International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:7, No:11, 2013

Angular-Coordinate Driven Radial Tree Drawing

Farshad Ghassemi Toosi, Nikola S. Nikolov

***Abstract*—**We present a visualization technique for radial drawing of trees consisting of two slightly different algorithms. Both of them make use of node-link diagrams for visual encoding. This visualization creates clear drawings without edge crossing. One of the algorithms is suitable for real-time visualization of large trees, as it requires minimal recalculation of the layout if leaves are inserted or removed from the tree; while the other algorithm makes better utilization of the drawing space. The algorithms are very similar and follow almost the same procedure but with different parameters. Both algorithms assign angular coordinates for all nodes which are then converted into 2D Cartesian coordinates for visualization. We present both algorithms and discuss how they compare to each other.

***Keywords*—**Radial Tree Drawing, Real-Time Visualization, Angular Coordinates, Large Trees.

I. Introduction

A tree is an undirected graph in which any two nodes are connected by exactly one simple path. In other words, any connected graph without simple cycles and loops is a tree. Trees are commonly used for representing data with a hierarchical nature and find wide application in various branches of computer science. Thus, the problem of automatic drawing of trees has attracted research attention in the last couple of decades. Many different graph drawing algorithms have been proposed so far in the research literature, and among them are also some algorithms specifically designed for drawing trees, such as the level-based algorithm [1], the radial tree drawing introduced by Book and Keshari [2] as well as the upward drawing method discussed by Alam et al. [3] which is a general graph drawing algorithm which can also be successfully used for trees.

In this work we present two different algorithms that are based on information extracted from the topology of tree. Having a rooted tree, our algorithms start with positioning the root node in the centre of the drawing and assigning all other nodes toradial layers. As a next step, angular coordinates are assigned to all nodes, again starting from root and going outwards. Finally, a *projection method* is applied to convert angular coordinates into 2D Cartesian coordinates for producing the drawing. We believe that our approach is well suited for drawing real-time trees with or without preservation of the mental map.

F. Ghassemi Toosi and N. S. Nikolov are with the Department of Computer Science and Information Systems, University of Limerick, Ireland (e-mail: Farshad.Toosi@ul.ie Nikola.Nikolov@ul.ie).

* This work is partially funded by Rayaneh-Scot-Toos [IT company] www.scotit.com.

II. Algorithms

A. Topology Extraction

Our approach makes use of information about the topology of the tree to be drawn. The final goal is to produce a radial drawing of the tree with nodes placed on concentric circles around the center of the drawing. Both algorithms do this by assigning angular coordinates to all nodes starting from the center and then progressing layer by layer outwards. The main scenario for this step is assigning a limited angular interval for each node which represents the space allowed for the direct children of that node.

If the tree is rooted then the root is placed in the center of the drawing. If the tree is undirected, and thus without a designated root node, then we can place the center of the tree (or the two centers, if the tree is bicentered) in the center of the drawing. Trees can be either *centered* or *bicentered* as shown in Fig. 1. To find the center(s), leaves are removed step by step until either one or two nodes remain. If only one node remains, that means the tree is centered; if two nodes remain then the tree is bicentered.

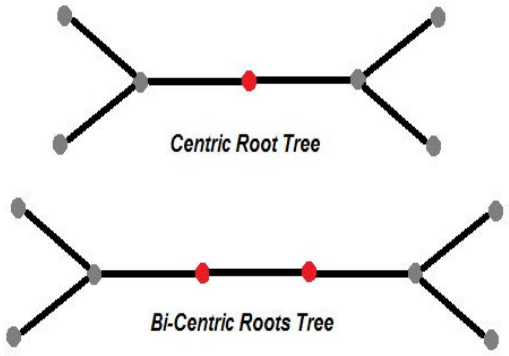

Fig. 1 An example of a centered tree with a single-node center (above) and a bicentered tree with two-node center (below)

First, the tree needs to be layered starting from its root/center/bicenter which we will call simply the *drawing center* for convenience. The drawing center is always in the first layer; its direct children become the second layer, and so on until all nodes are assigned to a layer. For example, both layouts in Fig. 1 have three layers.

