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

Ashim Garg       Adrian Rusu

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

Abstract

Trees are usually drawn planar, i.e. without any edge-crossings. It is important to minimize the area of a drawing, so that it can fit within the given drawing-window. It is also important to give user some control over the aspect ratio of the drawing, so that she can display the drawing in a drawing-window with arbitrary width-to-height ratio. In a grid drawing, each node is assigned integer coordinates. In [6], it was shown that any binary tree with \( n \) nodes admits a planar straight-line grid drawing with area \( O(n) \) and with any pre-specified aspect ratio in the range \( [n^{-\alpha}, n^\alpha] \), where \( \alpha \) is a constant such that \( 0 \leq \alpha < 1 \). It was also shown that such a drawing can be constructed in \( O(n \log n) \) time. In particular, this showed that optimal area (equal to \( O(n) \)) and optimal aspect ratio (equal to 1) is simultaneously achievable for such drawings.

However, the algorithm of [6] is not suitable for practical use. The main problem is that the constant \( c \) hidden in the “Oh” notation for area is quite large (for example, it can be as high as 3900).

In this paper, we make several non-trivial 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 binary trees with up to 30,000 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.

1 Introduction

Trees are very common data-structures, which are used to model information in a variety of applications. A drawing \( \Gamma \) of a tree \( T \) maps each node of \( T \) to a distinct point in the plane, and each edge \( (u,v) \) of \( T \) to a simple Jordan curve with endpoints \( u \) and \( v \). \( \Gamma \) is a straight-line drawing, if each edge is drawn as a single line-segment. \( \Gamma \) is a grid drawing if all the nodes have integer coordinates. \( \Gamma \) is a planar drawing, if edges do not intersect each other in the drawing. In this paper, we concentrate on grid drawings. So, we will assume that the plane is covered by a rectangular grid. Let \( R \) be a rectangle with sides parallel to the \( X \) and \( Y \)-axes. The width (height) of \( R \) is equal to the number of grid points with the same \( y \) (\( x \)) coordinate contained within \( R \). The area of \( R \) is equal to the number of grid points contained within \( R \). The aspect ratio of \( R \) is the ratio of its width and height. \( R \) is the enclosing rectangle of \( \Gamma \), if it is the smallest rectangle that covers the entire drawing. The width, height, area, and aspect ratio of \( \Gamma \) is equal to the width, height, area, and aspect ratio, respectively, of its enclosing rectangle. \( T \) is a binary tree if each node has at most two children. We denote by \( T[v] \), the subtree of \( T \) rooted at a node \( v \) of \( T \). \( T[v] \) consists of \( v \) and all the

* Research supported by NSF CAREER Award No. IIS-9885136, 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.
descendants of $v$. $\Gamma$ has the subtree separation property [1] if, for any two node-disjoint subtrees $T[u]$ and $T[v]$ of $T$, the enclosing rectangles of the drawings of $T[u]$ and $T[v]$ do not overlap with each other. Drawings with subtree separation property are more aesthetically pleasing than those without subtree separation property. The subtree separation property also allows for a focus+context style [8] rendering of the drawing, so that if the tree has too many nodes to fit in the given drawing area, then the subtrees closer to focus can be shown in detail, whereas those further away from the focus can be contracted and simply shown as filled-in rectangles.

2 Our Result

Planar straight-line drawings are more aesthetically pleasing than non-planar polyline drawings. Grid drawings guarantee at least unit distance separation between the nodes of the tree, and the integer coordinates of the nodes and edge-bends allow the drawings to be displayed in a display surface, such as a computer screen, without any distortions due to truncation and 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. The subtree separation property makes it easier for the user to detect the subtrees in the drawing, and also allows for a focus+context style [8] rendering of the drawing. Finally, it is important to minimize the area of a drawing, so that the users can display a tree in as small drawing area as possible.

It is therefore important to investigate the problem of constructing planar straight-line grid drawings of binary trees with small area. Clearly, any planar grid drawing of a binary tree with $n$ nodes requires $\Omega(n)$ area. A long-standing fundamental problem, therefore, was that whether this is a tight bound also, i.e., given a binary tree $T$ with $n$ nodes, can we construct a planar straight-line grid drawing of $T$ with area $O(n)$?

[6] settled this problem by proving that a binary tree can be drawn in linear area with arbitrary aspect ratio in the range $[n^{-\epsilon}, n^\epsilon]$, $0 \leq \epsilon < 1$, in $O(n \log n)$ time. In particular, this result showed that optimal area (equal to $O(n)$) and optimal aspect ratio (equal to 1) is simultaneously achievable (see Corollary 1). The drawing constructed also exhibited subtree separation property.

While the result of [6] was significant from a theoretical point of view, the drawing-algorithm presented in [6] 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 (for example, it can be as high as 3900 for $\epsilon = 0.6$, and $A = 1$). One might argue that $c$ is really a 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 $\epsilon$ 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. This makes it possible for the algorithm to construct a more area-efficient drawing of a tree in practice than that by the algorithm of [6]. The main difference between this newer version and the algorithm of [6] is as follows: The algorithm of [6] is a recursive algorithm, which, in each recursive step, splits a tree $T$ into several small trees, correspondingly splits its pre-assigned rectangle $R$ into several smaller rectangles, assigns these smaller rectangles to the smaller trees, recursively draws the smaller trees within their pre-assigned rectangles, and then combine their drawings to obtain a drawing of $T$. In this newer version, instead of pre-assigning a rectangle, we only pre-assign an aspect ratio to each smaller tree, recursively construct a drawing of each smaller tree using its pre-assigned aspect ratio, and then combine their drawings to obtain a drawing of $T$.
- This newer version does not require Algorithm $u^*-HV$-Draw as a subroutine, which makes it easier to implement.
• The proof for the area (see Lemma 4) given in [6] is based on a theorem by Leslie Valiant (Theorem 6 of [11]). 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 mathematical 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 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.

3 Previous Results

Previous to the result of [6], the best-known upper bound on the area of a planar straight-line grid drawing of an n-node binary tree was \( O(n \log \log n) \), which was shown in [1] and [9].

However, note that the \( O(n \log \log n) \) area drawn constructed by the algorithms of [1] and [9] has a fixed aspect ratio, equal to \( \theta(\log^2 n/(n \log \log n)) \), whereas the aspect ratio of the drawing constructed by our algorithm can be specified by the user.

We now summarize some other known results on planar grid drawings of binary trees (for more results, see [4]). Let \( T \) be an n-node binary tree. [5] presents an algorithm for constructing an upward polyline drawing of \( T \) with \( O(n) \) area, and any user-specified aspect ratio in the range \( [n^{-\alpha}, n^\alpha] \), where \( \alpha \) is any constant, such that \( 0 \leq \alpha < 1 \). [7] and [11] present algorithms for constructing a (non-upward) orthogonal polyline drawing of \( T \) with \( O(n) \) area. [1] gives an algorithm for constructing an upward orthogonal straight-line drawing of \( T \) with \( O(n \log n) \) area, and any user-specified aspect ratio in the range \( [\log n/n, n/\log n] \). It also shows that \( n \log n \) is also a tight bound for such drawings. [9] gives an algorithm for constructing an upward straight-line drawing of \( T \) with \( O(n \log \log n) \) area. If \( T \) is a Fibonacci tree, (AVL tree, complete binary tree), then [2, 10] ([3], [2], respectively) give algorithms for constructing an upward straight-line drawing of \( T \) with \( O(n) \) area.

Table 1 summarizes these results.

