An Experimental Study on Algorithms for Drawing Binary Trees

Adrian Rusu[1] **Radu Jianu**[2] **Confesor Santiago**[3] **Christopher Clement**[1]

[1]Department of Computer Science, Rowan University, Glassboro, NJ 08028, USA
[2]Department of Computer Science, Brown University, Providence, RI 02912, USA
[3]Department of Electrical and Computer Engineering, Rowan University, Glassboro, NJ 08028, USA
rusu@rowan.edu, jr@cs.brown.edu, {santia09, clemen55}@students.rowan.edu

Abstract

In this paper we present the results of a comprehensive experimental study on the four most representative algorithms for drawing binary trees without distorting or occluding the information, one for each of the following *distinct* tree-drawing approaches: **Separation-Based approach** (Garg & Rusu 2003), **Path-Based approach** (Chan, Goodrich, Rao Kosaraju & Tamassia 2002), **Level-Based approach** (Reingold & Tilford 1981), and **Ringed Circular Layout approach** (Teoh & Ma 2002).

Our study is conducted on randomly-generated, unbalanced, and AVL binary trees with up to 50,000 nodes, on Fibonacci trees with up to 46,367 nodes, on complete trees with up to 65,535 nodes, and on real-life molecular combinatory binary trees with up to 50,005 nodes. We compare the performance of the drawing algorithms with respect to five quality measures, namely **Area**, **Aspect Ratio**, **Uniform Edge Length**, **Angular Resolution**, and **Farthest Leaf**. None of the algorithms have been found to be the best in all categories.

Keywords: binary trees, experimental study, drawing algorithm, planar, straight-line

1 Introduction

Trees are ubiquitous data-structures, arising in a variety of applications such as Software Engineering, Business Administration, Knowledge Representation, and Web-site Design and Visualization.

Visualizing a tree can enhance a user's ability in understanding its structure. Hence, a lot of research has been done on visualizing trees, which has produced a plethora of tree-drawing algorithms (see for example, (Chan, Goodrich, Rao Kosaraju & Tamassia 2002, Garg & Rusu 2003, Reingold & Tilford 1981, Teoh & Ma 2002, Valiant 1981)).

An experimental evaluation of the practical performance of tree-drawing algorithms can help a practitioner in choosing the algorithm most appropriate for her application. However, we are not aware of any experimental study done to compare the practical performance of tree-drawing algorithms. As a first step, in this paper, we present an experimental study of some well-known algorithms for drawing binary trees. These algorithms represent the *distinct* approaches that have been used to draw binary trees without distorting or occluding the information.

A *binary* tree is one where each node has at most two children. In contrast to graphs, every tree accepts a *planar drawing*, i.e. without any crossings. A *straight-line* drawing has each edge drawn as a single line-segment. *Grid-based* algorithms place all the nodes of a drawing at integer coordinates. A drawing of a tree T has the *subtree separation* property if, for any two node-disjoint subtrees of T, the enclosing rectangles of the drawings of the two subtrees do not overlap with each other. All algorithms in our experimental study produce planar straight-line grid drawings and exhibit the subtree separation property.

2 The Algorithms Under Evaluation

We have compared four different algorithms for producing planar straight-line grid drawings of binary trees. The four algorithms can be classified into four categories on the basis of their approach to constructing drawings.

Separation-Based: In the *Separation-Based Approach*, a divide-and-conquer strategy is used to recursively construct a drawing of the tree, by performing the following actions at each recursive step: (1) *Find a Separator Edge*: A *separator edge* of a binary tree T is an edge, which, if removed, divides T into at most five smaller, partial binary trees. Every binary tree contains such an edge (Valiant 1981). (2) *Divide Tree*: Divide T into several partial binary trees by removing at most two nodes and their incident edges from it (including the separator edge). (3) *Assign Aspect Ratios*: Pre-assign a desirable aspect ratio to each partial binary tree. (4) *Draw Partial Trees*: Recursively construct a drawing of each partial binary tree using its pre-assigned aspect ratio. (5) *Compose Drawings*: Arrange the drawings of the partial binary trees and draw the nodes and edges that were removed from T to divide it such that the drawing of T thus obtained is a planar straight-line grid drawing. We have chosen to evaluate the $O(n)$-area bottom-up algorithm of (Garg & Rusu 2003) (we call it *Separation*).

Path-Based: The *Path-Based Approach* uses a recursive winding paradigm as follows: first lay down a small chain of nodes from left to right until near a distinguished node v, and then recursively lay out the subtrees rooted at the children of v in the opposite direction. For our study, we have implemented the $O(n \log \log n)$-area algorithm described in (Chan, Goodrich, Rao Kosaraju & Tamassia 2002) (we call it *Path*).

