

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

Ashim Garg                      Adrian Rusu<sup>†</sup>

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  $[n^{-\epsilon}, n^\epsilon]$ , where  $\epsilon$  is any constant such that  $0 < \epsilon < 1$ . 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 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.

## 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.

---

\*Research supported by NSF CAREER Award No. 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.

<sup>†</sup>Current Affiliation: Department of Computer Science, Rowan University, Glassboro, NJ 08028. Email: rusu@rowan.edu

## 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. 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  $[n^{-\epsilon}, n^\epsilon]$ , where  $\epsilon$  is any constant, such that  $0 < \epsilon < 1$ , 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).

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 [10]). 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 [8].

However, note that the  $O(n \log \log n)$  area drawing constructed by the algorithms of [1] and [8] 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  $[n^{-\epsilon}, n^\epsilon]$ , where  $\epsilon$  is any constant, such that  $0 < \epsilon < 1$ . [7] and [10] 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. [8] 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, 9] ([3], [2], respectively) give algorithms for constructing an upward straight-line drawing of  $T$  with  $O(n)$  area.

Table 1 summarizes these results.

Tree Type	Drawing Type	Area	Aspect Ratio	Reference
Fibonacci	Upward Straight-line	$O(n)$	$\theta(1)$	[2, 9]
AVL	Upward Straight-line	$O(n)$	$\theta(1)$	[3]
Complete Binary	Upward Straight-line	$O(n)$	$\theta(1)$	[2]
General Binary	Upward Orthogonal Polyline	$O(n \log \log n)$	$\theta(\log^2 n / (n \log \log n))$	[5, 8]
	(Non-upward) Orthogonal Polyline	$O(n)$	$\theta(1)$	[7, 10]
	Upward Orthogonal Straight-line	$O(n \log n)$	$[\log n/n, n/\log n]$	[1]
	Upward Polyline	$O(n)$	$[n^{-\epsilon}, n^\epsilon]$	[5]
	Upward Straight-line	$O(n \log \log n)$	$\theta(\log^2 n / (n \log \log n))$	[8]
	(Non-upward) Straight-line	$O(n)$	$[n^{-\epsilon}, n^\epsilon]$	[6] and <i>this paper</i>

**Table 1:** Bounds on the areas and aspect ratios of various kinds of planar grid drawings of an  $n$ -node binary tree. Here,  $\epsilon$  is an arbitrary constant, such that  $0 < \epsilon < 1$ .

### 4 Preliminaries

Throughout the paper, by the term *tree*, we will mean a rooted tree, i.e., a tree with a given root. We will assume that the plane is covered by an infinite rectangular grid. A *horizontal channel* (*vertical channel*) is a line parallel to the  $X$ -( $Y$ -)axis, passing through grid-points.

Let  $T$  be a tree with root  $o$ . 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. The *subtree* of  $T$  rooted at a node  $v$  consists of  $v$  and all the descendants of  $v$ .  $T$  is the *empty tree*, i.e.,  $T = \emptyset$ , if it has zero nodes in it.

Let  $\Gamma$  be a drawing of  $T$ . By the *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  $[n^{-\epsilon}, n^\epsilon]$ , where  $0 < \epsilon < 1$  is a constant.

Let  $w$  be a node of an ordered tree. We denote by  $p(w)$ ,  $l(w)$ ,  $r(w)$ , and  $s(w)$ , respectively, the parent, left child, right child, and sibling of  $w$ .

For some trees, we will designate a special *link* node  $u^*$ , that has at most one child. As we will see later in Section 5, the link node helps in combining the drawing of a tree with the drawing of another tree to obtain a drawing of a larger tree, that contains both the trees.

Let  $T$  be a tree with link node  $u^*$ . Let  $o$  be the root of  $T$ . A planar straight-line grid drawing  $\Gamma$  of  $T$  is a *feasible* drawing of  $T$ , if it has the following three properties:

- **Property 1:** The root  $o$  is placed at the top-left corner of  $\Gamma$ .
- **Property 2:** If  $u^* \neq o$ , 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  $u^* = o$ , then no other node or edge of  $T$  is placed on, or crosses the vertical and horizontal channels occupied by  $o$ . Moreover, we can move  $u^*$  (i.e.,  $o$ ) upwards in its vertical channel by any distance without causing any edge-crossings in  $\Gamma$ .

