A More Practical Algorithm for Drawing Binary Trees in Linear Area with Arbitrary Aspect Ratio*

Ashim Garg and Adrian Rusu

Department of Computer Science and Engineering
University at Buffalo, Buffalo, NY 14260
{agarg,adirusu}@cse.buffalo.edu

Abstract. Trees are usually drawn using planar straight-line drawings. [1] presented an algorithm for constructing a planar straight-line grid drawing of an n-node binary tree with area $O(n)$ and any pre-specified aspect ratio in the range $[n^{-\alpha}, n^{\alpha}]$, where $0 \leq \alpha < 1$ is any constant, in $O(n \log n)$ time. Unfortunately, the algorithm of [1] is not suitable for practical use. The main problem is that the constant hidden in the "Oh" notation for area is quite large (e.g., it can be as large as 3900).
In this paper, we have made several practical improvements to the algorithm, which make it suitable for practical use. We have also conducted experiments on this newer version of the algorithm for randomly-generated and complete binary trees with up to $50,000$, and $65,535$ nodes, respectively. Our experiments show that it constructs area-efficient drawings in practice, with area at most 10 times and 8 times the number of nodes for randomly-generated and complete binary trees, respectively.

1 Introduction

A drawing Γ of a tree is a *straight-line* drawing, if each edge is drawn as a single line-segment. Γ is a *grid* drawing if all the nodes have integer coordinates. Γ is a *planar* drawing, if edges do not intersect each other. Here, we concentrate on grid drawings. So, we will assume that the plane is covered by a rectangular grid. Let Γ be a grid drawing. Let R be the smallest rectangle with sides parallel to the X- and Y-axes, respectively, that covers Γ completely. The *width* (*height*) of Γ is equal to 1+ width of R (1+height of R). The *area* of Γ is equal to (1+width of R)·(1+height of R), which is equal to the number of grid points contained within R. The *aspect ratio* of Γ is the ratio of its width and height.

* Research supported by NSF CAREER Award IIS-9985136, NSF CISE Research Infrastructure Award No. 0101244, and Mark Diamond Research Grant No. 13-Summer-2003 from GSA of The State University of New York.

G. Liotta (Ed.): GD 2003, LNCS 2912, pp. 159–165, 2004.

2 Our Result

Planar straight-line drawings are more esthetically pleasing than non-planar polyline drawings. Grid drawings guarantee at least unit distance between the nodes, and the integer coordinates of the nodes allow the drawings to be displayed in a display surface, such as a computer screen, without any distortions due to rounding-off errors. Giving users control over the aspect ratio of a drawing allows them to display the drawing in different kinds of display surfaces with different aspect ratios. Finally, it is important to minimize the area of a drawing, so that the users can display it in small display surfaces also.

[1] presented an algorithm for constructing a planar straight-line grid drawing of an n-node binary tree T with area $O(n)$ and with any pre-specified aspect ratio A in the range $[n^{-\alpha}, n^{\alpha}]$, where $0 \leq \alpha < 1$ is any constant, in $O(n \log n)$ time (the algorithm actually takes three input parameters: T, A, and a user-defined constant ϵ, such that $0 < \epsilon < 1$ and $n^{-\epsilon} \leq A \leq n^{\epsilon}$).

While the algorithm of [1] was significant from a theoretical point of view, it suffered from the following drawbacks, that made it unsuitable for practical use:

- The constant c hidden in the "Oh" notation for area can be quite large (e.g., it can be as large as 3900 for $\epsilon = 0.6$, and $A = 1$). One might argue that c is really the worst-case bound, and the algorithm might perform better in practice. However, the problem is that given a tree T with n nodes, and two numbers ϵ and A as input, the algorithm will always pre-allocate a rectangle R with size *exactly equal* to cn, and draw T within R. Thus, the area of R is always equal to the worst-case area, and correspondingly, the drawing also has a large area. This is the major drawback of this algorithm.
- Also, it uses another algorithm, called *Algorithm u^*-HV-Draw*, as a subroutine. This increases the complexity of implementing the algorithm.