B. Angular Intervals

The drawing center (in the first layer) is assigned fixed angular coordinate(s) and thus fixed angular interval(s). The main difference between the two proposed algorithms is the way we assign the angular interval to nodes. Let Ac_v denote the angular coordinate of node v, and Ai_v- the corresponding

angular interval for the direct children of node *v*. If the drawing center consists of a single node *r*, then the $Ac_r = 0$ and $Ai_r = 2\pi$.If the drawing center consists of two nodes (for un-rooted bicentered tree) *r* and *s*, then $Ac_r = 0$, $Ac_s = \pi$, and $Ai_r = Ai_s = \pi$.*Ai* for all other nodes will be computed either as:

$$Ai_v = \frac{Ai_{p(v)}}{\left|\deg_{p(v)} - 1\right|} \quad (1)$$

or as

$$Ai_v = \frac{Ai_{p(v)}}{w_{p(v)}} \times w_v, \quad (2)$$

where *p(v)* is the parent of node *v*; and $\deg_{p(v)}$ is the degree of the parent of node*v*; w_v is the weight of node*v*, which we define as the number of children and grandchildren of *v* plus 1, i.e. the number of nodes in the sub-tree with depth 2, rooted at *v*. Fig. 2 illustrates the concept of node weight.

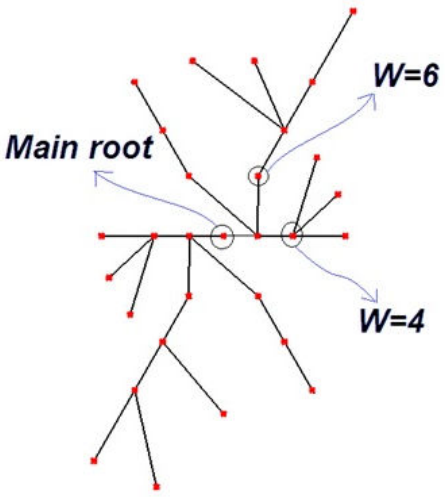

Fig. 2 Illustration of the concept of node weight. The weight w_v of node v is the number of nodes in the sub-tree with depth 2, rooted at *v*

The first algorithm we propose uses (1) for computing the space allowed space for the direct children of each node, and the second algorithm uses (2). This interval guarantees that there will be no edge-crossing since the children of each node have their own space for being accommodated.

C. Angular Coordinates

As a next step, we use the angular intervals, computed in the previous section, for assigning angular coordinates to all nodes. As mentioned above, the angular coordinate of the drawing center is either *0*(for a single-node drawing center) or *0 and π*(for bicentred trees). The angular coordinates of the rest of nodes are computed either as:

$$Ac_v = \frac{Ai_{p(v)}}{\left|\deg_{p(v)} - 1\right|} \times ind_v + Ac_{p(v)} \quad (3)$$

or as

$$Ac_v = \frac{Ai_{p(v)}}{w_{p(v)}} \times inc_v + Ac_{p(v)} \quad (4)$$

Equation (3) is computing *Ac*in our first algorithm, while (4) is used in our second algorithm. Again, *p(v)* is the parent of node v; ind_v in (3) is the index of node *v* among its siblings (assuming the siblings are ordered in no particular fashion). The function inc_v in (4) is what we call the *accumulative weight* of node *v* among its sibling. Assuming again that the children of *p(v)* are ordered somehow, the accumulative weight of the first child is *0*, and the accumulative weight of node *v* is it's the sum of weights of all its siblings placed before *v* in the order. The concept of accumulative weight is illustrated in Fig. 3.

