methods of high complexity, which would be too expensive for the entire graph, that yield quality drawings of the smaller subgraphs of the hierarchy. In TopoLayout, we partition the graph into **topological features** or **features** such as trees, connected components, and biconnected components. We lay out each feature with an algorithm tuned for its topology, exploiting the wealth of research in the graph drawing literature. Nodes in our graph hierarchy have explicit screen-space extents determined by the layout of the features they contain. Thus, all our layout methods should be made **area-aware**. Our approach introduces a node-edge overlaps reduction algorithm, and we employ multiple passes to prevent node-node overlaps and improve the information density of the final layout. We have implemented TopoLayout using the Tulip [1] framework.

2 MULTI-LEVEL APPROACHES

There have been a number of hierarchical methods developed to help improve algorithm runtime without loss, or perhaps even improvement, of layout quality. Generally, these algorithms recursively apply a coarsening operator to the graph, creating a graph hierarchy where coarser graphs in the hierarchy are representative of the more detailed ones, but are cheaper to lay out. Many of these approaches exist based on an estimates of maximal matching [13], graph filtration based on shortest path distance [5], and eigenvector computation on coarse approximations of the Laplacian matrix [8]. It is important to note that these approaches treat all nodes in the graph the same and that topological features in these graphs may not be apparent. In contrast, the work of Six and Tollis [10] non-recursively decomposes the graph into biconnected graphs and lay out the biconnected components using a radial tree layout algorithm that is area-aware. This work is similar to our own, but TopoLayout detects a larger number of topological features.

*{archam, tmm}@cs.ubc.ca
†auber@labri.fr

3 ALGORITHM

The TopoLayout algorithm consists of four main phases: a **decomposition** phase creates the hierarchy; a **layout** phase finds an initial two-dimensional embedding for each topological feature; a **reduce crossings** phase which reduces, but does not completely eliminate, node-edge and edge-edge crossings by rotating interior nodes; an **overlap** phase, where bounding objects for the subgraphs are computed and node-node overlaps are resolved by shrinking nodes; and a final **compaction** phase where nodes are pulled toward the centre of the drawing at each level, improving the information density of the layout.

3.1 Decomposition

When the coarsening operator detects a topological feature, the feature subgraph is collapsed into a single node and, depending on the feature identified, the coarsening operator is recursively applied to the resulting graph and subgraph. We detect **connected components** using a series of depth-first searches which span the nodes of each of the connected components in the graph. **Trees** are detected by finding the first cycle in the graph and selecting a node n on that cycle. If a cycle is not found, the entire graph is a tree. Otherwise, starting at n, we perform a depth first search on the entire graph and iteratively remove all nodes of degree one. When no more nodes of degree one are present a maximal perfect tree is detected and the graph proceeds with the depth first search. We use a standard algorithm for **biconnected component** detection described in the textbook of Baase and Van Gelder [3]. Although **near-meshes** are not soundly defined topological features, as the Embedder algorithm performs well on them, we detect them by computing statistics on the variance of node degree across the graph and ensuring the absence of high degree nodes. Finally, we compute **small world clusters** using the strength metric of Auber *et al.* [2].

3.1.1 Hierarchy

Figure 1 shows an example hierarchy that includes all five topological feature types we detect. The coarsening operator creates a hierarchy of topological features where the original nodes of the input graph are the **leaves** and the **interior nodes** contain topological features composed of leaves and other interior nodes.

3.2 Lay Out Features

The initial layout of the topological features in the graph depends on the detected topological type. We employ three types of layout algorithms: tree, circular, and force-directed.

Area-aware **tree layout** algorithms are employed for the tree and biconnected component topological types. The biconnected components are emphasized using one of the methods suggested by Six and Tollis [10] by placing the biconnecting node between the two components. Any area-aware tree layout algorithm can be used in our framework. TopoLayout currently uses [6, 12] for its trees. We

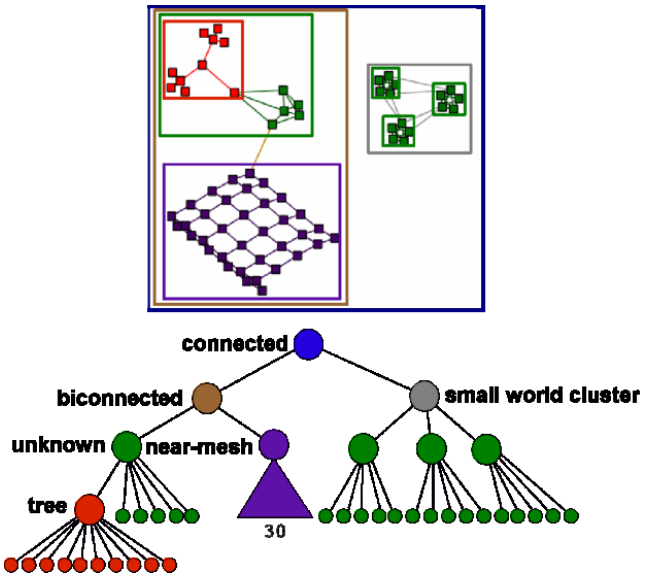

Figure 1: Subgraph hierarchy after decomposition, with topology encoded by colour. **Top**: Layout annotated with boxes to show hierarchy structure. **Bottom**: Diagram of subgraph hierarchy, with nodes labeled by feature type.

use an area-aware **circular layout** algorithm both for near-complete subgraphs and for large clusters or unknown components; the current threshold for large size is set to 1000 nodes in the subgraph. We define a **near-complete** subgraph as having a high node-to-edge ratio. Circular layout consists of simply placing the nodes of the graph around a circle, so area-aware circular layout is a straightforward adaption. We use **area-aware GEM** for all other cases: clusters and unknown components that are less than 1000 nodes and are not near-complete. We introduce area-aware GEM, which is similar to the algorithm developed by Harel and Koren [7], but we have adapted the GEM algorithm [4]. We use the Embedder algorithm [9] to lay out near-mesh subgraphs. We note Embedder is not area-aware.