Level-Based: The *Level-Based Approach* is characterized by the fact that in the drawings produced, the nodes at the same distance from the root are horizontally aligned. For our study, we have implemented the recursive algorithm described in (Reingold &

Tilford 1981) (we call it *Level*). This algorithm uses the following steps: draw the subtree rooted at the left child, draw the subtree rooted at the right child, place the drawings of the subtrees at horizontal distance 2, and place the root one level above and halfway between the children. If there is only one child, place the root at horizontal distance 1 from the child.

Ringed Circular Layout: The algorithms based on the *Ringed Circular Layout Approach* place a node and all its children in a circle. For our study, we have implemented the algorithm described in (Teoh & Ma 2002) (we call it *Rings*). Note that this algorithm was designed for general trees. In this study, we have implemented and studied its performance for the particular case of binary trees. In this algorithm, equal-sized circles corresponding to children are placed in concentric rings inside of the parent circle, around its center, thus trying to minimize the space wasted inside of the interior of the parent circle.

Figure 1 shows drawings of the Fibonacci tree with 88 nodes constructed by the algorithms of our study.

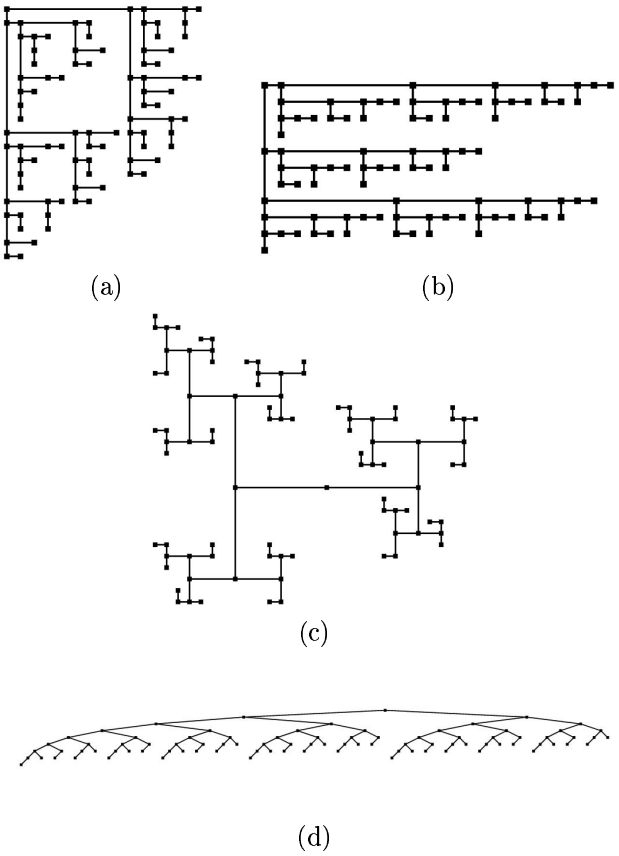

Figure 1: Drawings of the Fibonacci tree with 88 nodes, generated by the algorithms in our study: (a) Separation, (b) Path, (c) Rings, and (d) Level.

3 Experimental Setting

Our experimental setting consists of (*i*) a large suite of randomly-generated, unbalanced, complete, AVL, Fibonacci, and molecular combinatory binary trees of various sizes; (*ii*) five quality measures: area, aspect ratio, uniform edge length, angular resolution, and farthest leaf.

3.1 Test Suite

Our test suite consists of five binary trees for each of the following types: *random*, *complete*, *AVL*, *Fibonacci*, and *unbalanced*. We consider a binary tree T_n with n nodes as *unbalanced* if its height is greater than $n/\log n$. A binary tree T_n with n nodes is *unbalanced-to-the-left* (*unbalanced-to-the-right*) if it is unbalanced, and, in addition, the number of left (right) children in T_n is greater than its number of right (left) children. We have generated random, unbalanced, and AVL binary trees with up to 50,000 nodes, complete trees with up to 65,535 nodes, and Fibonacci trees with up to 46,367 nodes. In addition, our test suite includes binary trees from molecular combinatory programs. The data was obtained from the study in (MacLennan 2003) by Dr. Bruce MacLennan at the University of Tennessee. For our experimental analysis, he has generated molecular combinatory binary trees with up to 50,005 nodes.

3.2 Quality Measures

The following four well-known quality measures have been considered:

- **Area**: the number of grid points contained within the enclosing rectangle.
- **Aspect Ratio**: the ratio of the smaller and the longer sides of the enclosing rectangle.
- **Uniform Edge Length**: the variance of the edge lengths in the drawing.
- **Angular Resolution**: the smallest angle formed by two edges incident on the same node.

