Circular Layout in the Graph Layout Toolkit

Uğur Doğrusöz, Brendan Madden, Patrick Madden

Tom Sawyer Software, 804 Hearst Avenue, Berkeley, CA 94710
info@tomsawyer.com
http://www.tomsawyer.com

Abstract. The *Graph Layout Toolkit* is a family of portable, automated, graph layout libraries designed for integration into graphical user interface application programs. The *Circular Library* is one of the four styles currently available with the Graph Layout Toolkit. It produces layouts that emphasize natural group structures inherent in a graph's topology, and is well suited for the layout of ring and star network topologies. It clusters (groups) the nodes of a graph by group IDs, by IP addresses, and by biconnectivity or node degree, and allows the user to specify a range for the size of each cluster. The Library positions the nodes of a cluster on a radiating circle, and employs heuristics to reduce the crossings not only between edges incident to nodes of the same cluster but also between edges that connect different clusters.

1 Introduction

Graph layout is the automatic positioning of the nodes and the edges of a graph in order to produce an aesthetically pleasing drawing of the graph that is easy to comprehend. This is very important for visualization tools in numerous areas such as project management, software development, database design, and network management.

Many graph layout and editing systems have been developed in the past. Please refer to [DETT95] for an overview of such systems. The *Graph Layout Toolkit* [MMPH95, GLT96a, GLT96b] is a family of graph layout libraries with ANSI C++ and C APIs that facilitate easy integration with graphical user interface programs for development of graph visualization and editing tools.

Graph layout comes in different flavors, each being more suitable for a different area. The Graph Layout Toolkit offers four different layout styles: *Circular*, *Hierarchical*, *Orthogonal*, and *Symmetric*.

The Circular Library uses robust techniques such as clustering by biconnectivity and methods for minimizing the cut when a cluster needs to split. In addition, it can provide application oriented partitioning. It also shows good performance on the important aesthetic layout criterion of low number of crossings. In the following sections, we describe the methods used by the Circular Library. In particular, we focus on clustering, positioning, and crossing minimization techniques.

2 The Circular Library

The layout algorithm of the Circular Library is primarily designed for the layout of ring and star network topologies. It is an advanced version of the one developed by Kar, Madden, and Gilbert [KMG89]. It functions by partitioning the nodes of a graph into logical groups (clusters) based on a number of flexible grouping methods. These clusters are placed on radiating circles based on their logical interconnection. Part of the graph of clusters is laid out on a circle. A virtual cluster is used to manage this cluster of clusters. The remaining parts of the cluster graph (subgraphs that form trees), on the other hand, are laid out using the Hierarchical Library, and attach to the virtual cluster from their roots.

The clustering is either performed with a generic method, such as the biconnectivity of the graph or the degrees of the nodes, or can be performed with IP addresses, IP subnet masks or another domain specific technique. Domain specific techniques clearly should not the primary focus for the development of graph drawing techniques, however, it is noteworthy that domain specific clustering can be easily provided in certain circumstances. The Library also supports manually configured clustering with the help of group IDs.

The algorithm minimizes cluster to cluster crossings as well as crossings within each cluster. In addition, it employs tree balancing routines, and has ring and star detection and placement techniques within each cluster. Figure 1 shows two sample drawings produced by the Circular Library.

3 Definitions

In this section, we give some basic definitions.

A *cluster* C is a group of nodes to be laid out together. The Circular Library lays the nodes of a cluster around a circle. We refer to the radius of the circle around which the nodes of a cluster C is positioned as the *radius of* C.

The cluster graph G^C of a graph G is a graph in which each node is uniquely associated with a cluster in G, and there is an edge in between two nodes of G^C representing clusters C_1 and C_2 of G if and only if there is at least one pair of nodes $n_1 \in C_1$ and $n_2 \in C_2$ that are adjacent in G.

For a given graph $G = (N, E)$, *reduction of trees* refers to the process of recursively removing degree one nodes along with their incident edges until no degree one node is left in the graph. All such reduced nodes form a forest. These trees can be later inserted back into the original graph by a reverse procedure. We call this *expansion of trees*.