<table>
<thead>
<tr>
<th>Tree Type</th>
<th>Drawing Type</th>
<th>Area</th>
<th>Aspect Ratio</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fibonacci</td>
<td>Upward Straight-line</td>
<td>( O(n) )</td>
<td>( \theta(1) )</td>
<td>[2, 10]</td>
</tr>
<tr>
<td>AVL</td>
<td>Upward Straight-line</td>
<td>( O(n) )</td>
<td>( \theta(1) )</td>
<td>[3]</td>
</tr>
<tr>
<td>Complete Binary</td>
<td>Upward Straight-line</td>
<td>( O(n) )</td>
<td>( \theta(1) )</td>
<td>[2]</td>
</tr>
<tr>
<td>General Binary</td>
<td>Upward Orthogonal Polyline</td>
<td>( O(n \log \log n) )</td>
<td>( \theta(\log^2 n/(n \log \log n)) )</td>
<td>[5, 9]</td>
</tr>
<tr>
<td>(Non-upward) Orthogonal Polyline</td>
<td>( O(n) )</td>
<td>( \theta(1) )</td>
<td></td>
<td>[7, 11]</td>
</tr>
<tr>
<td>Upward Orthogonal Straight-line</td>
<td>( O(n \log n) )</td>
<td>( [\log n/n, n/\log n] )</td>
<td></td>
<td>[1]</td>
</tr>
<tr>
<td>Upward Polyline</td>
<td>( O(n) )</td>
<td>( [n^{-\alpha}, n^\alpha] )</td>
<td></td>
<td>[5]</td>
</tr>
<tr>
<td>Upward Straight-line</td>
<td>( O(n \log \log n) )</td>
<td>( \theta(\log^2 n/(n \log \log n)) )</td>
<td></td>
<td>[9]</td>
</tr>
<tr>
<td>(Non-upward) Straight-line</td>
<td>( O(n \log \log n) )</td>
<td>( \theta(\log^2 n/(n \log \log n)) )</td>
<td></td>
<td>[1]</td>
</tr>
</tbody>
</table>

Table 1: Bounds on the areas and aspect ratios of various kinds of planar grid drawings of an n-node binary tree. Here, \( \alpha \) is a constant, such that \( 0 \leq \alpha < 1 \).
4 Preliminaries

Throughout this paper, by the term drawing, we will mean a planar straight-line grid drawing. We will assume that the plane is covered by an infinite rectangular grid. A horizontal channel (vertical channel) is an infinite line parallel to X- (Y-) axis, passing through the grid-points.

Let $T$ be a tree, with one distinguished node $v$, which has at most one child. $v$ is called the link node of $T$. Let $n$ be the number of nodes in $T$. $T$ 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 leftmost children, except the first one, which is the root of $T$. The last node of $p$ is called the leftmost node of $T$. Two nodes of $T$ are siblings if they have the same parent in $T$. $T$ is an empty tree, i.e., $T = \phi$, if it has zero nodes in it.

Let $\Gamma$ be a drawing of $T$. By bottom (top, left, and right, respectively) boundary of $\Gamma$, we will mean the bottom (top, left, and right, respectively) boundary of the enclosing rectangle $R(\Gamma)$ of $\Gamma$. Similarly, by top-left (top-right, bottom-left, and bottom-right, respectively) corner of $\Gamma$, we mean the top-left (top-right, bottom-left, and bottom-right, respectively) corner of $R(\Gamma)$.

Let $R$ be a rectangle, such that $\Gamma$ is entirely contained within $R$. $R$ 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 $T$. $\Gamma$ 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 $\Gamma$.
- **Property 2**: If $u^* \neq r$, then $u^*$ is placed at the bottom boundary of $\Gamma$. Moreover, we can move $u^*$ downwards in its vertical channel by any distance without causing any edge-crossings in $\Gamma$.
- **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 [11])** 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 $T_1$ and $T_2$, with $n_1$ and $n_2$ nodes, respectively, such that for some $x$, where $1/3 \leq x \leq 2/3$, $n_1 \leq xn$, and $n_2 \leq (1-x)n$. Moreover, $e$ can be found in $O(n)$ time.

Let $v$ be a node of tree $T$ located at grid point $(i,j)$ in $\Gamma$. Let $\Gamma$ be a drawing of $T$. Assume that the root $r$ of $T$ is located at the grid point $(0,0)$ in $\Gamma$. We define the following operations on $\Gamma$ (see Figure 1):

- **rotate operation**: rotate $\Gamma$ counterclockwise by $\delta$ degrees around the z-axis passing through $r$. After a rotation by $\delta$ degrees of $\Gamma$, node $v$ will get relocated to the point $(i \cos \delta - j \sin \delta, i \sin \delta + j \cos \delta)$. In particular, after rotating $\Gamma$ by $90^\circ$, node $v$ will get relocated to the grid point $(-j, i)$.
- **flip operation**: flip $\Gamma$ vertically or horizontally. After a horizontal flip of $\Gamma$, node $v$ will be located at grid point $(-i,j)$. After a vertical flip of $\Gamma$, node $v$ will be located at grid point $(i,-j)$.
5 Our Tree Drawing Algorithm

\[ \text{Figure 2: Drawing } T \text{ in all the seven subcases of Case 1 (when the separator } (u, v) \text{ is not in the leftmost path of } T): (a) } T_A \neq \emptyset, T_C \neq \emptyset, d \neq u^*, (b) T_A = \emptyset, T_C = \emptyset, (c) T_A \neq \emptyset, T_C \neq \emptyset, d = u^*, (d) T_A \neq \emptyset, T_C = \emptyset, r \neq e, (e) T_A \neq \emptyset, T_C = \emptyset, r = e, (f) T_A = \emptyset, T_C \neq \emptyset, d \neq u^*, \text{ and (g) } T_A = \emptyset, T_C \neq \emptyset, d = u^*. \]

For each subcase, we first show the structure of } T \text{ for that subcase, then its drawing when } A < 1, \text{ and then its drawing when } A \geq 1. \text{ Here, } x \text{ is the same as } f \text{ if } T \beta \neq \phi \text{ and is the same as the root of } T_A \text{ if } T \beta = \phi. \text{ In Subcases (a) and (c), for simplicity, } e \text{ is shown to be in the interior of } \Gamma_A, \text{ but actually, either it is the same as } r, \text{ or if } A < 1 (A \geq 1), \text{ then it is placed on the bottom (right) boundary of } \Gamma_A. \text{ For simplicity, we have shown } \Gamma_A, \Gamma_B, \text{ and } \Gamma_C \text{ as identically sized boxes, but in actuality, they may have different sizes.}

Let } T \text{ be a binary tree with a link node } u^*. \text{ Let } n \text{ be the number of nodes in } T. \text{ Let } A \text{ and } \epsilon \text{ be two numbers such that } 0 < \epsilon < 1, \text{ and } A \text{ is in the range } [n^{-\epsilon}, n^{\epsilon}]. A \text{ is called the \textit{desirable aspect ratio} for } T. \text{ Our tree drawing algorithm, called } \textit{DrawTree}, \text{ takes } \epsilon, A, \text{ and } T \text{ as input, and uses a simple divide-and-conquer strategy to recursively construct a feasible drawing } \Gamma \text{ of } T, \text{ by performing the following actions at each recursive step (as we will prove later, } \Gamma \text{ will fit inside a rectangle with area } O(n) \text{ and aspect ratio } A):}

- \textit{Split Tree}: Split } T \text{ into at most five partial trees by removing at most two nodes and their incident edges from it. Each partial tree has at most } (2/3)n \text{ nodes. Based on the arrangement of these partial trees within } T, \text{ we get two cases, which are shown in Figures 2 and 3, and described later in Section 5.1.}

- \textit{Assign Aspect Ratios}: Correspondingly, assign a desirable aspect ratio } A_k \text{ to each partial tree } T_k. \text{ The value of } A_k \text{ is based on the value of } A, \text{ and the number of nodes in } T_k.

