A Framework for Circular Drawings of Networks[*]

Janet M. Six and Ioannis G. Tollis

CAD & Visualization Lab
Department of Computer Science
The University of Texas at Dallas
P.O. Box 830688, EC 31
Richardson, TX 75083-0688
{janet,tollis}@utdallas.edu

Abstract. Drawings of graphs which show the inherent strengths and weaknesses of structures with clustered views would be advantageous additions to many network design tools. In this paper we present a framework for producing circular drawings of networks represented by non-biconnected graphs. Furthermore, the drawings produced by these techniques clearly show the biconnectivity structure of the given networks. We also include results of an extensive experimental study which shows our approach to significantly outperform the current state of the art.

1 Introduction

Graphs are used to represent many kinds of information structures: computer, telecommunication, and social networks, entity-relationship diagrams, data flow charts, resource allocation maps, and much more. *Graph Drawing* researchers develop techniques which embed graphs onto a two or three-dimensional surface where nodes are represented by circles or polygons and edges by polygonal chains of line segments. The input to a graph drawing algorithm is a graph, $G = (V, E)$, where V is the set of n nodes and E is the set of m edges, while the output is a set of layout coordinates for each graph element. See [3,4] for a comprehensive annotated bibliography and introduction to the area.

A *circular graph drawing* (see Figure 1 for an example) is an embedding of a graph with the following characteristics: the graph is partitioned into clusters, the nodes of each cluster are placed onto the circumference of an individual embedding circle, and each edge is drawn as a straight line.

Tools for the design and manipulation of telecommunication networks can be strengthened with the addition of a circular drawing component which shows clustered views of those information structures. See [11] for a comprehensive discussion of telecommunication network design algorithms. The partitioning of the graph into clusters can show structural information such as biconnectivity

[*] Research supported in part by the Texas Advanced Research Program under grant number 009741-040.

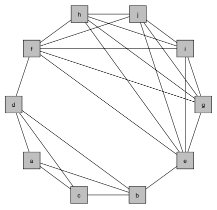

Fig. 1. A circular drawing as produced by our algorithm.

characteristics or can highlight semantic qualities of the network such as subnets. Emphasizing natural group structures within the topology of the network is vital to pin-point strengths and weaknesses within that design. It is essential that the number of crossings within the drawing of each cluster remain low and that nodes have good proximity to their neighbors. Researchers have produced several circular drawing techniques [2,6,10,12,17], some of which have been integrated into commercial tools. The resulting drawings can be compared with each other by counting the number of edge crossings. These drawings are often visually complex with respect to the number of edge crossings. In fact, the problem of producing a circular drawing with a minimum number of crossings was proven to be NP-Complete in [13]. In [15] we present a technique for producing circular drawings of biconnected graphs on a single embedding circle and an experimental study which shows the technique to perform well. A refined version of this approach is discussed in [16]. In this paper, we introduce a framework of efficient techniques to produce circular drawings of nonbiconnected networks. These algorithms require at most $O(m)$ time and have been designed to produce drawings which clearly show biconnectivity.

2 Circular Drawings of Biconnected Graphs

In this section we present a summary of an algorithm for obtaining circular drawings of biconnected networks such that all the nodes are placed onto the circumference of a single embedding circle. In addition, we include the results of an experimental study which shows this technique to be a significant improvement over the current state of the art.

In order to find a circular drawing with a lower number of crossings than previous techniques, we have developed an algorithm which tends to place edges toward the outside of the embedding circle. This characteristic means that there are not many edges in the middle of the drawing to be crossed and also that nodes are placed near their neighbors. In fact, this algorithm tries to maximize the number of edges appearing on the circumference of the embedding circle. This is achieved by selectively removing some edges and then building a DFS-based node ordering of the resulting graph. The number of crossings can then be further reduced with a postprocessing step.