Clusters in a cluster graph can be of two types: *sub site* clusters are those that are removed as a result of the reduction of trees of the cluster graph, whereas *main site* clusters are the ones that remain in the cluster graph after the reduction of the trees. We will refer to each reduced tree of the cluster graph as a *sub site tree*. In addition, the cluster formed from the main site clusters and their interconnections is called the *virtual* cluster. Circular positioning of the clusters in the virtual cluster forms the *backbone* of the drawing.

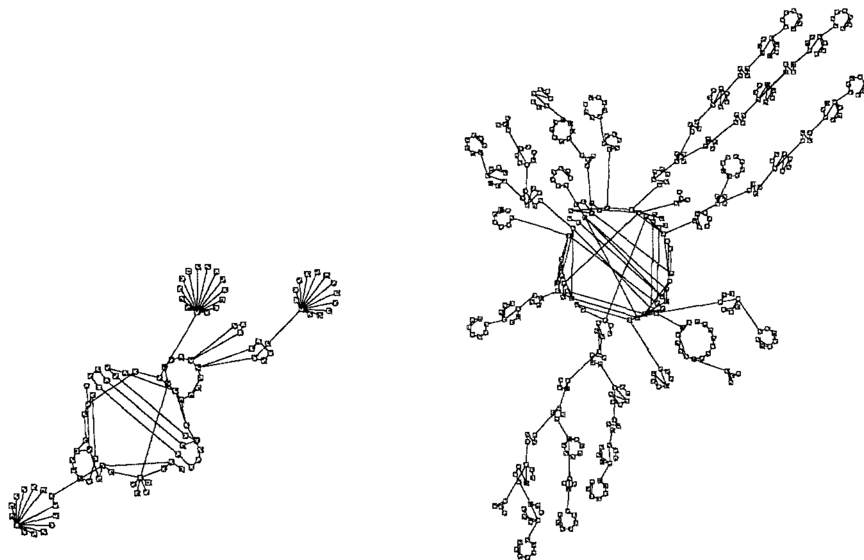

Fig. 1. Sample drawings produced by the Circular Library.

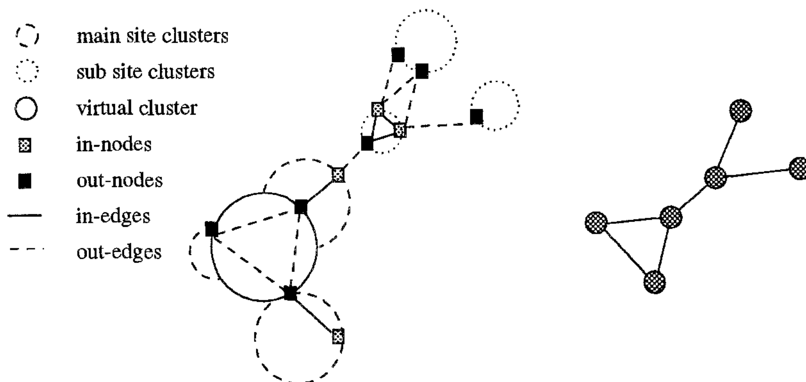

Fig. 2. A circular drawing of a graph with *only a portion* of the set of nodes and edges shown as an illustration of the definitions (left), and the corresponding cluster graph (right).

The nodes (edges) of a cluster are categorized into two: *in* and *out*-nodes (edges). An *in*-node (edge) of a cluster C is one that is connected to (connects) *only* nodes in cluster C. An *out*-node (edge) of C, on the other hand, is one that is connected to (incident to) at least one node in a cluster other than C.

A graph $G = (N, E)$ is said to be *biconnected* if there are at least two distinct paths between any pair of nodes $u, v \in N$. We use n (n_C) and e (e_C) to denote the number of nodes and edges in a graph (cluster), respectively.

The *ports* of a node help define a circular ordering for the edges that are incident to the node. These edges, when drawn, are clipped according to their port assignments.