3.3 Reduce Crossings

Our reduce crossings phase is similar to that of Symeonidis and Tollis [11], but it is applied using a force model with some oscillation control. For a leaf node l contained inside an interior node i with an edge e that leaves i, a torque τ is applied to i with magnitude:

$$\tau = \frac{\pi}{2}\mathrm{sg}(\vec{r}\times\vec{f})(\vec{r}\cdot\vec{f})$$

The vector $\vec{f}$ is a unit vector along e and $\vec{r}$ is the distance between l and the centre of i. The function $\mathrm{sg}(\vec{x})$ returns the sign of the torque perpendicular to the xy-plane. Each interior node i is iteratively rotated by its average τ. Oscillation control is done by damping average torques that change sign between iterations.

3.4 Resolve Overlaps and Compaction

Node-node overlaps are checked by ensuring that no two nodes at any level in the hierarchy overlap. Typically, there are few nodes in a level of the hierarchy, and the $O(|N_l|^2)$ work can be done quickly. If two nodes overlap, the minimum shrinking factor to resolve the overlap is computed. After checking all pairs of nodes the maximum of the minimum shrink factors of each node is applied to ensure that there are no overlaps. Compaction is done by computing the maximum shrink factor that can be applied about the centre of the drawing before node overlaps are incurred. This is done by computing the maximum shrinking factor that can be applied to any pair of nodes along the straight line between them. The minimum of these maximums is applied to the entire graph about its centre.

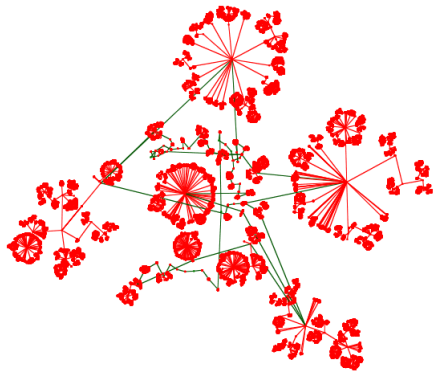

Figure 2: Layout of near spanning tree, with cycles shown in green, of ICMP trace of the main servers on the Internet backbone. Data from Feb 26 2002, courtesy of Bill Cheswick.

References

[1] D. Auber. Tulip : A huge graph visualisation framework. In P. Mutzel and M. Jünger, editors, *Graph Drawing Software*, Mathematics and Visualization, pages 105–126. Springer-Verlag, 2003.

[2] D. Auber, Y. Chiricota, F. Jourdan, and G. Melancon. Multiscale visualization of small world networks. In *Proc. IEEE Symposium on Information Visualization (InfoVis'03)*, pages 75–81, 2003.

[3] S. Baase and A. Van Gelder. *Computer Algorithms: Introduction to Design and Analysis*, chapter 7.7.2: The Bicomponent Algorithm, pages 368–373. Addison-Wesley, 3rd edition, 2000.

[4] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Proc. Graph Drawing (GD'94)*, volume 894 of *LNCS*, pages 388–403, 1995.

[5] P. Gajer, M. Goodrich, and S. Kobourov. A multi-dimensional approach to force-directed layouts of large graphs. In *Proc. Graph Drawing (GD'00)*, volume 1984 of *LNCS*, pages 211–221. Springer, Berlin, 2001.

[6] S. Grivet, D. Auber, J. Domenger, and G. Melancon. Bubble tree drawing algorithm. In *nternational Conference on Computer Vision and Graphics*, pages 633–641, 2004.

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

[8] Y. Koren, L. Carmel, and D. Harel. Ace: A fast multiscale eigenvectors computation for drawing huge graphs. In *Proc. IEEE Symposium on Information Visualization (InfoVis'02)*, pages 137–144, 2002.

[9] Y. Koren and D. Harel. Graph drawing by high-dimensional embedding. In *Proc. Graph Drawing (GD'02)*, pages 207–219, 2002.

[10] J. M. Six and I. G. Tollis. A framework for circular drawings of networks. In *Proc. Graph Drawing (GD'99)*, volume 1731 of *LNCS*, pages 107–116. Springer, Berlin, 1999.

[11] A. Symeonidis and I. G. Tollis. Visualization of biological information with circular drawings. In *Intl Symposium on Medical Data Analysis (ISBMDA)*, pages 468–478, 2004.

[12] J. Q. Walker. A node positioning algorithm for general trees. *Software - Practice and Experience*, 20(7):685–705, July 1990.

[13] C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *Proc. Graph Drawing (GD'00)*, volume 1984 of *LNCS*, pages 171–182. Springer, Berlin, 2001.