In this paper, we have made several practical improvements to the algorithm, which make it more suitable for practical use: (Note that the area of the drawing constructed by this newer version of the algorithm is still $O(n)$)

- We have developed a newer version of the algorithm that does not require the pre-assignment of a rectangle with the worst-case area to draw a tree. Instead, it only pre-assigns an aspect ratio to the tree, which is used to draw the tree recursively in a bottom-up fashion. This makes it possible for the algorithm to construct a more area-efficient drawing in practice.
- This newer version does not require *Algorithm u^*-HV-Draw* as a subroutine, which makes it easier to implement.
- The proof for the area being $O(n)$ given in [1] is based on a theorem by Valiant (Theorem 6 of [3]). Unfortunately, the most natural way of using the theorem seemed to be requiring the pre-assignment of a rectangle (with a large area). Hence, developing an algorithm that does not pre-assign a rectangle required developing a new proof. Correspondingly, we have developed a new proof that does not use the theorem. Instead, it simply uses Induction.

– We have also implemented this newer version, and experimentally evaluated its performance for randomly-generated binary trees with up to 50,000 nodes, and for complete binary trees with up to $65,535 = 2^{16} - 1$ nodes. Our experiments show that it constructs area-efficient drawings in practice, with area at most 8 times the number of nodes for complete binary trees, and at most 10 times the number of nodes for randomly-generated binary trees.

Due to space limitations, in this paper, we only present a brief overview of this newer version of the algorithm. Full details are given in [2].

3 Preliminaries

Let T be an n-node binary tree, with one distinguished node v, which has at most one child. v is called the *link* node of T. T is an *ordered* tree if the children of each node are assigned a left-to-right order. A *partial tree* of T is a connected subgraph of T. If T is an ordered tree, then the *leftmost path* p of T is the maximal path consisting of nodes that are the left children of their parents, except the first one, which is the root of T; the last node of p is called the *leftmost* node of T. T is an *empty tree*, if it has zero nodes in it.

Let Γ be a planar straight-line grid drawing of T. Γ has a *good* aspect ratio, if its aspect ratio is in the range $[n^{-\alpha}, n^{\alpha}]$, where $0 \leq \alpha < 1$ is a constant. Let r be the root of T. Let u^* be the link node of T. Γ is a *feasible* drawing of T, if it has the following three properties:

– **Property 1**: The root r is placed at the top-left corner of Γ.
– **Property 2**: If $u^* \neq r$, then u^* is placed at the bottom boundary of Γ. Moreover, we can move u^* downwards in its vertical channel by any distance without causing any edge-crossings in Γ.
– **Property 3**: If $u^* = r$, then no other node or edge of T is placed on, or crosses the vertical and horizontal channels occupied by r.

Theorem 1 (Separator Theorem [3]). *Every n-node binary tree T contains an edge e, called a* separator edge*, such that removing e from T splits T into two trees with at most $(2/3)n$ nodes each. Moreover, e can be found in $O(n)$ time.*

4 Our Tree Drawing Algorithm

Let T be a n-node binary tree with a link node u^*. Let A and ϵ be numbers such that $0 < \epsilon < 1$, and $n^{-\epsilon} \leq A \leq n^{\epsilon}$. A is called the *desirable aspect ratio* for T.

Our tree drawing algorithm, called *DrawTree*, takes ϵ, A, and T as inputs, and uses a simple divide-and-conquer strategy to recursively construct a feasible drawing Γ of T, by performing the following actions at each recursive step:

– *Split Tree*: Convert T into an ordered tree, in which u^* is the leftmost node. Using Theorem 1, find a separator edge $e = (u, v)$. Using e, split T into at

most five partial trees by removing at most two nodes and their incident edges from it. We get two cases: in Case 1 (see Figure 1(a)), e is not in the leftmost path of T, and in Case 2 (see Figure 1(b)), e is in the leftmost path of T. The partial trees obtained in Case 1 are $T_A, T_\beta, T_1, T_2, T_C$, and in Case 2 are T_A, T_B, T_C. The nodes removed are a and u in Case 1, and u in Case 2. Based on which partial trees are empty, we get 7 subcases for Case 1 and 8 subcases for Case 2. These subcases are described in detail in [2] (Figures 1(a), and 1(b) show the subcases for Case 1 and 2, respectively, in which all the partial trees are non-empty).

– *Assign Aspect Ratios*: Correspondingly, assign a desirable aspect ratio A_k to each partial tree T_k. Let n_k be number of nodes in T_k. T_k is a *large* partial tree of T if either $A \geq 1$ and $n_k \geq (n/A)^{1/(1+\epsilon)}$, or $A < 1$ and $n_k \geq (An)^{1/(1+\epsilon)}$; T_k is a *small* partial tree otherwise.

  The assignment of A_k to T_k is done as follows: Let $x_k = n_k/n$.