In order to selectively remove some edges, Algorithm *CIRCULAR* visits the nodes in a wave-like fashion. Define a *pair edge* to be incident to two nodes which share at least one neighbor. Each time a node is processed, pair edges induced by the current node are found and placed into a removal list, a “thread” of edges is run through its neighbors to maintain biconnectivity and then the node is deleted. After all of the nodes have been processed, the graph is restored to its original topology and then the removal list edges are removed. It is this selective edge removal that causes the behavior of edge placement towards the perimeter of the embedding circle. Subsequent to the edge removal, *CIRCULAR* conducts a depth-first search (DFS) and places the nodes in a longest path of the DFS tree around the embedding circle. Finally, the remaining nodes are nicely merged into this ordering. The time complexity of *CIRCULAR* is $O(m)$ where m is the number of edges. Please see [15,16] for further details of the algorithm and its analysis. The algorithm has the following interesting property:

Theorem 1. *Given a biconnected graph, G, if G admits a circular layout with zero crossings, then CIRCULAR produces a circular drawing with zero crossings in $O(n)$ time.*

A graph G is *outerplanar* if and only if G can be drawn on the plane such that all nodes lie on the boundary of a single face and no two edges cross [9]. This fact implies that the set of biconnected graphs which admit a plane circular drawing is exactly the set of outerplanar graphs. Given an outerplanar graph, *CIRCULAR* will produce a plane circular drawing since the edge removal portion of the technique is inspired by the algorithm for recognizing outerplanar graphs in [14].

By Theorem 1, if a zero-crossing layout exists for an input biconnected graph, then *CIRCULAR* will find it, however if the input graph does not admit a plane drawing, we offer in [15,16] a heuristic technique to iteratively reduce the number of crossings. We have implemented our technique for building circular drawings of biconnected graphs, *CIRCULAR*, in C++ and run experiments over a set of 10,328 biconnected graphs from [5]. The average number of crossings in the drawings produced by our $O(m)$ time *CIRCULAR* technique is about 15% less than that of the $O(n^2)$ time GLT technique [6,10]. Performing the postprocessing step improves upon the average number of crossings of GLT drawings by 30%.

3 Circular Drawings of Nonbiconnected Graphs

Graphs are not always biconnected, therefore it is important for a graph drawing tool to provide a component which visualizes nonbiconnected graphs. In this section we present techniques for drawing nonbiconnected graphs. Furthermore, these drawings will clearly show the biconnected components.

3.1 Drawings of Nonbiconnected Graphs

Given a nonbiconnected graph, G, we can decompose the structure into biconnected components in $O(m)$ time. By definition, the biconnected components will be connected via bridges or articulation points and this superstructure will form a *block-cut point tree*. Taking advantage of this inherent structure, we first layout the biconnected components of the block-cut point tree with a radial layout technique similar to [1,7,8], then layout each biconnected component of the network with a variant of the *CIRCULAR* algorithm. See Figure 2. This technique is called *CIRCULAR - with Radial*.

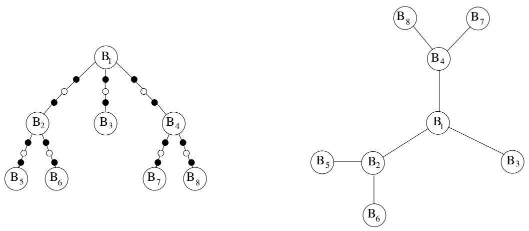

Fig. 2. The illustration on the left shows the block-cut point tree of a non-biconnected graph. The small black tree nodes represent articulation points and the small white tree nodes represent bridges. The right illustration is a drawing of the same graph where the block-cut point tree is laid out with a radial tree layout technique.

CIRCULAR - with Radial must address several issues in order to produce good quality circular drawings: articulation points can appear in multiple biconnected components of the block-cut point tree, the nodes of the block-cut point tree can represent biconnected components of differing size, and the nodes of each biconnected component should be visualized such that the articulation points appear in good position and also there is a low number of edge crossings. We will address each of these issues in turn.