- \textit{Draw Partial Trees}: Recursively construct a feasible drawing of each partial tree } T_k \text{ with } A_k \text{ as its desirable aspect ratio.}

- \textit{Compose Drawings}: Arrange the drawings of the partial trees, and draw the nodes and edges, that
Figure 3: Drawing $T$ in all the eight subcases of Case 2 (when the separator $(u, v)$ is in the leftmost path of $T$): (a) $T_A \neq \emptyset$, $T_B \neq \emptyset$, $v \neq u^*$, (b) $T_A = \emptyset$, $T_B = \emptyset$, $v \neq u^*$, (c) $T_A = \emptyset$, $T_B \neq \emptyset$, $v \neq u^*$, (d) $T_A \neq \emptyset$, $T_B = \emptyset$, $v \neq u^*$, (e) $T_A \neq \emptyset$, $T_B \neq \emptyset$, $v = u^*$, (f) $T_A = \emptyset$, $T_B = \emptyset$, $v = u^*$, (g) $T_A = \emptyset$, $T_B \neq \emptyset$, $v = u^*$, and (h) $T_A \neq \emptyset$, $T_B = \emptyset$, $v = u^*$. For each subcase, we first show the structure of $T$ for that subcase, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. In Subcases (a), (d), (e), and (h), for simplicity, $e$ is shown to be in the interior of $\Gamma_A$, but actually, either it is same as $r$, or if $A < 1$ ($A \geq 1$), then it is placed on the bottom (right) boundary of $\Gamma_A$. For simplicity, we have shown $\Gamma_A$, $\Gamma_B$, and $\Gamma_C$ as identically sized boxes, but in actuality, they may have different sizes.

were removed from $T$ to split it, such that the drawing $\Gamma$ of $T$ thus obtained is a feasible drawing. Note that the arrangement of these drawings is done based on the cases shown in Figures 2 and 3. In each case, if $A < 1$, then the drawings of the partial trees are stacked one above the other, and if $A \geq 1$, then they are placed side-by-side.

Figure 4 shows a drawing of the complete binary tree with 63 nodes constructed by Algorithm $DrawTree$, with $A = 1$ and $\epsilon = 0.2$.

We now give the details of each action performed by Algorithm $DrawTree$:

### 5.1 Split Tree

The splitting of tree $T$ into partial trees is done as follows:

- Order the children of each node such that $u^*$ becomes the leftmost node of $T$.
- Using Theorem 1, find a separator edge $(u, v)$ of $T$, where $u$ is the parent of $v$.
- Based on whether, or not, $(u, v)$ is in the leftmost path of $T$, we get two cases:
  - **Case 1**: The separator edge $(u, v)$ is not in the leftmost path of $T$. We get seven subcases: (a) In the general case, $T$ has the form as shown in Figure 2(a). In this figure:
Figure 4: Drawing of the complete binary tree with 63 nodes constructed by Algorithm DrawTree, with $A = 1$ and $\epsilon = 0.2$.

* $r$ is the root of $T$,  
* $T_2$ is the subtree of $T$ rooted at $v$,  
* $c$ is the sibling of $v$, $T_1$ is the subtree rooted at $c$,  
* $w$ is the parent of $u$,  
* $a$ is the last common node of the path $r \leadsto v$ and the leftmost path of $T$,  
* $f$ is the right child of $a$,  
* if $u \neq a$ then $T_\alpha$ is the subtree rooted at $u$, otherwise $T_\alpha = T_2$,  
* $T_\beta$ is the maximal tree rooted at $f$ that contains $w$ but not $u$,  
* $T_B$ is the tree consisting of the trees $T_\alpha$ and $T_\beta$, and the edge $(w, u)$,  
* $e$ is the parent of $a$, and $d$ is the left child of $a$,  
* $T_A$ is the maximal tree rooted at $r$ that contains $e$ but not $a$,  
* $T_C$ is the tree rooted at $d$, and  
* $d \neq u^*$.

In addition to this general case, we get six special cases: (b) when $T_A = \emptyset$ and $T_C = \emptyset$ (see Figure 2(b)), (c) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $d = u^*$ (see Figure 2(c)), (d) $T_A \neq \emptyset$, $T_C = \emptyset$, $r = e$ (see Figure 2(d)), (e) $T_A \neq \emptyset$, $T_C = \emptyset$, $r = e$ (see Figure 2(e)), (f) $T_A = \emptyset$, $T_C \neq \emptyset$, $d \neq u^*$ (see Figure 2(f)), and (g) $T_A = \emptyset$, $T_C \neq \emptyset$, $d = u^*$ (see Figure 2(g)). (The reason we get these seven subcases is as follows: $T_2$ has at least $n/3$ nodes in it because of Theorem 1. Hence $T_2 \neq \emptyset$, and so, $T_B \neq \emptyset$. Based on whether $T_A = \emptyset$ or not, $T_C = \emptyset$ or not, $d = u^*$ or not, and $r = e$ or not, we get totally sixteen cases. From these sixteen cases, we obtain the above seven subcases, by grouping some of these cases together. For example, the cases $T_A = \emptyset$, $T_C = \emptyset$, $d \neq u^*$, $r = u^*$, and $T_A = \emptyset$, $T_C = 0$, $d \neq u^*$, $r = u^*$ are grouped together to give Case (a), i.e., $T_A = \emptyset$, $T_C = \emptyset$, $d \neq u^*$, $r = e$. So, Case (a) corresponds to 2 cases. Similarly, Cases (c), (d), (e), and (f) correspond to 2 cases each, and Case (b) corresponds to 4 cases.) In each case, we remove nodes $a$ and $u$, and their incident edges, to split $T$ into at most five partial trees $T_A$, $T_C$, $T_B$, $T_1$, and $T_2$. We also designate $e$ as the link node of $T_A$, $w$ as the link node of $T_B$, and $u^*$ as the link node of $T_C$. We arbitrarily select a leaf of $T_1$, and a leaf of $T_2$, and designate them as the link nodes of $T_1$ and $T_2$, respectively.

Case 2: The separator edge $(u, v)$ is in the leftmost path of $T$. We get eight subcases: (a) In the general case, $T$ has the form as shown in Figure 3(a). In this figure,
* r is the root of T,
* c is the right child of u,
* TB is the subtree of T rooted at c,
* e is the parent of u,
* TA is the maximal tree rooted at r that contains e but not u,
* TC is the tree rooted at v, and
* v ≠ u*.

In addition to the general case, we get the following seven special cases: (b) TA = ∅, TB = ∅, v ≠ u* (see Figure 3(b)), (c) TA = ∅, TB ≠ ∅, v ≠ u* (see Figure 3(c)), (d) TA ≠ ∅, TB = ∅, v ≠ u* (see Figure 3(d)), (e) TA ≠ ∅, TB ≠ ∅, v = u* (see Figure 3(e)), (f) TA = ∅, TB = ∅, v = u* (see Figure 3(f)), (g) TA = ∅, TB ≠ ∅, v = u* (see Figure 3(g)), and (h) TA ≠ ∅, TB = ∅, v = u* (see Figure 3(h)). (The reason we get these eight subcases is as follows: TC has at least n/3 nodes in it because of Theorem 1. Hence, TC ≠ φ. Based on whether TA = φ or not, TB = φ or not, and v = u* or not, we get the eight subcases given above.) In each case, we remove node u, and its incident edges, to split T into at most three partial trees TA, TB, and TC. We also designate e as the link node of TA, and u* as the link node of TC. We arbitrarily select a leaf of TB and designate it as the link node of TB.