4 Algorithm

An outline of the algorithm used in the Circular Library follows. The first phase of the algorithm partitions the nodes of a graph into clusters, each of which is to be laid out as a circle. It also gathers information to be used for the positioning of the nodes within each cluster as well as relative positioning of the clusters during the second and final phases of the algorithm.

Partition:

(1) cluster nodes
(2) **foreach** cluster C **do**
(3) calculate in and out-nodes and edges in C
(4) calculate radius of C
(5) **end foreach**
(6) **repeat**
(7) merge neighboring clusters that are too small
(8) **until** each cluster is larger than the minimum given
(9) **repeat**
(10) split clusters that are too large
(11) **until** each cluster is smaller than the maximum given
(12) construct cluster graph
(13) reduce trees of cluster graph (remaining nodes represent main site clusters and form a virtual cluster)
(14) expand trees of cluster graph (expanded nodes represent sub site clusters)

Position (draft):

(15) **foreach** cluster C **do**
(16) reduce trees of the subgraph associated with cluster C only
(17) **end foreach**
(18) order, position, and swap main site clusters on the virtual cluster
(19) **foreach** main site cluster C **do**
(20) position and swap out-nodes

(21) order, position, and swap in-nodes
(22) **end foreach**
(23) **foreach** sub site cluster C, in a pre-order traversal order **do**
(24) position and swap out-nodes
(25) order, position, and swap in-nodes
(26) assign ports for the inter-cluster edges in between C and its children clusters
(27) **end foreach**
(28) position sub site clusters with a layout of sub site tree using Hierarchical Library with port support

Position (final):
(29) **foreach** cluster C **do**
(30) expand the reduced trees of C
(31) **end foreach**
(32) **foreach** sub site cluster C **do**
(33) position in-nodes
(34) position out-nodes
(35) **end foreach**
(36) rotate the drawing according to the aspect ratio of the window in which the graph is to be drawn.

In the following sections, we discuss some parts of the algorithm in more detail that we believe deserve more attention.

4.1 Clustering

The Circular Library clusters the nodes of a given graph in three stages. First, the *group IDs* of the nodes, if any, are used; that is, two nodes with the same group ID are put in the same cluster. This gives the user a chance to manually group nodes. Then, the remaining unclustered nodes are clustered by *IP addresses* if desired. IP addresses are a very effective way to logically group network devices.[1] Finally, any unclustered node is clustered with *biconnectivity*, the default clustering method. The user has the option of using a clustering based on *node degrees* instead of biconnectivity, however. In this paper, we will concentrate on clustering by biconnectivity.

We believe that the most natural way of capturing the structure of graphs encountered in networking environments is to first find the biconnected components of the graph to be laid out. By clustering only nodes that are in the same biconnected component together, we make sure that there are at least two distinct paths in between each node pair. In a network map, this implies that the failure of no single device in a cluster will leave the entire network disconnected. This will naturally reveal the weaknesses in the design of a network topology.

[1] IP is the most widely used network protocol today; support for it was specifically requested by several of our customers.

In addition, such clustering will be reasonably "stable" over modifications to the topology of the network. For instance, the overal clustering of the graph will not change over operations such as addition of a link in between two nodes that belong to the same biconnected component cluster, or deletion of a link within a cluster that remains biconnected after the deletion. Similarly, in many cases the addition or deletion of a node or edge does not change the block tree of biconnected graphs, or causes only a minor change to the block tree, which helps with drawing stability.

Furthermore, the Circular Library allows users to control the sizes of clusters via several tailoring options. The user can specify the minimum and the maximum number of nodes that each cluster may have. This is especially useful when the underlying graph has large biconnected components.

The Library merges clusters that are too small (i.e., smaller than the minimum chosen by the user) with neighboring clusters. The merging continues until the minimum size requirement is met by all clusters.