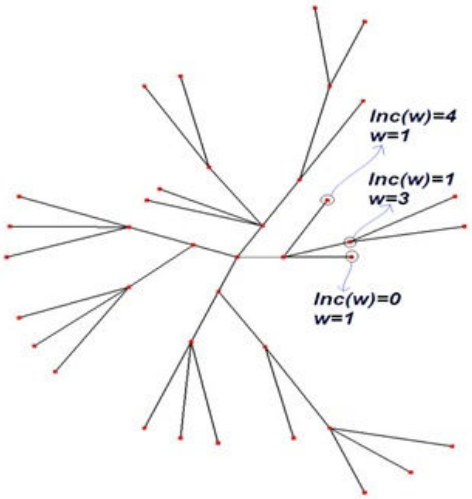

Fig. 3 Illustration of the concept accumulative weight

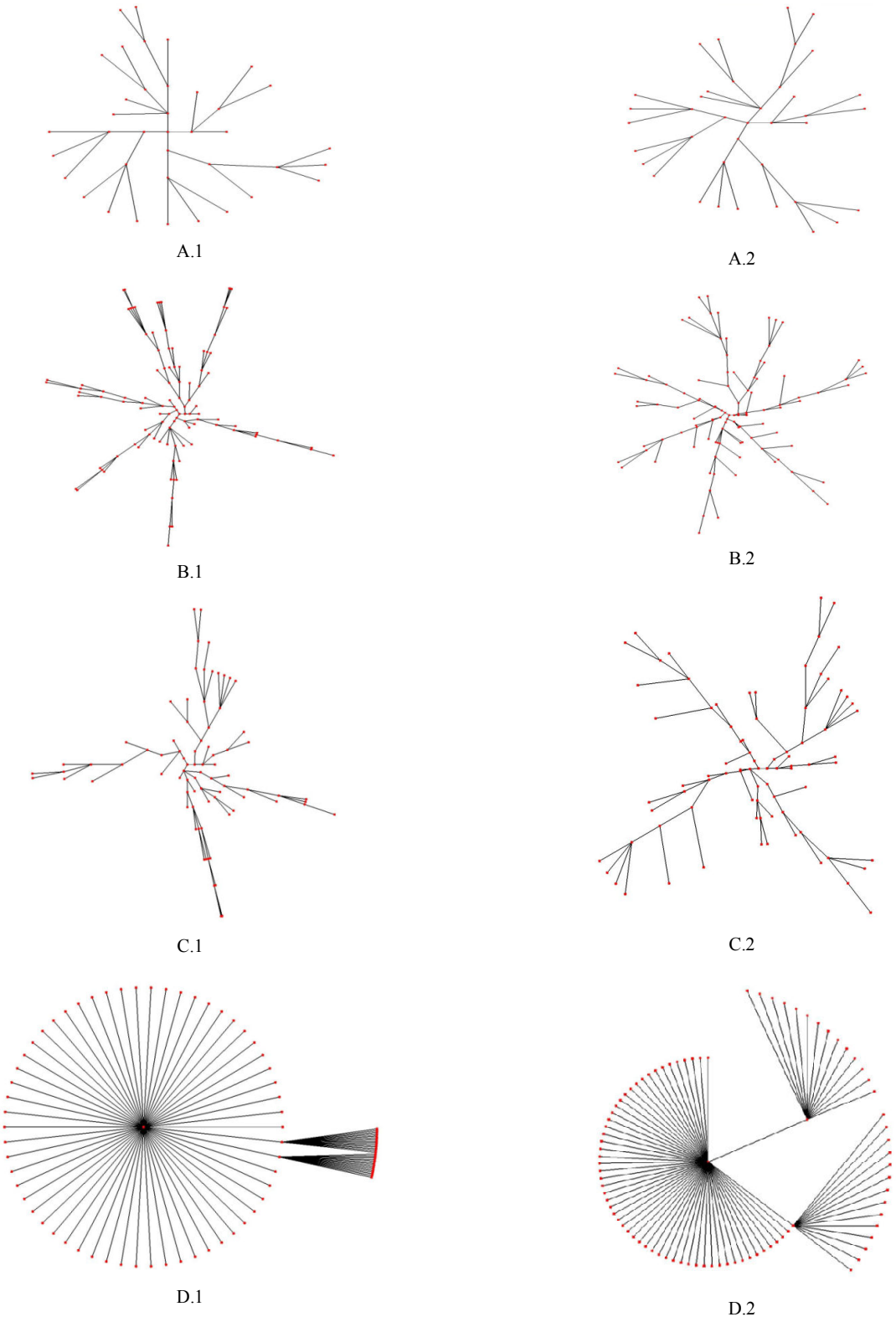

Fig. 4 Four different trees drawn by the two proposed algorithms

D. Visualization

To compute *Ai* and *Ac* for a node, we need to have its parent data (*Ac, Ai)*.Therefore those values should be computed starting from the drawing center whose angular interval(s) and angular coordinate(s) is(are) fixed as explained above. Once this is done, we need to find the geometric position *(x, y)* of each node. We use the angular coordinates for this purpose. They are scalar values which can be projected into a 2D Cartesian space with an appropriate technique.