5.2 Assign Aspect Ratios

Let Tk be a partial tree of T, where for Case 1, Tk is either TA, TC, Tβ, T1, or T2, and for Case 2, Tk is either TA, TB, or TC. Let nk be the number of nodes in Tk.

**Definition:** Tk is a large partial tree of T if:

- A ≥ 1 and nk ≥ (n/A)1/(1+ε), or
- A < 1 and nk ≥ (An)1/(1+ε),

and is a small partial tree of T otherwise.

In Step Assign Aspect Ratios, we assign a desirable aspect ratio Ak to each nonempty Tk as follows: Let xk = nk/n.

- If A ≥ 1: If Tk is a large partial tree of T, then Ak = xk A, otherwise (i.e., if Tk is a small partial tree of T) Ak = nk A.
- If A < 1: If Tk is a large partial tree of T, then Ak = A/xk, otherwise (i.e., if Tk is a small partial tree of T) Ak = nk A.

Intuitively, this assignment strategy ensures that each partial tree gets a good desirable aspect ratio, and so, the drawing of each partial tree constructed recursively by Algorithm DrawTree will fit inside a rectangle with linear area and good aspect ratio.

5.3 Draw Partial Trees

First, we change the desirable aspect ratios assigned to TA and Tβ in some cases as follows: Suppose TA and Tβ get assigned desirable aspect ratios equal to m and p, respectively, where m and p are some positive numbers. In Subcase (d) of Case 1, and if A ≥ 1, then in Subcases (a) and (c) of Case 1, and Subcases (a), (d), (e), and (h) of Case 2, we change the value of the desirable aspect ratio of TA to 1/m. In Case 1, if A ≥ 1, we change the value of the desirable aspect ratio of Tβ to 1/p. We make these changes because, as explained later in Section 5.4, in these cases, we need to rotate the drawings of TA and Tβ by 90° during the Compose Drawings step. Drawing TA and Tβ with desirable aspect ratios 1/m and 1/p, respectively, compensates for this rotation, and ensures that the drawings of TA and Tβ used to draw T have the desirable aspect ratios, m and p, respectively.

Next we draw recursively each nonempty partial tree Tk with Ak as its desirable aspect ratio, where the value of Ak is the one computed in the previous step. The base case for the recursion happens when Tk contains exactly one node, in which case, the drawing of Tk is simply the one consisting of exactly one node.
5.4 Compose Drawings

Let \( \Gamma_k \) denote the drawing of a partial tree \( T_k \) constructed in Step Draw Partial Trees. We now describe the construction of a feasible drawing \( \Gamma \) of \( T \) from the drawings of its partial trees in both Cases 1 and 2.

In Case 1, we first construct a feasible drawing \( \Gamma_\alpha \) of the partial tree \( T_\alpha \) by composing \( \Gamma_1 \) and \( \Gamma_2 \) as shown in Figure 5, then construct a feasible drawing \( \Gamma_B \) of \( T_B \) by composing \( \Gamma_\alpha \) and \( \Gamma_\beta \) as shown in Figure 6, and finally construct \( \Gamma \) by composing \( \Gamma_A \), \( \Gamma_B \) and \( \Gamma_C \) as shown in Figure 2.

\( \Gamma_\alpha \) is constructed as follows (see Figure 5): (Recall that if \( u \neq a \) then \( T_\alpha \) is the subtree of \( T \) rooted at \( u \), otherwise \( T_\alpha = T_2 \))

- If \( u \neq a \), and \( T_1 \neq \emptyset \) (see Figure 5(a)), then, if \( A < 1 \), place \( \Gamma_1 \) above \( \Gamma_2 \) such that the left boundary of \( \Gamma_1 \) is one unit to the right of the left boundary of \( \Gamma_2 \). Place \( u \) in the same vertical channel as \( v \) and in the same horizontal channel as \( c \). If \( A \geq 1 \), place \( \Gamma_1 \) one unit to the left of \( \Gamma_2 \), such that the top boundary of \( \Gamma_1 \) is one unit below the top boundary of \( \Gamma_2 \). Place \( u \) in the same vertical channel as \( c \) and in the same horizontal channel as \( v \). Draw edges \((u, c)\) and \((u, v)\).
- If \( u = a \), and \( T_1 = \emptyset \) (see Figure 5(b)), then, if \( A < 1 \), place \( u \) in the same horizontal channel and at one unit to the left of \( v \); otherwise (i.e. \( A \geq 1 \)), place \( u \) in the same vertical channel and at one unit above \( v \). Draw edge \((u, v)\).
- Otherwise (i.e., if \( u = a \), \( \Gamma_\alpha \) is the same as \( \Gamma_2 \) (see Figure 5(c)).

\( \Gamma_B \) is constructed as follows (see Figure 6): Let \( y \) be the root of \( T_\alpha \). Note that \( y = u \) if \( u \neq a \), and \( y = v \) otherwise.

- If \( T_\beta \neq \emptyset \) (see Figure 6(a)) then, if \( A < 1 \), then place \( \Gamma_\beta \) one unit above \( \Gamma_\alpha \) such that the left boundaries of \( \Gamma_\beta \) and \( \Gamma_\alpha \) are aligned; otherwise (i.e., if \( A \geq 1 \)), first rotate \( \Gamma_\beta \) by 90° and then flip it vertically, then place \( \Gamma_\beta \) one unit to the left of \( \Gamma_\alpha \) such that the top boundaries of \( \Gamma_\beta \) and \( \Gamma_\alpha \) are aligned. Draw edge \((w, y)\).
- Otherwise (i.e., if \( T_\beta = \emptyset \)), \( \Gamma_B \) is same as \( \Gamma_\alpha \) (see Figure 6(b)).

\( \Gamma \) is constructed from \( \Gamma_A \), \( \Gamma_B \), and \( \Gamma_C \) as follows (see Figure 2): Let \( x \) be the root of \( T_B \). Note that \( x = f \) if \( T_B \neq \emptyset \), and \( x = y \) otherwise.

- In Subcase (a), as shown in Figure 2(a), if \( A < 1 \), stack \( \Gamma_A \), \( \Gamma_B \), and \( \Gamma_C \) one above the other, such that they are separated by unit vertical distance from each other, and the left boundaries of \( \Gamma_A \) and \( \Gamma_C \) are aligned with each other and are placed at unit horizontal distance to the left of the left boundary of \( \Gamma_B \). If \( A \geq 1 \), then first rotate \( \Gamma_A \) by 90°, and then flip it vertically. Then, place \( \Gamma_A \), \( \Gamma_C \), and \( \Gamma_B \) from left-to-right in that order, separated by unit horizontal distances, such that the top boundaries of \( \Gamma_A \) and \( \Gamma_B \) are aligned, and are at unit vertical distance above the top boundary of \( \Gamma_C \). Then, move \( \Gamma_C \) down until \( u^* \) becomes the lowest node of \( \Gamma \). Place node \( a \) in the same vertical channel as \( d \) and in the same horizontal channel as \( r \) and \( x \). Draw edges \((e, a)\), \((a, x)\), and \((a, d)\).
- In Subcase (b), for both \( A < 1 \) and \( A \geq 1 \), place node \( r \) one unit above and left of the top boundary of \( \Gamma_B \) (see Figure 2(b)). Draw edge \((r, x)\).
- The drawing procedure for Subcase (c) is similar to the one in Subcase (a), except that we also flip \( \Gamma_C \) vertically (see Figure 2(c)).