  - If $A \geq 1$: If T_k is a large partial tree of T, then $A_k = x_k A$, otherwise (i.e., if T_k is a small partial tree of T) $A_k = n_k^{-\epsilon}$.
  - If $A < 1$: If T_k is a large partial tree of T, then $A_k = A/x_k$, otherwise (i.e., if T_k is a small partial tree of T) $A_k = n_k^{\epsilon}$.

  Moreover, if $A \geq 1$, and $T_k = T_A$ or $T_k = T_\beta$, then the value of A_k is changed to $1/A_k$. This is done so because later in Step *Compose Drawings*, when $A \geq 1$, T_A and T_β are rotated by 90°, when constructing Γ.

  Intuitively, the above assignment strategy ensures that each partial tree gets a good desirable aspect ratio.
– *Draw Partial Trees*: Recursively construct a feasible drawing Γ_k of each partial tree T_k with A_k as its desirable aspect ratio.
– *Compose Drawings*: Arrange the drawings of the partial trees, and draw the nodes and edges, that were removed from T to split it, to obtain a feasible drawing Γ of T. If $A < 1$, then the drawings of the partial trees are stacked one above the other, and if $A \geq 1$, they are placed side-by-side. Also note that when $A \geq 1$, the drawing of T_A and T_β are rotated clockwise by 90° and flipped left-to-right before placing in Γ. Figure 1(c) and 1(e) (Figures 1(d) and 1(f)) show the arrangement, when $A < 1$, and $A \geq 1$, respectively, for the case shown in Figure 1(a) (Figure 1(b)). For the arrangement in all the subcases of Case 1 and 2, see [2].

Figure 2 shows a drawing of the complete binary tree with 63 nodes constructed by Algorithm *DrawTree*, with $A = 1$ and $\epsilon = 0.2$.

Theorem 2. *Let T be a binary tree with n nodes. Given any number A, where $n^{-\alpha} \leq A \leq n^{\alpha}$, for some constant α, where $0 \leq \alpha < 1$, we can construct in $O(n \log n)$ time, a planar straight-line grid drawing of T with $O(n)$ area, and aspect ratio A, using Algorithm* DrawTree*.*

Proof. Omitted here due to lack of space. See [2] for the proof.

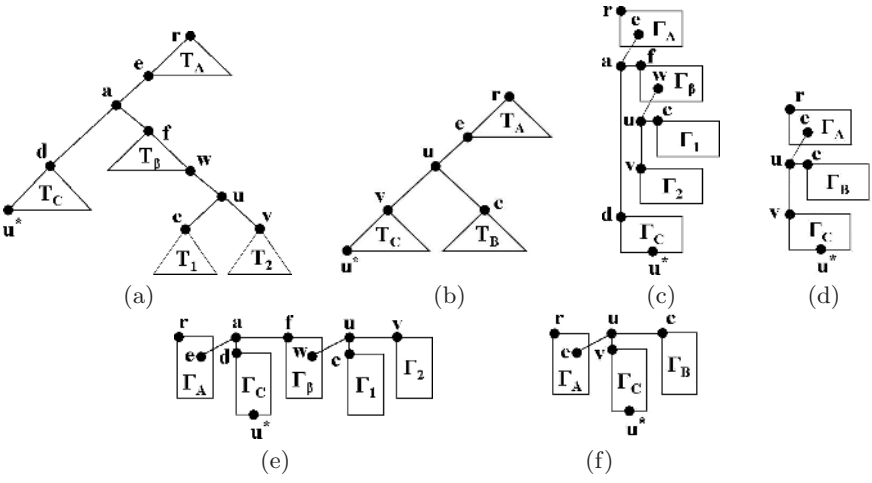


Fig. 1. General structure of tree T in (a) Case 1, and (b) Case 2, respectively; Drawing T, when $A < 1$, in (c) Case 1, and (d) Case 2, respectively, and when $A \geq 1$, in (e) Case 1, and (f) Case 2, respectively.

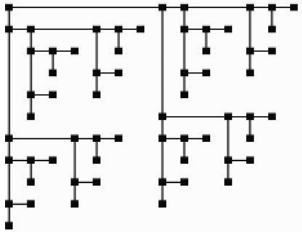


Fig. 2. Drawing of the complete binary tree with 63 nodes constructed by Algorithm *DrawTree*, with $A = 1$ and $\epsilon = 0.2$.