Assignment of Articulation Points: Define a **strict articulation point** as an articulation point which is not adjacent to a bridge. Strict articulation points are duplicated in more than one biconnected component of the block-cut point tree, but of course each node should appear only once in a drawing of that graph. Therefore, we offer three approaches in which the articulation point will appear only once in the drawing. The first approach assigns each strict articulation point to the biconnected component which is closest to the root in the block-cut point tree. This biconnected component will be the parent of the other biconnected components, see Figure 3(a). The second approach assigns the articulation point to the biconnected component which contains the most neighbors of that articulation point, see Figure 3(b). The third approach assigns the articulation point to a position between its biconnected components, see Figure 3(c). Placing a node in this manner will highlight the fact that this node is an important articulation point. Following the assignment step, the duplicates of a strict articulation point are removed from the blocks in the block-cut point tree. We refer to the nodes adjacent to a removed strict articulation point in a biconnected component as *inter-block nodes*. In order to maintain biconnectivity for the method which will layout this biconnected component, a thread is run through the inter-block nodes. Due to space constraints, further discussion of articulation point assignment is not given here.

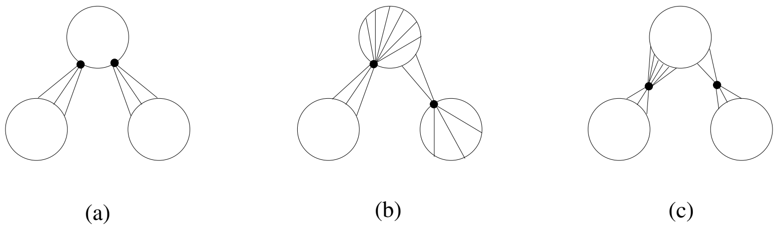

Fig. 3. This figure shows examples of three approaches for the assignment of strict articulation points to biconnected components. The black nodes are strict articulation points.

Radial Layout of Trees With Differing Node Sizes: While performing the layout of the block-cut point tree, we must consider that the biconnected components may be of differing sizes. The radial layout algorithms presented in [1,7,8] place the root at $(0, 0)$ and the subtrees on concentric circles around the origin. Also see [4] for a summary. These algorithms require linear time and produce plane drawings. However, unlike our block-cut point trees, the nodes of the trees laid out with [1,7,8] are all the same size. In order to produce radial drawings of trees with differing node sizes, we introduce *RADIAL - with Different Node Sizes*.

For each node, *RADIAL - with Different Node Sizes* must assign a ρ coordinate, which is the distance from point $(0, 0)$ to the placement of that node and a θ coordinate which is the angle between the line from $(0, 0)$ to $(\infty, 0)$ and the line from $(0, 0)$ to the placement of that node. The ρ coordinate of node v, $\rho(v)$, is defined to be $\rho(u) + \delta + \frac{d_u}{2} + \frac{max(d_1,d_2,...,d_k)}{2}$, where $\rho(u)$ is the ρ coordinate of the parent u of v, δ is the minimum distance allowed between two nodes, d_u is the diameter of u, and $max(d_1, d_2, ..., d_k)$ is the maximum of the diameters of all the children of u. In order to prevent edge crossings, each subtree must be placed inside a wedge, and the width of each wedge must be restricted such that it does not overlap a wedge of any other subtree. The θ coordinate of node v depends on the widths of the descendants of v, not just the number of leaves as in [1,7,8]. This assignment of coordinates leads to a layout of the form shown in Figure 4.