It is widely accepted (Di Battista, Eades, Tamassia, & Tollis 1999) that small values of the area and uniform edge length are related to the perceived aesthetic appeal and visual effectiveness of the drawing. In addition, an aspect ratio is considered *optimal* if it is equal to 1. High angular resolution is desirable in visualization applications and in the design of optical communication networks. For binary trees, the degree of a node is at most 3, hence a trivial upper bound on the angular resolution is 120°.

We have also considered a new quality measure, specially designed for trees: **Farthest Leaf**: the largest Euclidean distance between the root and a leaf in the drawing. Farthest leaf helps determine whether the algorithm places leaves far from the root. It is important to minimize the distance between the root and the leaves of the tree, especially in the case when the user wants to visually analyze binary search trees.

4 Experimental Analysis

Let T_n be a binary tree with n nodes that is provided as input to the algorithms being evaluated.

Two of the algorithms chosen in this study, namely *Separation* and *Path*, allow user-controlled aspect ratio. The other two algorithms, namely *Level* and *Rings*, generate unique drawings for each value of n. In order to find the parameters for which *Separation* and *Path* perform the best on each of the aesthetics considered in our study, we used the studies in (Garg & Rusu 2003) and (Rusu, Clement, & Jianu 2005), respectively.

Since, for any tree, when the desirable aspect ratio is set to 1, we can always find the actual aspect ratio of the drawing produced by *Separation* close to 1 (Garg & Rusu 2003), we decided not to evaluate the performance of *Separation* for this quality measure.

Since both *Path* and *Rings* produce *orthogonal drawings*, the angles between the edges connecting the nodes will always be either 90° or 180°. For this reason, we do not consider *Path* and *Rings* in our analysis of quality measure angular resolution.

Very interestingly, the performances for all algorithms on the real-life molecular combinatory binary trees resemble very closely those on unbalanced binary trees. Hence, we make many references to their similarities.

In the case of unbalanced and molecular combinatory binary trees, *Rings* quickly becomes prohibitive to use. For this reason, we have decided to not consider *Rings* in our comparisons for unbalanced and molecular combinatory binary trees.

Due to limited space, we only provide a textual analysis of the algorithms under evaluation. For the same reason, the analysis on unbalanced binary trees is not included in this paper. The complete display of performances for all categories of binary trees (which includes 84 charts), as well as the analysis on unbalanced binary trees and the performance of the algorithms on several other aesthetics (such as **Total Edge Length**, **Average Edge Length**, **Maximum Edge Length**, **Size**, **Closest Leaf**, and **Minimum Angle Size**), is provided in a technical report (Rusu, Jianu, Santiago, & Clement 2005).

The analysis of the performance of the four algorithms is summarized below:

Area:

- *Complete trees*: Order of performance: *Rings*, *Separation*, *Path*, *Level*. While the difference in the areas produced by *Rings* and *Separation* grows slowly, the difference in the areas produced by *Separation* and *Path* grows much faster. The same behavior is exhibited by *Level* and *Path*. For $n = 65,535$, *Level* produces a drawing having an area almost four times more than the drawing produced by *Path*.

- *Randomly-generated binary trees*: Order of performance: *Separation*, *Path*, *Level*, *Rings*. The performances of all the algorithms are worse than their respective performances on complete trees. In comparison to its behavior on complete trees, where it was the best, *Rings* exhibits the most dramatic change: its behavior is now the worst of all four algorithms. The area produced by *Level* grows rapidly in comparison to the area produced by *Path*, being already three times more for $n = 50,000$. *Rings* and *Separation* exhibit similar behavior, with *Rings* being slightly better. The differences in the areas produced grow slowly.

- *Fibonacci trees*: Order of performance: *Separation*, *Path*, *Level*, *Rings*. *Rings* quickly becomes prohibitive, with area ten times more than the area of *Separation*, for 10,000 nodes. The difference between the areas produced by *Separation* and *Path* grows slowly, while the difference between the areas produced by *Path* and *Level* grows much faster.

- *Molecular combinatory binary trees*: Order of performance: *Path*, *Separation*, *Level*, *Rings*. Even though *Path* is the best performing algorithm on both unbalanced and molecular combinatory binary trees, its behavior on molecular combinatory binary trees is much better: for $n = 50,000$, the area of molecular combinatory binary trees is 78,360, as opposed to unbalanced-to-the-left binary trees, with an area of 258,355. *Level* rapidly becomes prohibitive to use, producing an area over $1,000,000$ for $n = 5,989$. This is the best case for *Path* and the worst case for *Separation*.