The following Theorem paraphrases a theorem of Valiant [10]:

**Theorem 1 (Separator Theorem [10])** *Every binary tree  $T$  with  $n$  nodes, where  $n \geq 2$ , contains an edge  $e$ , called a separator edge, such that removing  $e$  from  $T$  splits it into two non-empty trees with  $n_1$  and  $n_2$  nodes, respectively, such that for some  $x$ , where  $1/3 \leq x \leq 2/3$ ,  $n_1 = xn$ , and  $n_2 = (1-x)n$ . Moreover,  $e$  can be found in  $O(n)$  time.*

Let  $\Gamma$  be a drawing of  $T$ . Let  $v$  be a node of  $T$  located at grid point  $(i, j)$  in  $\Gamma$ . Assume that the root  $o$  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  $o$ . 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 about the  $X$ - or  $Y$ -axis, respectively. 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  $(i, -j)$ .

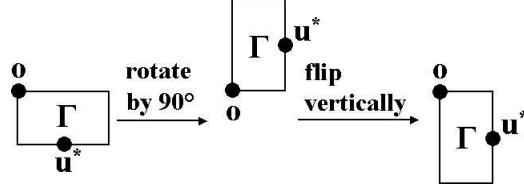
Suppose  $\Gamma$  were a feasible drawing, where the link node  $u^*$  was placed at the bottom of  $\Gamma$ . On applying a rotation operation followed by a vertical-flip operation,  $u^*$  will get relocated to the right-boundary of  $\Gamma$ , but  $o$  will continue to stay at the top-left corner (see Figure 1). We will use this fact later in Section 5 in the *Compose Drawings* step of our drawing algorithm.

## 5 Our Tree Drawing Algorithm

Let  $T$  be a binary tree with link node  $u^*$ . Let  $n$  be the number of nodes in  $T$ . Let  $A$  and  $\epsilon$  be two numbers, where  $\epsilon$  is a constant, 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  $\epsilon$ ,  $A$ , and  $T$  as input, and uses a simple divide-and-conquer strategy to recursively construct a feasible drawing  $\Gamma$  of  $T$ , by performing the following actions at each recursive step:

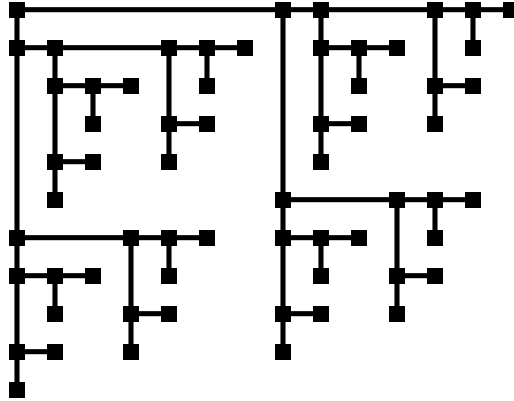
- *Split Tree:* Split  $T$  into at most five partial trees by removing at most two nodes and their incident edges from it. Each partial tree has at most  $(2/3)n$  nodes. Based on the arrangement of these partial trees within  $T$ , we get two cases, which are shown in Figures 3 and 4, respectively, and described later in Section 5.1.
- *Assign Aspect Ratios:* Correspondingly, assign a desirable aspect ratio  $A_k$  to each partial tree  $T_k$ . The value of  $A_k$  is based on the value of  $A$  and the number of nodes in  $T_k$ .



**Figure 1:** Rotating a drawing  $\Gamma$  by  $90^\circ$ , followed by flipping it vertically. Note that initially node  $u^*$  was located at the bottom boundary of  $\Gamma$ , but after the rotate operation,  $u^*$  is at the right boundary of  $\Gamma$ .

- *Draw Partial Trees:* Recursively construct a feasible drawing 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, such that the drawing  $\Gamma$  of  $T$  is a feasible drawing. Note that the arrangement of these drawings is done based on the cases shown in Figures 3 and 4. 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.

*Remark:* The drawing  $\Gamma$  constructed by the algorithm may not have aspect ratio exactly equal to  $A$ , but as we will prove later in Lemma 4, it will fit inside a rectangle with area  $O(n)$  and aspect ratio  $A$ .