- In Subcase (d), as shown in Figure 2(d), if \( A < 1 \), first flip \( \Gamma_B \) vertically, and then flip it horizontally, so that its root \( x \) gets placed at its lower-right corner. Then, first rotate \( \Gamma_A \) by 90°, and then flip it vertically. Next, place \( \Gamma_A \) above \( \Gamma_B \) with unit vertical separation, such that their left boundaries are aligned, next move node \( e \) (which is the link node of \( T_A \)) to the right until it is either to the right of, or aligned with the right boundary of \( \Gamma_B \) (since \( \Gamma_A \) is a feasible drawing of \( T_A \), by Property 2, as given in Section 4, moving \( e \) will not create any edge-crossings), and then place \( u^* \) in the same horizontal channel as \( x \) and one unit to the right of \( e \). If \( A \geq 1 \), first rotate \( \Gamma_A \) by 90°, and then flip it vertically. Then flip \( \Gamma_B \) vertically. Then, place \( \Gamma_A \), \( u^* \), and \( \Gamma_B \) left-to-right in that order separated by unit horizontal distances, such that the top boundaries of \( \Gamma_A \) and \( \Gamma_B \) are aligned, and \( u^* \) is placed in the same horizontal channel with the bottom boundary of the drawing among \( \Gamma_A \) and \( \Gamma_B \) with greater height. Draw edges \((u^*, e)\) and \((u^*, x)\).
- In Subcase (e), as shown in Figure 2(e), if \( A < 1 \), first flip \( \Gamma_B \) vertically, then place \( \Gamma_A \) and \( \Gamma_B \) one
Figure 5: Drawing $T_\alpha$, when: (a) $u \neq a$ and $T_1 \neq \emptyset$, (b) $u \neq a$ and $T_1 = \emptyset$, and (c) $u = a$. For each case, we first show the structure of $T_\alpha$ for that case, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. For simplicity, we have shown $\Gamma_1$ and $\Gamma_2$ as identically sized boxes, but in actuality, their sizes may be different.

Figure 6: Drawing $T_\beta$ when: (a) $T_\beta \neq \emptyset$, and (b) $T_\beta = \emptyset$. Node $y$ shown here is either node $u$ or $v$. For each case, we first show the structure of $T_\beta$ for that case, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. In Case (a), for simplicity, $w$ is shown to be in the interior of $\Gamma_\beta$, but actually, it is either same as $f$, or if $A < 1$ ($A \geq 1$), then is placed on the bottom (right) boundary of $\Gamma_\beta$. For simplicity, we have shown $\Gamma_\beta$ and $\Gamma_\alpha$ as identically sized boxes, but in actuality, their sizes may be different.

above the other with unit vertical separation, such that the left boundary of $\Gamma_A$ is at unit horizontal distance to the left of the left boundary of $\Gamma_B$. If $A \geq 1$, then first flip $\Gamma_B$ vertically, place $\Gamma_A$ to the left of $\Gamma_B$ at unit horizontal distance, such that their top boundaries are aligned. Next, move $\Gamma_B$ down until its bottom boundary is at least one unit below the bottom boundary of $\Gamma_A$. Place $u^*$ in the same vertical channel as $r$ and in the same horizontal channel as $x$. Draw edges $(r, u^*)$ and $(u^*, x)$. Note that, since $\Gamma_A$ is a feasible drawing of $T_A$, by Property 3 (see Section 4), drawing $(u^*, r)$ will not create any edge-crossings.

- The drawing procedure in Subcase (f) is similar to the one in Subcase (a), except that we do not have $\Gamma_A$ here (see Figure 2(f)).
- The drawing procedure in Subcase (g) is similar to the one in Subcase (f), except that we also flip $\Gamma_C$ vertically (see Figure 2(g)).

In Case 2, we construct $\Gamma$ by composing $\Gamma_A$, $\Gamma_B$, and $\Gamma_C$, as follows (see Figure 3):

- The drawing procedures in Subcases (a) and (c) are similar to those in Subcases (a) and (f), respectively, of Case 1 (see Figures 3(a,c)).
- The drawing procedure in Subcase (b) is similar to that in Case (b) of drawing $T_\alpha$ (see Figure 5(b)).
- In Subcase (d), as shown in Figure 3(d), if $A > 1$, we place $\Gamma_A$ above $\Gamma_C$, separated by unit vertical distance such that the left boundary of $\Gamma_C$ is one unit to the right of the left boundary of $\Gamma_A$. Place $u$ in the same vertical channel as $r$ and in the same horizontal channel as $v$. If $A \geq 1$, then first rotate $\Gamma_A$ by $90^\circ$, and then flip it vertically. Then, place $\Gamma_A$ to the left of $\Gamma_C$, separated by unit horizontal distance, such that the top boundary of $\Gamma_C$ is one unit below the top boundary of $\Gamma_A$. Then, move $\Gamma_C$ down until $u^*$ becomes the lowest node of $\Gamma$. Place $u$ in the same vertical channel as $v$ and in the same horizontal channel as $r$. Draw edges $(u, v)$ and $(u, e)$.
- The drawing procedures in Subcases (e), (f), (g), and (h) are similar to those in Subcases (a), (b), (c), and (d), respectively, (see Figures 3(e,f,g,h)), except that we also flip $\Gamma_C$ vertically.

5.5 Proof of Correctness

Lemma 1 (Planarity) Given a binary tree $T$ with a link node $u^*$, Algorithm DrawTree will construct a feasible drawing $\Gamma$ of $T$. 

10
Proof: We can easily prove using induction over the number of nodes $n$ in $T$ that $\Gamma$ is a feasible drawing:

**Base Case** ($n = 1$): $\Gamma$ consists of exactly one node and is trivially a feasible drawing.

**Induction** ($n > 1$): Consider Case 1. By the inductive hypothesis, the drawing constructed of each partial tree of $T$ is a feasible drawing.

Hence, from Figure 5, it can be easily seen that the drawing $\Gamma_a$ of $T_a$ is also a feasible drawing.

From Figure 6, it can be easily seen that the drawing $\Gamma_B$ of $T_B$ is also a feasible drawing. Note that because $\Gamma_2$ is a feasible drawing of $T_B$ and $w$ is its link node, $w$ is either at the bottom of $\Gamma_2$ (from Property 2, see Section 4), or at the top-left corner of $\Gamma_2$ and no other edge or node of $T_B$ is placed on, or crosses the vertical channel occupied by it (Properties 1 and 3, see Section 4). Hence, in Figure 6(a), in the case $A < 1$, drawing edge $(w, x)$ will not cause any edge crossings. Also, in Figure 6(a), in the case $A \geq 1$, drawing edge $(w, x)$ will not cause any edge crossings because after rotating $\Gamma_2$ by 90° and flipping it vertically, $w$ will either be at the right boundary of $\Gamma_2$ (see Property 2), or at the top-left corner of $\Gamma_2$ and no other edge or node of $T_B$ will be placed on, or cross the horizontal channel occupied by it (see Properties 1 and 3).

Finally, by considering each of the seven subcases shown in Figure 2 one-by-one, we can show that $\Gamma$ is also a feasible drawing of $T$:

- **Subcase (a):** See Figure 2(a). $\Gamma_A$ is a feasible drawing of $T_A$ and $e$ is the link node of $T_A$. Hence, $e$ is either at the bottom of $\Gamma_A$ (from Property 2), or is at the top-left corner of $\Gamma_A$, and no other edge or node of $T_A$ is placed on, or crosses the horizontal and vertical channels occupied by it (from Properties 1 and 3). Hence, in the case $A < 1$, drawing edge $(e, a)$ will not create any edge-crossings, and $\Gamma$ will also be a feasible drawing of $T$. In the case $A \geq 1$ also, drawing edge $(e, a)$ will not create any edge-crossings because after rotating $\Gamma_A$ by 90° and flipping it vertically, $e$ will either be at the right boundary of $\Gamma_A$ (see Property 2), or at the top-left corner of $\Gamma_A$ and no other edge or node of $T_A$ will be placed on, or cross the horizontal channel occupied by it (see Properties 1 and 3). Thus, for the case $A \geq 1$ also, $\Gamma$ will also be a feasible drawing of $T$.