Uniform Edge Length:

- *Complete trees*: Order of performance: *Rings*, *Separation*, *Path*, *Level*. *Rings* and *Separation* produce very low, almost constant values. *Path* exhibits a non-linear growing behavior. *Level* quickly becomes prohibitive to use.

- *Randomly-generated binary trees*: Order of performance: *Rings*, *Separation*, *Path*, *Level*. The performances of *Rings* and *Separation* are very similar to those on complete binary trees. *Path* grows slowly and linearly. *Level* quickly becomes prohibitive to use.

- *AVL trees*: Order of performance: *Rings*, *Separation*, *Path*, *Level*. The performances of *Rings* and *Separation* are very good: they almost mimic their performances on complete binary trees. The results for *Path* do not have a consistent rate of change and are about 50 times greater than *Rings*.

- *Fibonacci trees*: Order of performance: *Separation*, *Path*, *Rings*, *Level*. *Separation* outperforms all other algorithms, by maintaining a nearly constant value. The difference between the performance of *Path* and *Rings* stays at an approximately constant 30 as n increases.

- *Molecular combinatory binary trees*: Order of Performance: *Separation*, *Path*, *Level*, *Rings*. *Separation* has a slowly growing behavior, similar to unbalanced-to-the-left binary trees. Interestingly, *Level* performs very poorly, which is different from unbalanced binary trees. *Path* quickly becomes prohibitive with $n = 9,973$ resulting in a uniform edge length of 60.

Farthest Leaf:

- *Complete trees*: Order of performance: *Rings*, *Separation*, *Path*, *Level*. While the performances of *Path*, *Rings* and *Separation* are very good, with a very slow growth rate, the performance of *Level* is unsatisfactory. For example, for $n = 8,191$, farthest leaf for *Rings* is 89.1, for *Separation* is 219.1, for *Path* is 355.7, and for *Level* is 4,095. For $n = 65,535$, farthest leaf for *Rings* is 284.8, for *Separation* is 626, for *Path* is 811.3, and for *Level* is 32,767.

- *Randomly-generated binary trees*: Order of performance: *Separation*, *Rings*, *Path*, *Level*. Surprisingly, both *Separation* and *Rings* perform just slightly worse on randomly-generated binary trees compared to complete trees. Performances of *Path* and *Rings* are almost identical. Again, the distance to the farthest leaf grows much faster for *Level* than for *Path*, *Separation* and *Rings*.

- *AVL trees*: Order of performance: *Rings*, *Separation*, *Path*, *Level*. The performances of the algorithms on this measure are almost identical to their respective performances on complete trees.

- *Fibonacci trees*: Order of performance: *Separation*, *Path*, *Rings*, *Level*. For *Separation* and *Path* the same pattern remains. On the other hand, *Rings* still remains better than *Level*, but the rate of growth exhibited by *Rings* has increased. *Level* exhibits the same unsatisfactory behavior.

- *Molecular combinatory binary trees*: Order of performance: *Separation*, *Path*, *Level*, *Rings*. As the case in all categories of binary trees, *Level* rapidly becomes prohibitive. *Separation* produces satisfactory results, placing its farthest leaf

at a distance of 764.6 for $n = 50,005$. The behavior of *Path* is approximately three times greater than *Separation*.

Aspect Ratio:

- *Complete trees*: Order of performance: *Separation*, *Path*, *Rings*, *Level*. Quite interestingly, the behaviors of *Path* and *Rings* are very similar. Neither algorithm always produces drawings with aspect ratios close to optimal. For example, if $n = 2^{14} - 1$, the best aspect ratio *Path* produces is around 0.5. *Rings* produces optimal aspect ratios when $n = 2^i - 1$, with i an odd number, and aspect ratios close to 0.5, with i an even number. The aspect ratios of the drawings produced by *Level* are very low (the highest value is close to 0.06), decreasing rapidly as n increases.
- *Randomly-generated binary trees*: Order of performance: *Separation*, *Path*, *Rings*, *Level*. *Level* produces drawings with better aspect ratios for trees with a smaller number of nodes (the highest value is close to 0.1). Still, its behavior is unsatisfactory, as the value of aspect ratio decreases rapidly as n increases. *Path* and *Rings* have uneven behaviors. Most of their aspect ratios are over 0.8, and none is under 0.5.
- *AVL trees*: Order of performance: *Separation*, *Rings*, *Path*, *Level*. *Rings* exhibits a very interesting pattern: its aspect ratios are either 0.5 or optimal. The performances of *Path* and *Level* decrease dramatically, with *Level* quickly producing very small aspect ratios, and *Path* producing aspect ratios less than 0.01 for 50,000 nodes.
- *Fibonacci trees*: Order of performance: *Separation*, *Rings*, *Path*, *Level*. Interestingly, *Rings* exhibits exactly the same behavior as in the case of complete binary trees: alternating optimal aspect ratios with aspect ratios close to 0.5. The behavior of *Level* is only significant for trees with a small number of nodes.
- *Molecular combinatory binary trees*: Order of performance: *Separation*, *Path*, *Level*, *Rings*. *Path* produces close to optimal values until $n = 5,989$. After this point, the values plummet, decreasing to 0.1 for $n = 50,005$. Very interestingly, *Level* always produces values close to 0.1. In our analysis, it was discovered that *Level* always produces drawings of width equal to n. Hence, for molecular combinatory binary trees, the height of the drawings produced by *Level* is almost always one-tenth of the width.