Then, a heuristic algorithm derived from Kernighan-Lin's algorithm [KL70] is employed in order to split the clusters that are larger than the maximum specified. This algorithm minimizes the cut (number of inter-cluster connections) created by the split operation. Similarly, this process is repeated until all the clusters are within the specified range.[2]

4.2 Positioning and Crossing Minimization

In this section, we discuss the ideas for positioning clusters along with the nodes within each cluster. In addition, the methods applied to minimize the crossings inside each cluster and the crossings in between clusters are explained.

"Ordering", "positioning", and "swapping" (for crossing minimization) are the main operations used throughout the entire layout algorithm of the Circular Library. The in-nodes of a cluster are ordered such that we have a good initial positioning around the cluster's circle for crossing minimization. This is significant because the minimization algorithm works locally and settles for a local minimum. The ordering is determined based on a method that tries to find a depth-first search tree with a maximal depth. The ordering of the out-nodes of the same cluster is also taken into account to reduce the number of crossings for the edges in between in and out-nodes.

Once we have an ordering for the in-nodes, we position them around a circle spread out according to a spacing option set by the user. After that, we minimize the crossings inside a cluster by repeatedly swapping adjacent pairs of nodes as long as the swap results in a lower number of crossings. This process is repeated until there is a pass with no swaps. The number of crossings is calculated *only* for the in-edges of the two in-nodes considered. The out-nodes in a cluster are positioned and swapped similarly.

[2] Notice however that the algorithm might not always yield clusters within the desired range due to the fact that a cluster may not split if it would result in clusters that are smaller than desired.

After the clustering phase is finished, the second phase of the algorithm first reduces trees in each cluster independently. The benefit of this is two-fold. First, it speeds up the draft positioning phase since operations in this phase are performed on subgraphs with reduced trees. Secondly, it ensures that each tree is treated as a single entity and its expansion does *not* introduce any crossings.

Then, the algorithm orders, positions, and swaps the main site clusters around a main circle (backbone). For this, we use the concept of a virtual cluster whose nodes and edges correspond to the main site clusters, and the interconnections among the main site clusters, respectively. After this the number of crossings of the interconnections among the main site clusters will be minimal.

After that, the Library applies the same operations to each main site cluster followed by the application of these operations to the sub site clusters in a preorder traversal of the sub sites in the corresponding sub site tree.

The ordering of the in-nodes of each cluster help reduce the number of crossings. This is due to the fact that the longer the depth-first search tree used for ordering in-nodes of a cluster, the better chances are for a lower number of edge crossings within the cluster initially. Furthermore this depth-first search traversal yields deterministic ring drawing when a cluster is comprised of a chain of degree two nodes. In most cases, this will result in a better minima for the final number of crossings than a random initial positioning.

In addition, the order in which each sub site cluster is processed is crucial to the overall success of crossing reduction because the crossing minimization in a particular cluster assumes the stability of the positions of the nodes in its parent clusters. Another important factor in crossing reduction is the usage of ports during the layout of the cluster graph with the Hierarchical Library. In the Hierarchical Library, each node can have ports at the bottom and top (or left and right depending on the orientation of the drawing) of the rectangle representing the drawing of the node. Port support is used to reflect the order in which in-nodes connect to children clusters.

The final phase of the algorithm positions the clusters according to the coordinates calculated with a hierarchical layout of the cluster graph after the reduced trees are expanded back into each cluster in a way that does *not* introduce any crossings.

4.3 Time Complexity

Lines (1)-(8), (12)-(14), (15)-(17), (29)-(31), and (36) of the algorithm take linear time in the number of edges of the graph to execute. A variant of the KL algorithm in lines (9)-(11) is of $O(n_C \cdot logn_C)$ for each cluster C. The major operations in lines (18)-(27) and lines (32)-(35) are "order", "position", and "swap". These operations take $O(e_C)$, $O(n_C)$, and $O(n_C^2)$ time for each cluster C, respectively. The hierarchical layout algorithm used in line (28) is loosely based on [STT81] and the time it takes to execute is normally negligible since it lays out sub site trees of relatively small sizes. Overall, the layout algorithm of the Circular Library is of $O(n^2 + e)$ in the worst case. However, the algorithm performs much better on average since most these operations are applied to parts

of the graph in a divide-and-conquer fashion. Please refer to Figure 3 for some statistical results on the performance of the Library's layout algorithm.