- **Subcase (b):** See Figure 2(b). Because $\Gamma_B$ is a feasible drawing of $T_B$, it is straightforward to see that $\Gamma$ is also a feasible drawing of $T$ for both the cases when $A < 1$ and $A \geq 1$.

- **Subcase (c):** See Figure 2(c). The proof is similar to the one for Subcase (a).

- **Subcase (d):** See Figure 2(d). $\Gamma_A$ is a feasible drawing of $T_A$, $e$ is the link node of $T_A$, and $e \neq r$. Hence, from Property 2, $e$ is located at the bottom of $\Gamma_A$. Rotating $\Gamma_A$ by 90° and flipping it vertically will move $e$ to the right boundary of $\Gamma_A$. Moving $e$ to the right until it is either to the right of, or aligned with the right boundary of $\Gamma_B$ will not cause any edge-crossings because of Property 2. It can be easily seen that in both the cases, $A < 1$ and $A \geq 1$, drawing edge $(e, u^*)$ does not create any edge-crossings, and $\Gamma$ is a feasible drawing of $T$.

- **Subcase (e):** See Figure 2(e). $\Gamma_A$ is a feasible drawing of $T_A$, $e$ is the link node of $T_A$, and $e = r$. Hence, from Properties 1 and 3, $e$ is at the top-left corner of $\Gamma_A$, and no other edge or node of $T_A$ is placed on, or crosses the horizontal and vertical channels occupied by it. Hence, in both the cases, $A < 1$ and $A \geq 1$, drawing edge $(e, u^*)$ will not create any edge-crossings, and $\Gamma$ is a feasible drawing of $T$.

- **Subcase (f):** See Figure 2(f). It is straightforward to see that $\Gamma$ is a feasible drawing of $T$ for both the cases when $A < 1$ and $A \geq 1$.

- **Subcase (g):** See Figure 2(g). $\Gamma_C$ is a feasible drawing of $T_C$, $u^*$ is the link node of $T_C$, and $u^*$ is also the root of $T_C$. Hence, from Properties 1 and 3, $u^*$ is at the top-left corner of $\Gamma_C$, and no other edge or node of $T_C$ is placed on, or crosses the horizontal and vertical channels occupied by it. Flipping $\Gamma_C$ vertically will move $u^*$ to the bottom-left corner of $\Gamma_C$ and no other edge or node of $T_C$ will be on or crosses the vertical channel occupied by it. Hence, drawing edge $(r, u^*)$ will not create any edge-crossings, and $\Gamma$ will be a feasible drawing of $T$.

Using a similar reasoning, we can show that in Case 2 also, $\Gamma$ is a feasible drawing of $T$. 

\[\square\]

**Lemma 2 (Time)** Given an $n$-node binary tree $T$ with a link node $u^*$, Algorithm DrawTree will construct a drawing $\Gamma$ of $T$ in $O(n \log n)$ time.

**Proof:** From Theorem 1, each partial tree into which Algorithm DrawTree would split $T$ will have at most $(2/3)n$ nodes in it. Hence, it follows that the depth of the recursion for Algorithm DrawTree is $O(\log n)$. At the first recursive level, the algorithm will split $T$ into partial trees, assign aspect ratios to the partial
trees and compose the drawings of the partial trees to construct a drawing of $T$. At the next recursive level, it will split all of these partial trees into smaller partial trees, assign aspect ratios to these smaller partial trees, and compose the drawings of these smaller partial trees to construct the drawings of all the partial trees. This process will continue until the bottommost recursive level is reached. At each recursive level, the algorithm takes $O(m)$ time to split a tree with $m$ nodes into partial trees, assign aspect ratios to the partial trees, and compose the drawings of partial trees to construct a drawing of the tree. At each recursive level, the total number of nodes in all the trees that the algorithm considers for drawing is at most $n$. Hence, at each recursive level, the algorithm totally spends $O(n)$ time. Hence, the running time of the algorithm is $O(n \cdot O(\log n)) = O(n \log n)$.

In Lemma 4 given below, we prove that the algorithm will draw the tree in $O(n)$ area. Note that the proof given below is different from the one given in [6], which used Theorem 6 of [11]. We believe that the proof given below is more straightforward, and easier to understand.

**Lemma 3** Let $R$ be a rectangle with area $D$ and aspect ratio $A$. Let $W$ and $H$ be the width and height, respectively, of $R$. Then, $W = \sqrt{AD}$ and $H = \sqrt{D/A}$.

**Proof:** By the definition of aspect ratio, $A = W/H$. $D = WH = W(W/A) = W^2/A$. Hence, $W = \sqrt{AD}$. $H = W/A = \sqrt{AD}/A = \sqrt{D/A}$.

**Lemma 4** (Area) Let $T$ be a binary tree with a link node $u^*$. Let $n$ be the number of nodes in $T$. Let $\epsilon$ and $A$ be two numbers such that $0 < \epsilon < 1$, and $A$ is in the range $[n^{\epsilon}, n]$. Given $T$, $\epsilon$, and $A$ as input, Algorithm DrawTree will construct a drawing $\Gamma$ of $T$ that can fit inside a rectangle $R$ with $O(n)$ area and aspect ratio $A$.

**Proof:** Let $D(n)$ be the area of $R$. We will prove, using induction over $n$, that $D(n) = O(n)$. More specifically, we will prove that $D(n) \leq c_1n - c_2n^\beta$ for all $n \geq n_0$, where $c_0, c_1, c_2, \beta$ are some positive constants and $\beta < 1$.

We now give the proof for the case when $A \geq 1$ (the proof for the case $A < 1$ is symmetrical). Algorithm DrawTree will split $T$ into at most 5 partial trees. Let $T_k$ be a non-empty partial tree of $T$, where $T_k$ is one of $T_A, T_B, T_1, T_2, T_C$ in Case 1, and is one of $T_A, T_B, T_C$ in Case 2. Let $n_k$ be the number of nodes in $T_k$, and let $x_k = n_k/n$. Let $P_k = c_1n - c_2n^\beta/x_k^\beta$. From Theorem 1, it follows that $n_k \leq (2/3)n$, and hence, $x_k \leq 2/3$. Hence, $P_k \leq c_1n - c_2n^\beta/(2/3)^{1-\beta} = c_1n - c_2n^\beta(3/2)^{1-\beta}$. Let $P' = c_1n - c_2n^\beta(3/2)^{1-\beta}$. Thus, $P_k \leq P'$.

From the inductive hypothesis, Algorithm DrawTree will construct a drawing $\Gamma_k$ of $T_k$ that can fit inside a rectangle $R_k$ with aspect ratio $A_k$ and area $D(n_k)$, where $A_k$ is as defined in Section 5.2, and $D(n_k) \leq c_1n_k - c_2n_k^\beta$. Since $x_k = n_k/n$, $D(n_k) \leq c_1n_k - c_2n_k^\beta = c_1x_kn - c_2(x_kn)^\beta = x_k(c_1n - c_2n^\beta/x_k^\beta) = x_kP'$.

Let $W_k$ and $H_k$ be the width and height, respectively, of $R_k$. We now compute the values of $W_k$ and $H_k$ in terms of $A$, $P'$, $x_k$, $n$, and $\epsilon$. We have two cases:

- **$T_k$ is a small partial tree of $T$:** Then, $n_k < (n/A)^{1/(1+\epsilon)}$, and also, as explained in Section 5.2, $A_k = 1/n_k$. From Lemma 3, $W_k = \sqrt{A_kD(n_k)} \leq \sqrt{1/n_k}(x_kP') = \sqrt{(1/n_k)(n_k/n)P'} = \sqrt{n_k^{1-\epsilon}P'/n}$. Since $n_k < (n/A)^{1/(1+\epsilon)}$, $W_k < \sqrt{(n/A)^{1-\epsilon}/(1+\epsilon)}P'/n = \sqrt{1/A^{1-\epsilon}/(1+\epsilon)}P'/n^{2\epsilon/(1+\epsilon)} \leq P'/n^{2\epsilon/(1+\epsilon)}$ since $A \geq 1$.