Circular Layout of a Biconnected Component: After performing *RADIAL - with Different Node Sizes* we have a layout of the block-cut point tree, and need to visualize the nodes and edges of each biconnected component. The radial layout of the block-cut point tree should be considered while drawing each

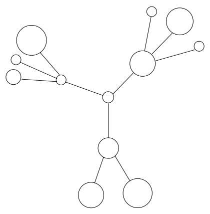

Fig. 4. This illustration demonstrates a radial layout of a tree with differing size nodes.

biconnected component, see Figure 5. In order to reduce the number of crossings caused by inter-biconnected component edges, *CIRCULAR - with Radial* tries to place nodes which are adjacent to the parent biconnected component (in the block-cut point tree) (these will be referred to as *ancestor nodes*) in the arc between α and β and the nodes which are adjacent to child biconnected components (*descendant nodes*) at points γ, δ, ϵ, etc. The size of the arc from α to β is dependent on the distance of the biconnected component to the parent in the radial layout of the block-cut point tree. The points γ, δ, and ϵ are placed in the bottom half of the biconnected component layout. These special positions for the ancestor and descendant nodes are called *ideal positions.* Placing the articulation points and inter-block nodes in this manner reduce the number of crossings caused by inter-biconnected component edges going through a biconnected component. In fact, the only times that these edges do cause crossings with this approach are when the number of ancestor or descendant nodes in the biconnected component, B_i is more than about $\frac{n_i}{2}$, where n_i is the number of nodes in B_i. In those cases, the set of ideal positions includes all the positions in the upper (respectively lower) half of the embedding circle and also positions in the lower(upper) half which are as close as possible to the upper(lower) half.

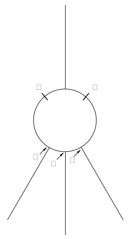

Fig. 5. This figure shows the relation between the layout of the block-cut point tree and the layout of an individual biconnected component.

We present two techniques for the layout of each biconnected component such that ideal positions of the ancestor and descendant nodes are considered. The first step of each technique is to perform *CIRCULAR* on the current biconnected component, B_i. This requires $O(m_i)$ time, where m_i is the number of edges in biconnected component B_i. Then we update this drawing such that the node placement of the articulation points is considered.

The first technique, *CLUSTER1*, rotates the layout of the biconnected component as found by *CIRCULAR* such that a maximum number of ancestor and descendant nodes are placed close to their ideal positions. Then, the remaining ancestor and descendant nodes are moved to the closest ideal position. Algorithm *Cluster1* requires $O(m_i)$ time. See Figure 6(b) for an example.

The second technique, *Cluster2* is an alternative technique which may lead to layouts with fewer edge crossings. The first steps of Algorithm *Cluster2* are the same as that of *Cluster1*. During the placement of ancestor and descendant nodes which are not in ideal positions, each such node, v, is placed in an ideal position and if the number of edge crossings added exceeds a threshold T_1 or the movement of v exceeds a threshold T_2, then the size of the embedding circle is increased such that node v can be placed in an ideal position without changing the relative order between v and its neighbors on the embedding circle. See Figure 6(c) for an example. The time required for Algorithm *Cluster2* is $O(m_i)$ if the threshold T_2 (based on node movement) is used or $O(m_i * k)$, where k is the number of ancestor and descendant nodes in the cluster, if the threshold T_1 (based on the number of crossings) is used.

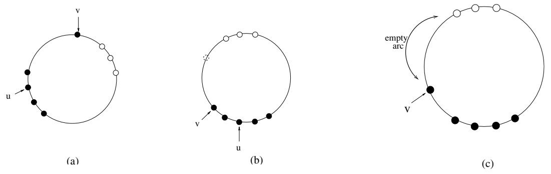

Fig. 6. This figure demonstrates Algorithms *Cluster1* and *Cluster2*. The black nodes are descendant nodes and the white nodes are ancestor nodes. (a) drawing produced by *CIRCULAR*; (b) the rotated drawing of part (a) produced by *Cluster1*. (c) the resulting drawing of part (a) produced by *Cluster2*.