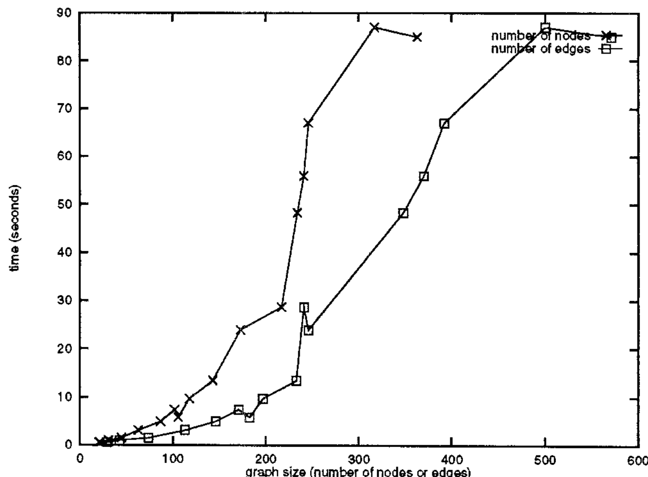

Fig. 3. The performance of the layout algorithm of the Circular Library on randomly created graphs. The tests were run on a Sun Sparc 5 workstation.

5 Conclusion

In this paper, we described some of the interesting algorithms employed by the Circular Library of the Graph Layout Toolkit, a family of portable graph layout libraries designed for integration into graphical user interface application programs.

The Library cluster by biconnectivity by default. After that we apply a variant of Kernighan-Lin's heuristic algorithm for clustering the nodes of a graph which further splits biconnected component clusters if necessary. It applies heuristic algorithms to minimize the crossings inside each cluster as well as the crossings between edges connecting two clusters.

The algorithms employed by the Circular Library take $O(n^2 + e)$ time to execute in the worst case. The average time complexity of the Library's layout algorithm, however, is much lower according to the statistical data gathered on random graphs.

One possible modification for our layout algorithm that we are currently looking into is support for multiple backbones and/or layout of the cluster graph with a spring embedder based algorithm. Both approaches seem to demand much

more complicated crossing minimization techniques. Proper handling of crossings among inter-cluster edges may yield better drawings of cluster graphs.

Acknowledgements: The authors wish to thank Dr. Ioannis Tollis for his helpful suggestions, Therese Biedl for providing us with the software for biconnectivity, and Dr. Michael Doorley for revising an earlier version of this paper.

References

[DETT95] Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: *Algorithms for drawing graphs: An annotated bibliography.* In Computational Geometry: Theory and Applications, **4**, (1994), 235–282.

[GLT96a] Tom Sawyer Software: *Graph Layout Toolkit User's Guide*, Berkeley, CA, (1992 - 1996).

[GLT96b] Tom Sawyer Software: *Graph Layout Toolkit Reference Manual*, Berkeley, CA, (1992 - 1996).

[KMG89] Kar, G., Madden, B.P., Gilbert, R.S.: *Heuristic Layout Algorithms for Network Management Presentation Services.* IEEE Network November (1988) 29–36.

[KL70] Tollis, I.G., Xia, C.: *Drawing Telecommunications Networks.* Proc. Graph Drawing '94, Lecture Notes in Computer Science 894, Springer Verlag, (1994), 206–217.

[KL70] Kernighan, B.W., Lin, S.: *An Efficient Heuristic for Partitioning Graphs.* Bell Systems Technical Journal, **49**, (1970), 291–307.

[MMPH95] Madden, B., Madden, P., Powers, S., Himsolt, M.: *Portable Graph Layout and Editing.* Proc. Graph Drawing '95, Lecture Notes in Computer Science 1027, Springer Verlag, (1995), 385–395.

[STT81] Sugiyama, K., Tagawa, S., Toda, M.: *Methods for visual understanding of hierarchical systems.* IEEE Transactions on Systems, Man and Cybernetics, **11**, (1981) 109–125.