- **$T_k$ is a large partial tree of $T$:** Then, as explained in Section 5.2, $A_k = x_kA$. From Lemma 3, $W_k = \sqrt{A_kD(n_k)} \leq \sqrt{x_kA(x_kP')} = x_k\sqrt{AP'}$.

From Lemma 3, $H_k = \sqrt{D(n_k)/A_k} \leq \sqrt{x_kP'/(n_k)} = (n_k/n)P'n_k^{\beta/2} = n_k^{\beta/2}P'/n$. Since $n_k < (n/A)^{1/(1+\epsilon)}$, $H_k < \sqrt{(n/A)^{1-\epsilon}/(1+\epsilon)}P'/n = \sqrt{n/A}P'/n = \sqrt{P'/A}$.

In Step Compose Drawings, we use at most two additional horizontal channels and at most one additional vertical channel while combining the drawings of the partial trees to construct a drawing $\Gamma$ of $T$. Hence, $\Gamma$ can fit inside a rectangle $R'$ with width $W'$ and height $H'$, respectively, where,

$$H' \leq \max_H \{H_k\} + 2 \leq \sqrt{P'/A} + 2,$$
and
\[
W' \leq \sum_{T_k \text{ is a large partial tree}} W_k + \sum_{T_k \text{ is a small partial tree}} W_k + 1
\]
\[
\leq \sum_{T_k \text{ is a large partial tree}} x_k \sqrt{AP'} + \sum_{T_k \text{ is a small partial tree}} \sqrt{P'/n^{2/(1+\epsilon)}} + 1
\]
\[
\leq \sqrt{AP'} + 5 \sqrt{P'/n^{2/(1+\epsilon)}} + 1
\]
(because $\sum_{T_k \text{ is a large partial tree}} x_k \leq 1$, and $T$ has at most 5 partial trees)

$R'$ does not have aspect ratio equal to $A$, but it is contained within a rectangle $R$ with aspect ratio $A$, area $D(n)$, width $W$, and height $H$, where
\[
W = \sqrt{AP'} + 5 \sqrt{P'/n^{2/(1+\epsilon)}} + 1 + 2A,
\]
and
\[
H = \sqrt{P'/A} + 2 + (5/A) \sqrt{P'/n^{2/(1+\epsilon)}} + 1/A
\]
Hence, $D(n) = WH = (\sqrt{AP'} + 5 \sqrt{P'/n^{2/(1+\epsilon)}} + 1 + 2A)(\sqrt{P'/A} + 2 + (5/A) \sqrt{P'/n^{2/(1+\epsilon)}} + 1/A) \leq
\]
\[
P' + c_3 P' / \sqrt{An^{2(1+\epsilon)}} + c_4 \sqrt{AN} P' + c_5 P' / (An^{2(1+\epsilon)}) + c_6 \sqrt{P'/n^{2/(1+\epsilon)}} + c_7 A + c_9 + c_9 / A + c_{10} \sqrt{P'/A} + c_{11} \sqrt{P'/n^{2(1+\epsilon)}} / A,
\]
where $c_3, c_4, \ldots, c_{11}$ are some constants.

Since, $1 \leq A \leq n^\epsilon$, we have that
\[
D(n) \leq P' + c_3 P' / \sqrt{n^{2/(1+\epsilon)}} + c_4 \sqrt{n^\epsilon} P' + c_5 P' / n^{2(1+\epsilon)} + c_6 \sqrt{P'/n^{2/(1+\epsilon)}} + c_7 n^\epsilon + c_8
\]
\[
+ c_9 + c_{10} \sqrt{P'} + c_{11} \sqrt{P'/n^{2(1+\epsilon)}},
\]
Since $P' < c_1 n$,
\[
D(n) \leq P' + c_3 c_1 n / \sqrt{n^{2(1+\epsilon)}} + c_4 \sqrt{n^\epsilon} c_1 n + c_5 c_1 n / n^{2(1+\epsilon)} + c_6 \sqrt{c_1 n / n^{2(1+\epsilon)}} + c_7 n^\epsilon + c_8
\]
\[
+ c_9 + c_{10} \sqrt{c_1 n} / n^{1/2} + c_{11} \sqrt{c_1 n / n^{2(1+\epsilon)}}
\]
\[
\leq P' + c_3 c_1 n^{1+\epsilon} + c_4 c_1 n^{1+\epsilon}/2 + c_5 c_1 n^{1-\epsilon}/(1+\epsilon) + c_6 \sqrt{c_1 n^{1-\epsilon}/(2(1+\epsilon))} + c_7 n^\epsilon + c_8
\]
\[
+ c_9 + c_{10} \sqrt{c_1 n} / n^{1/2} + c_{11} \sqrt{c_1 n^{1-\epsilon}/(2(1+\epsilon))}
\]
\[
\leq P' + c_2 n^{1+\epsilon} + c_3 n^{1+\epsilon}/2
\]
where $c_2$ and $c_3$ are some constants (because, since $0 < \epsilon < 1$, $(1-\epsilon)/(2(1+\epsilon)) < (1-\epsilon)/(1+\epsilon) < 1/(1+\epsilon)$, $\epsilon < (1+\epsilon)/2$, and $1/2 < (1+\epsilon)/2$).

$P' = c_3 n - c_9 n^\beta(3/2)^{1-\beta} = c_3 n - c_9 n^\beta(1 + c_{14})$, where $c_{14}$ is a constant such that $1 + c_{14} = (3/2)^{1-\beta}$.

Hence, $D(n) \leq c_1 n - c_2 n^\beta(1 + c_{14}) + c_1 n^{1+\epsilon} + c_{12} n^{1+\epsilon}/2 = c_1 n - c_2 n^\beta - (c_{14} n^\beta - c_{12} n^{1+\epsilon}/2)$. Thus, for a large enough constant $n_0$, and large enough $\beta$, where $1 > \beta > \max\{1/(1+\epsilon), (1+\epsilon)/2\}$, for all $n \geq n_0$, $c_{14} n^\beta - c_{12} n^{1+\epsilon}/2 - c_{13} n^{1+\epsilon}/2 \geq 0$, and hence $D(n) \leq c_1 n - c_2 n^\beta$.

The proof for the case $A < 1$ uses the same reasoning as for the case $A \geq 1$. With $T_k, R_k, W_k, H_k, R', W', H'$, $R, W$, and $H$ defined as above, and $A_k$ as defined in Section 5.2, we get the following values for $W_k, H_k, W', H', W, H,$ and $D(n)$:
\[
W_k \leq \sqrt{AP'}
\]
\[
H_k \leq \sqrt{P'/n^{2/(1+\epsilon)}} \text{ if } T_k \text{ is a small partial tree}
\]
\[
\leq x_k \sqrt{P'/A} \text{ if } T_k \text{ is a large partial tree}
\]
\[
W' \leq \sqrt{AP'} + 2
\]
\[
H' \leq \sqrt{P'/A} + 5 \sqrt{P'/n^{2/(1+\epsilon)}} + 1
\]
\[
\begin{align*}
W & \leq \sqrt{AP^2 + 2} + 5A\sqrt{P^2/n}^{2/(1+\epsilon)} + A \\
H & \leq \sqrt{P^2/A} + 5\sqrt{P^2/n}^{2/(1+\epsilon)} + 1 + 2/A \\
D(n) & \leq P' + c_{12}n^{\beta/(1+\epsilon)} + c_{13}n^{(1+\epsilon)/2}
\end{align*}
\]

where \(c_{12}\) and \(c_{13}\) are the same constants as in the case \(A \geq 1\). Therefore, \(D(n) \leq c_1 n - c_2 n^{\beta}\) for \(A < 1\) too. (Notice that in the values that we get above for \(W_k, H_k, W', H', W,\) and \(H,\) if we replace \(A\) by \(1/A,\) exchange \(W_k\) with \(H_k,\) exchange \(W'\) with \(H',\) and exchange \(W\) with \(H,\) we will get the same values for \(W_k, H_k, W', H', W,\) and \(H\) as in the case \(A \geq 1.\) This basically reflects the fact that the cases \(A \geq 1\) and \(A < 1\) are symmetrical to each other.)