3.2 Algorithm CIRCULAR - with Radial

Now we present a comprehensive technique for obtaining circular layouts of nonbiconnected graphs. First the input graph, G, is decomposed into a block-cut point tree, T. If G is not biconnected, then we layout the root cluster with *CIRCULAR* in order to produce a drawing of the root cluster with a low number of edge crossings. Next a radial layout of each subtree of the root cluster is placed in previously computed wedges around that cluster. See [16] for more details. Then drawings of each biconnected component are produced. Finally, the biconnected component drawings are translated to their final coordinates according to the radial layout of their parent subtrees.

Algorithm 31 *Algorithm* **CIRCULAR - with Radial**

Input: Any graph, G.
Output: A circular drawing of G.

```
 1. Decompose $G$ into a block-cut point tree, $T$.
 2. If $G$ has only one biconnected component
 3.           Perform CIRCULAR on $G$.
 4. Else
 5.           Assign the strict articulation points to a biconnected component.
 6.           Layout the root cluster of $T$ with CIRCULAR.
 7.           For each subtree, $S$, of the root cluster
 8.                     Perform the $\rho$ coordinate assignment phase of RADIAL -
                         with Different Node Sizes on $S$.
 9.                     For each biconnected component, $B_i$, of $S$
10.                               Layout $B_i$ with Cluster1 or Cluster2 taking into
                                   account the radii defined for the superstructure
                                   tree in Step 8.
11.                     Considering the order of the subtrees defined during the
                         layout of biconnected components in Step 10, perform
                         the $\theta$ coordinate assignment phase of RADIAL - with
                         Different Node Sizes on $S$.
12.                     Translate and rotate the clusters of $S$ according to the radial
                         layout of $S$.
```

The time complexity of *CIRCULAR - with Radial* is $O(m)$ if the biconnected components are laid out with *Cluster1* or $O(m * k)$, where k is the total number of ancestor and descendant nodes in the graph, if *Cluster2* is the algorithm chosen. Regardless of the choice of these two algorithms for the layout of each biconnected component, the biconnectivity structure is clearly shown in the drawings produced by *CIRCULAR - with Radial* since the block-cut point tree superstructure is laid out with a tree layout method that prevents the drawings of any two biconnected components from overlapping. Furthermore, the layout of each biconnected component is based on a technique which has been shown to perform very well with respect to producing drawings with a low number of crossings.

3.3 Implementation and Experiments

We have implemented *CIRCULAR - with RADIAL* with postprocessing in C++ and run preliminary experiments with 11,399 graphs from [5]. The plot in Figure 7 shows the average number of edge crossings produced by the circular layout component of Tom Sawyer Software's **Graph Layout Toolkit** (GLT), and *CIRCULAR - with Radial*. As is shown by these results, the average number of crossings in the drawings produced by our technique is about 35% less than that of the GLT technique [6,10]. Sample drawings from both the GLT and *CIRCULAR - with Radial* are shown in Figure 8.

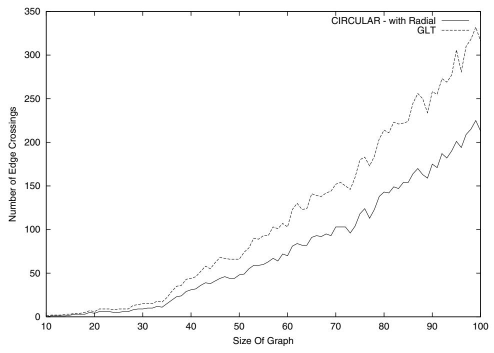

Fig. 7. This plot shows the average number of edge crossings produced by *CIRCULAR* and the Graph Layout Toolkit over 11,399 graphs from [5].