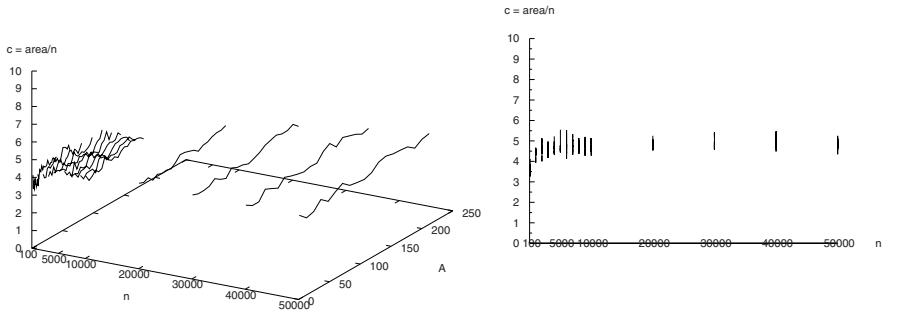
5 Experimental Results

We have implemented the algorithm using about 2100 lines of C++ code, and evaluated its performance on two types of binary trees, namely, randomly-generated, with up to 50,000 nodes, and complete, with up to $65,535$ nodes.

Recall that the algorithm takes three values as input: a binary tree T with n nodes, a number ϵ, where $0 < \epsilon < 1$, and a number A in the range $[n^{-\epsilon}, n^{\epsilon}]$.

The performance criteria we have used to evaluate the algorithm is the ratio c of the area of the drawing constructed of T, and n.

To evaluate the algorithm, we varied n up to $50,000$ for randomly-generated, and up to $65,535 = 2^{16} - 1$ for complete binary trees. For each n, we used five different values for ϵ, namely, 0.1, 0.25, 0.5, 0.75, and 0.9. For each (n, ϵ) pair, we used 20 different values of A uniformly distributed in the range $[1, n^{\epsilon}]$. The performance of the algorithm is symmetrical for $A < 1$ and $A > 1$. Hence, we varied A only from 1 through n^{ϵ}, not from $n^{-\epsilon}$ through n^{ϵ}. Hence, in the rest



(a) (b)

Fig. 3. (a) Performance of the algorithm, as given by the value of c, for drawing a randomly-generated binary tree T with $\epsilon = 0.5$, and different values of A, where *c=area of drawing/number of nodes n in T*. (b) Projection of the 3D plot of (a) on XZ-plane.

of the section, we will assume that $A \geq 1$. For each type of tree (randomly-generated and complete), and for each triplet (n, A, ϵ), we generated three trees of that type. We constructed a drawing of each tree using the algorithm, and computed the value of c. Next, we averaged the values of c obtained for the three trees to get a single value for each triplet (n, A, ϵ) for each tree-type.

Our experiments show that the value of c is generally small, and is at most 10 for randomly-generated, and at most 8 for complete trees. Figure 3 shows how c varies with n, and A, for $\epsilon = 0.5$ for randomly-generated trees. (See [2], to see how c varies with n, and A, for all the five values of ϵ for both randomly-generated and complete binary trees.)

We also discovered that c increases with A for a given n and ϵ. However, the rate of increase is very small. Consequently, for a given n and ϵ, the range for c over all the values of A is small (see Figure 3). E.g., for $n = 10,000$, and $\epsilon = 0.5$, for randomly-generated trees, the range for c is $[4.2, 5.2]$.

Similarly, for a given n and A, c increases with ϵ.

Finally, we would like to comment that the aspect ratio A_{act} of the drawing Γ constructed by the algorithm is, in general, different from the input aspect ratio A. We computed the ratio r of A_{act} and A. We discovered that r is close to 1 for $A = 1$, generally decreases as we increase A, and can get as low as 0.1 for $A = n^{\epsilon}$. However, we also discovered that for a large range of values for A, namely, $[1, \min\{n^{\epsilon}, n/\log^2 n\}]$, r stays within the range $[0.8, 1.5]$, and so is close to 1. Hence, if we need to construct a drawing with aspect ratio exactly equal to A, we can do so by adding enough "white space" to Γ. This will increase the area of Γ by a factor of at most $\max\{1/0.8, 1.5\} = 1.5$ (assuming that A is in the above-mentioned range). Hence, the area of Γ will still be small.

References

1. A. Garg and A. Rusu. Straight-line drawings of binary trees with linear area and arbitrary aspect ratio. In *Proc. 10th International Symposium on Graph Drawing (GD 2002)*, volume 2528 of *LNCS*, pages 320–331. Springer-Verlag, 2002.
2. A. Garg and A. Rusu. A more practical algorithm for drawing binary trees in linear area with arbitrary aspect ratio. Technical Report No. 2003-12, Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY, 2003.
3. L. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Comput.*, C-30(2):135–140, 1981.