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

**Proof:** Let \(\epsilon\) be any constant such that \(n^{-\epsilon} \leq A \leq n^\epsilon\) and \(0 < \epsilon < 1.\) Designate any leaf of \(T\) as its link node. Construct a drawing \(\Gamma\) of \(T\) in \(R\) by calling Algorithm `DrawTree` with \(T, A\) and \(\epsilon\) as input. From Lemmas 1, 2, and 4, \(\Gamma\) will be a planar straight-line grid drawing of \(T\) contained entirely within a rectangle with \((\mathcal{O}(n)\) area, and aspect ratio \(A.\)

**Corollary 1** Let \(T\) be a binary tree with \(n\) nodes. We can construct in \(O(n \log n)\) time, a planar straight-line grid drawing of \(T\) with optimal (equal to \(O(n)\)) area, and optimal aspect ratio (equal to 1).

**Proof:** Immediate from Theorem 2, with \(A = 1.\)

### 6 Experimental Results

We have implemented the algorithm using C++. The implementation consists of about 2100 lines of code. We have also experimentally evaluated the algorithm on two types of binary trees, namely, randomly-generated, consisting of up to 50,000 nodes, and complete, consisting of up to 65,535 = \(2^{16}\) − 1 nodes.

Each randomly-generated binary tree \(T_n\) with \(n\) nodes was generated by generating a sequence \(T_0, T_1, \ldots, T_n\) of binary trees, where \(T_0\) is the empty tree, and \(T_i\) was generated from \(T_{i-1}\) by inserting a new leaf \(v_i\) into it. The position, where \(v_i\) is inserted in \(T_{i-1},\) is determined by traversing a path \(p = u_0 u_1 \ldots u_n\) of \(T_{i-1},\) where \(u_0\) is the root of \(T_{i-1},\) and \(u_n\) has at most one child. More precisely, we start at the root \(u_0,\) and in the general step, assuming that we have already traversed the subpath \(u_0 u_1 \ldots u_{i-1},\) we flip a coin. If “head” comes up, then if \(u_{i-1}\) has a left child \(c,\) then we set \(u_i = c,\) and move to \(u_i,\) otherwise we make \(v_i\) the left child of \(u_{i-1},\) and stop. If “tail” comes up, then if \(u_{i-1} \) has a right child \(c,\) then we set \(u_i = c,\) and move to \(u_i,\) otherwise we make \(v_i\) the right child of \(u_{i-1},\) and stop.

Recall that the algorithm takes three values as input: a binary tree \(T\) with \(n\) nodes, a number \(\epsilon,\) 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 a tree \(T,\) and the number of nodes in \(T.\) Recall that the area and aspect ratio of a drawing is defined as the area and aspect ratio, respectively, of its enclosing rectangle.

To evaluate the algorithm, we varied \(n\) up to 50,000 for randomly-generated trees, and up to \(65,535 = 2^{16} - 1\) for complete trees. For each \(n,\) we used five different values for \(\epsilon,\) namely, 0.1, 0.25, 0.5, 0.75, and 0.9. For each \((n, \epsilon)\) 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 \(\epsilon\) only from 1 through \(n^\epsilon,\) not from \(n^{-\epsilon}\) through \(n^\epsilon\) (the only difference between \(A < 1\) and \(A > 1\) is that for \(A < 1\) the algorithm constructs drawings with longer height than width, whereas for \(A > 1,\) it constructs drawings with longer width than height). Hence, in the rest of the section, we will assume that \(A \geq 1.\) For each type of tree (randomly-generated and complete), and for each triplet \((n, A, \epsilon),\) 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 \((n, A, \epsilon)\) 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 7, and Figure 8 show how \(c\) varies with \(n, A,\) and \(\epsilon\) for randomly-generated, and complete trees, respectively.
We also discovered that $c$ increases with $A$ for a given $n$ and $\epsilon$. However, the rate of increase is very small. Consequently, for a given $n$ and $\epsilon$, the range for $c$ over all the values of $A$ is small (see Figure 7(b,d,f,h,j), and Figure 8(b,d,f,h,j)). For example, 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 $\epsilon$.

Finally, we would like to comment that the aspect ratio of the drawing constructed is, in general, different from the input aspect ratio $A$. We computed the ratio $r$ of the aspect ratio of the drawing constructed by the algorithm and input aspect ratio $A$. 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 $A = n'$. However, we also discovered that for a large range of values for $A$, namely, $[1, \min\{n', n/\log^2 n\}]$, $r$ stays within the range $[0.8, 1.5]$, and so is close to 1. Hence, even in applications, that require the drawing to be of exactly the same aspect ratio as $A$, we can obtain a drawing with small area and aspect ratio exactly equal to $A$ by adding enough “white space” to the drawing constructed by our drawing algorithm. Adding the white space will increase the area of the drawing 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 the drawing will still be small.

References


(a) $\epsilon = 0.9$

(b) $\epsilon = 0.9$

(c) $\epsilon = 0.75$

(d) $\epsilon = 0.75$

(e) $\epsilon = 0.5$

(f) $\epsilon = 0.5$
Figure 7: Performance of the algorithm, as given by the value of $c$, for drawing a randomly-generated binary tree $T$ with different values of $A$ and $\epsilon$, where $c = \text{area of drawing/number of nodes} n$ in $T$: (a) $\epsilon = 0.9$, (c) $\epsilon = 0.75$, (e) $\epsilon = 0.5$, (g) $\epsilon = 0.25$, and (i) $\epsilon = 0.1$. Figures (b), (d), (f), (h), and (j) contain the projections on the $XZ$-plane of the plots shown in Figures (a), (c), (e), (g), and (i), respectively, and show for each $\epsilon$, the ranges for the values of $c$ for different values of $A$ for each $n$. 
\( c = \frac{\text{area}}{n} \)

(a) \( \epsilon = 0.9 \)

(b) \( \epsilon = 0.9 \)

(c) \( \epsilon = 0.75 \)

(d) \( \epsilon = 0.75 \)

(e) \( \epsilon = 0.5 \)

(f) \( \epsilon = 0.5 \)
Figure 8: Performance of the algorithm, as given by the value of $c$, for drawing a complete binary tree $T$ with different values of $A$ and $\epsilon$, where $c=\text{area of drawing/number of nodes } n$ in $T$: (a) $\epsilon = 0.9$, (c) $\epsilon = 0.75$, (e) $\epsilon = 0.5$, (g) $\epsilon = 0.25$, and (i) $\epsilon = 0.1$. Figures (b), (d), (f), (h), and (j) contain the projections on the $XZ$-plane of the plots shown in Figures (a), (c), (e), (g), and (i), respectively, and show for each $\epsilon$, the ranges for the values of $c$ for different values of $A$ for each $n$. 