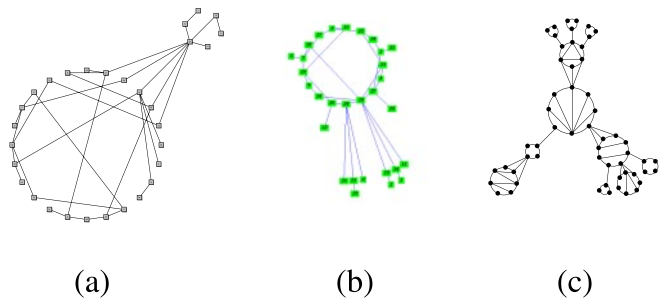

Fig. 8. Example output from the GLT and *CIRCULAR - with RADIAL*. (a) one of the graphs from [5] as drawn with the GLT; (b) drawing of the graph in (a) as produced by *CIRCULAR - with RADIAL*; (c) a sample drawing from *CIRCULAR - with RADIAL*.

Acknowledgements

We would like to thank Tom Kurien and Yong "Bruce" Zhao for helpful discussions and their skilled implementation of software components for this project.

References

1. M. A. Bernard, On the Automated Drawing of Graphs, *Proc. 3rd Caribbean Conf. on Combinatorics and Computing*, pp. 43-55, 1994.
2. F. Brandenburg, Graph Clustering 1: Cycles of Cliques, *Proc. GD '97, Rome, Italy, Lecture Notes in Computer Science 1353, Springer-Verlag*, pp. 158-168, 1998.
3. G. Di Battista, P. Eades, R. Tamassia and I. Tollis, Algorithms for Drawing Graphs: An Annotated Bibliography, *Computational Geometry: Theory and Applications*, 4(5), pp. 235-282, 1994. Also available at `http://www.utdallas.edu/∼tollis`.
4. G. Di Battista, P. Eades, R. Tamassia and I. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, Englewood Cliffs, NJ, 1999.
5. G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, F. Vargiu and L. Vismara, An Experimental Comparison of Four Graph Drawing Algorithms, *Computational Geometry: Theory and Applications*, 7(5-6), pp.303-26, 1997. Also available at `http://www.cs.brown.edu/people/rt`.
6. U. Doğrusöz, B. Madden and P. Madden, Circular Layout in the Graph Layout Toolkit, *Proc. GD '96, Berkeley, California, Lecture Notes in Computer Science 1190, Springer-Verlag*, pp. 92-100, 1997.
7. P. D. Eades, Drawing Free Trees, *Bulletin of the Institute for Combinatorics and its Applications*, 5, pp. 10-36, 1992.
8. C. Esposito, Graph Graphics: Theory and Practice, *Comput. Math. Appl.*, 15(4), pp.247-253, 1988.
9. F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
10. G. Kar, B. Madden and R. Gilbert, Heuristic Layout Algorithms for Network Presentation Services, *IEEE Network*, 11, pp. 29-36, 1988.
11. A. Kershenbaum, *Telecommunications Network Design Algorithms*, McGraw-Hill, 1993.
12. V. Krebs, Visualizing Human Networks, *Release 1.0: Esther Dyson's Monthly Report*, pp. 1-25, February 12, 1996.
13. S. Masuda, T. Kashiwabara, K. Nakajima and T. Fujisawa, On the NP-Completeness of a Computer Network Layout Problem, *Proc. IEEE 1987 International Symposium on Circuits and Systems, Philadelphia, PA*, pp.292-295, 1987.
14. S. Mitchell, Linear Algorithms to Recognize Outerplanar and Maximal Outerplanar Graphs, *Information Processing Letters*, 9(5), pp. 229-232, 1979.
15. J. M. Six and I. G. Tollis, Circular Drawings of Biconnected Graphs, *Proc. of ALENEX '99, Baltimore, MD*, To appear, 1999.
16. J. M. Six and I. G. Tollis, Algorithms for Drawing Circular Visualizations of Networks, Manuscript, 1999.
17. I. G. Tollis and C. Xia, Drawing Telecommunication Networks, *Proc. GD '94, Princeton. New Jersey, Lecture Notes in Computer Science 894, Springer-Verlag*, pp. 206-217, 1994.