Angular Resolution: Order of performance: *Path*, *Rings*, *Separation*, *Level*.

Path and *Rings* always produce orthogonal drawings, hence their angular resolution is always 90°. *Separation* produces orthogonal drawings for complete, AVL, and Fibonacci trees. *Level* produces unsatisfactory results, with angular resolution less than 10° for all types of trees. *Separation* exhibits good behavior on randomly-generated and molecular combinatory binary trees with small number of nodes. For unbalanced and molecular combinatory binary trees with large number of nodes, *Separation* produces unsatisfactory results.

5 Conclusion

The performance of a drawing algorithm on a tree-type is not a good predictor of the performance of the same algorithm on other tree-types: some of the algorithms perform best on one tree-type, and worst on other tree-types. Overall, if user-controlled aspect ratio is important, *Separation* is the algorithm which achieves best results in the majority of the aesthetics. If user-controlled aspect ratio is important, *Rings* should be the choice for AVL and complete trees, and *Path* should be the choice for unbalanced-to-the-right and molecular combinatory binary trees. If visualization of leaf nodes is important, then *Separation* should be the choice for randomly-generated, Fibonacci, and molecular combinatory binary trees, *Rings* should be the choice for complete and AVL trees, and *Path* should be the choice for unbalanced binary trees. If angular resolution is important, *Rings*, *Path*, and *Separation* are all good choices, only *Separation* is not recommended for unbalanced binary trees. *Level*, the algorithm used very often in practice, scores worse in comparison to the other algorithms for almost all tree-types and aesthetics considered in our study.

Acknowledgment

We are grateful to Dr. Bruce MacLennan from the Department of Computer Science at the University of Tennessee for providing to us the real-life molecular combinatory binary trees.

References

T. Chan, M. Goodrich, S. Rao Kosaraju, & R. Tamassia. (2002), Optimizing area and aspect ratio in straight-line orthogonal tree drawings, in *Computational Geometry: Theory and Applications*, 23:153-162.

G. Di Battista, P. Eades, R. Tamassia, & I. G. Tollis. (1999), *Graph Drawing*, Prentice Hall, Upper Saddle River, NJ.

A. Garg & A. Rusu. (2003), A more practical algorithm for drawing binary trees in linear area with arbitrary aspect ratio, in *Proceedings 11th International Symposium on Graph Drawing*, volume 2912 of *Lecture Notes Comput. Sci.*, pages 159-165, Springer.

B. MacLennan. (2003), Molecular Combinatory Computing for Nanostructure Synthesis and Control, in *Proceedings 3rd IEEE Conference on Nanotechnology*, IEEE Press, pp. 179-182 vol. 2.

E. Reingold & J. Tilford. (1981), Tidier drawings of trees, in *IEEE Transactions on Software Engineering*, 7(2):223-228.

A. Rusu, R. Jianu, C. Santiago, & C. Clement. (2005), Planar Straight-Line Grid Drawings of Binary Trees: An Experimental Study, *Technical Report 2005-01*, Department of Computer Science, Rowan University, Glassboro, NJ.

A. Rusu, C. Clement, & R. Jianu. (2005), Performance Analysis of a Path-Based Algorithm for Drawing Binary Trees, in *Proc. 5th International Conference on Artificial Intelligence and Digital Communications*, Research Notes in Computer Science, pages 84-102.

S. T. Teoh, K. L. Ma. (2002), RINGS: A Technique for Visualizing Large Hierarchies, in *Proceedings 10th International Symposium on Graph Drawing*, volume 2528 of *Lecture Notes Comput. Sci.*, pages 268-275, Springer.

L. Valiant. (1981), Universality considerations in VLSI circuits, in *IEEE Trans. Comput.*, C-30(2):135-140.