In the drawings above, we used the MDS projection technique [5] which takes a dissimilarity matrix (computed for a set of scalar values) as an input and produces mD coordinates (m ≥2), in this work 2Dcoordinates. The drawing center does need special care as by definition it will be placed in the center of the drawing area. The dissimilarity matrix for a set of n scalar values is an $n \times n$ matrix M where M_{ij} is the distance between i and j, thus all diagonal elements will be 0. Our algorithms make use of angular coordinates; therefore the distance between two nodes can be computed as shown in (5).

$$D_{ij} = \sqrt{(\sin Ac_i - \sin Ac_j)^2 + (\cos Ac_i - \cos Ac_j)^2} \quad (5)$$

Since our goal is a radial drawing, all the nodes are placed along the edge of a circle with radius *1*. We need to take care for the distance between the layers in order to avoid overlaps. We do this similarly to how it is done [1] and [2], however we further adjust the distance between a pair of adjacent layers depending on the edge density between them. The larger the edge density, the larger is the distance between them for producing a clearer drawing. This can be observed in Fig. 4.

A simpler method for computing Cartesian coordinates would be to simply use $(d_v cosAc_v, d_v sinAc_v)$ as the position of node v, assuming d_v is the distance of node v from the drawing center. This method is faster; however it is limited to radial drawings and cannot be generalized for other types of drawings, which may use scalar values for producing might graph layouts, such as the dynamics-driven graph drawing introduced by Toosi et al. [4].

III. Result and Discussion

A. Comparison between the Algorithms

In Fig. 4 we show the results of our algorithms for 4 different tree topologies. The layouts in left column show the result from the first algorithm; and the right column shows the results from the second algorithm. It can be observed that the first algorithm divides the space reserved for the children of each node evenly between those children, disregarding their weights. Therefore, the further sibling nodes are located from the drawing center, the closer they are placed to each other. This approach, however, has the advantage that it *reserves* space for new leaves in the outer layers, which can be interactively (or real-time) introduced with minimum recalculation of the layout.

The second algorithm takes care of the node weights and this can be seen very well in the difference between drawings D.1 and D.2 in Fig. 4. In D.1, the space for future leaves in the third layer has been reserved, therefore, if a node is inserted in that layer there is no need to re-compute the positions of any other nodes. However, in D.2 a new leaf in the third layer leads to re-computing the coordinates of a portion of the nodes. From another point of view, the nodes in the third layer in D.1 are tightly close to each other, while in third layer in D.2 looks like a better drawing. In B.2 and C.2, we can also observe that nodes with heavier weight are better spaced in comparison to B.1 and C.1, respectively. However, for small-size trees, like the one in A.1, the result of the first algorithm is similar to the one of the second but more aesthetic for small size trees.

B. Discussion

We propose a graph visualization technique for trees which produces radial tree drawings. Two slightly different algorithms within the same approach have been proposed and applied. The time complexity for both algorithms is O(n) for a tree with n nodes.

The first algorithm has an advantage for drawing trees either interactively or in real-time. While it results in a tighter space for nodes in outer layers, it leads to drawings in which the branches of the tree are better expressed and easily distinguishable. We suggest that this version of our drawing approach might be suitable for drawing very large trees in real-time with preservation of the mental map, where it is more important to visualize the general structure of the tree at first and then apply an appropriate navigation technique to explore specific parts of the tree in detail.

The second algorithm is suitable enough for large trees since it takes care for the descendants of nodes in order to space them accordingly. This version of the algorithm can be a better option for static drawings of large trees. As our results suggest, either of the two algorithms can produce an aesthetically pleasing drawing for small trees.

Appendix

All the results have been produced with a system written in C++ and OpenGL. All example trees are generated by the authors. Our software and example trees are available on request.

References

[1] Rusu, A., *Tree Drawing Algorithms*, in Tamassia, R. (ed.): *Handbook of Graph Drawing and Visualization*. Chapman and Hall/ CRC, 2013, chapter 5, pp. 155–192.

[2] Book, G. and Keshary, N., *Radialtree graph drawing algorithm for representing large hierarchies.* Technical report, University of Connecticut, 2001.

[3] Alam, Md. J., Samee, Md. A. H., Rabbi, M. and Rahman, Md. S., *Upward drawings of trees on the minimum number of layers.* WALCOM, 2008, pp.88—99.

[4] Ghassemi Toosi, F., Paulovich, F. V., Hutt, M.-T. and Linsen, L. *Projection-based Visualization of Dynamical Processes on Networks.* Eurovis 2012, pp. 61--65.

[5] Cox, T.F. and Cox, A. A. *Multidimensional Scaling, Second Edition.* Chapman & Hall/CRC Monographs on Statics & Applied Probablility. Taylor & Francis, 2010.