**Figure 2:** Drawing of the complete binary tree with 63 nodes constructed by Algorithm *DrawTree*, with  $A = 1$  and  $\epsilon = 0.5$ .

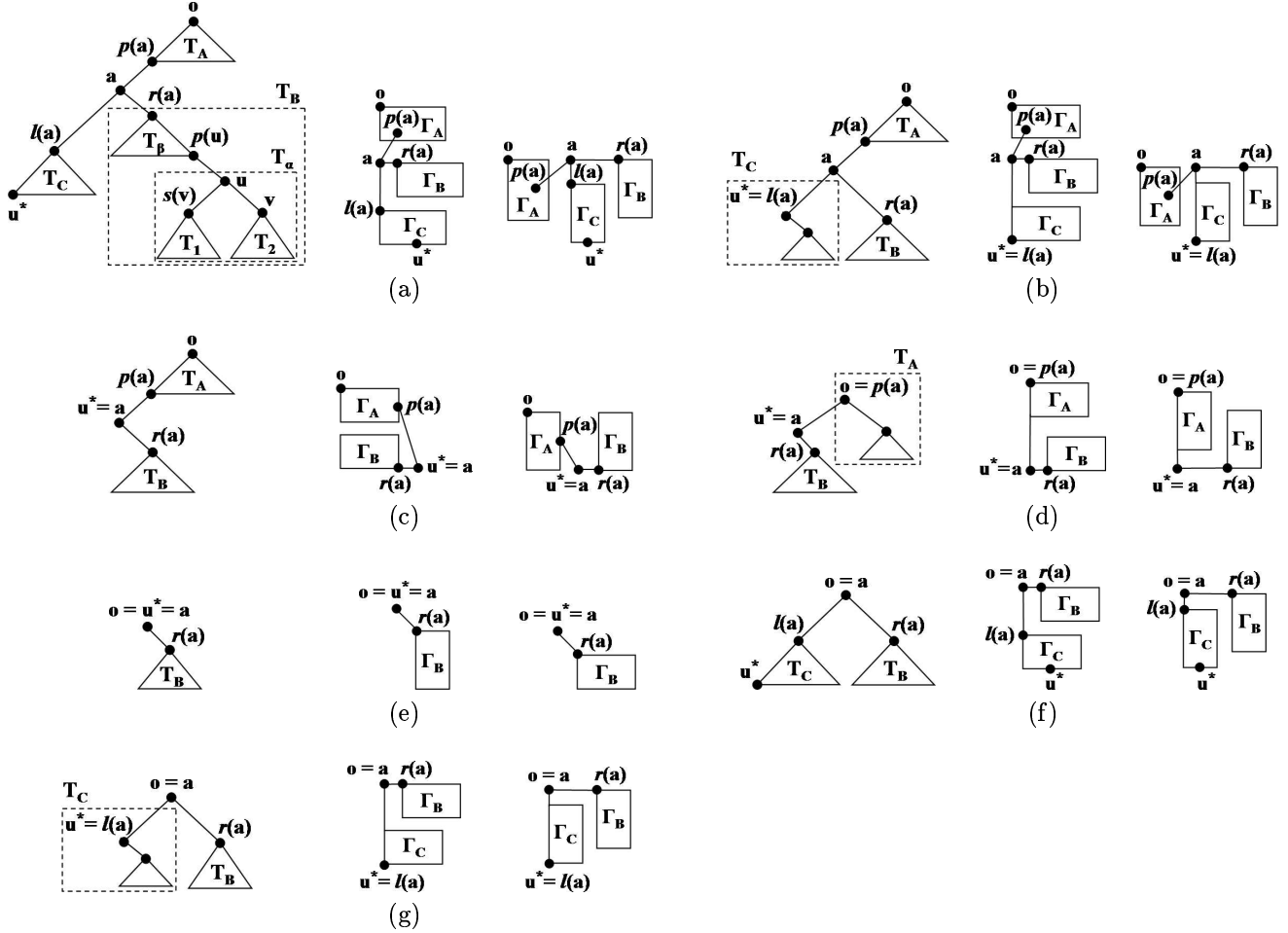
Figure 2 shows a drawing of a complete binary tree with 63 nodes constructed by Algorithm *DrawTree*, with  $A = 1$  and  $\epsilon = 0.5$ .

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

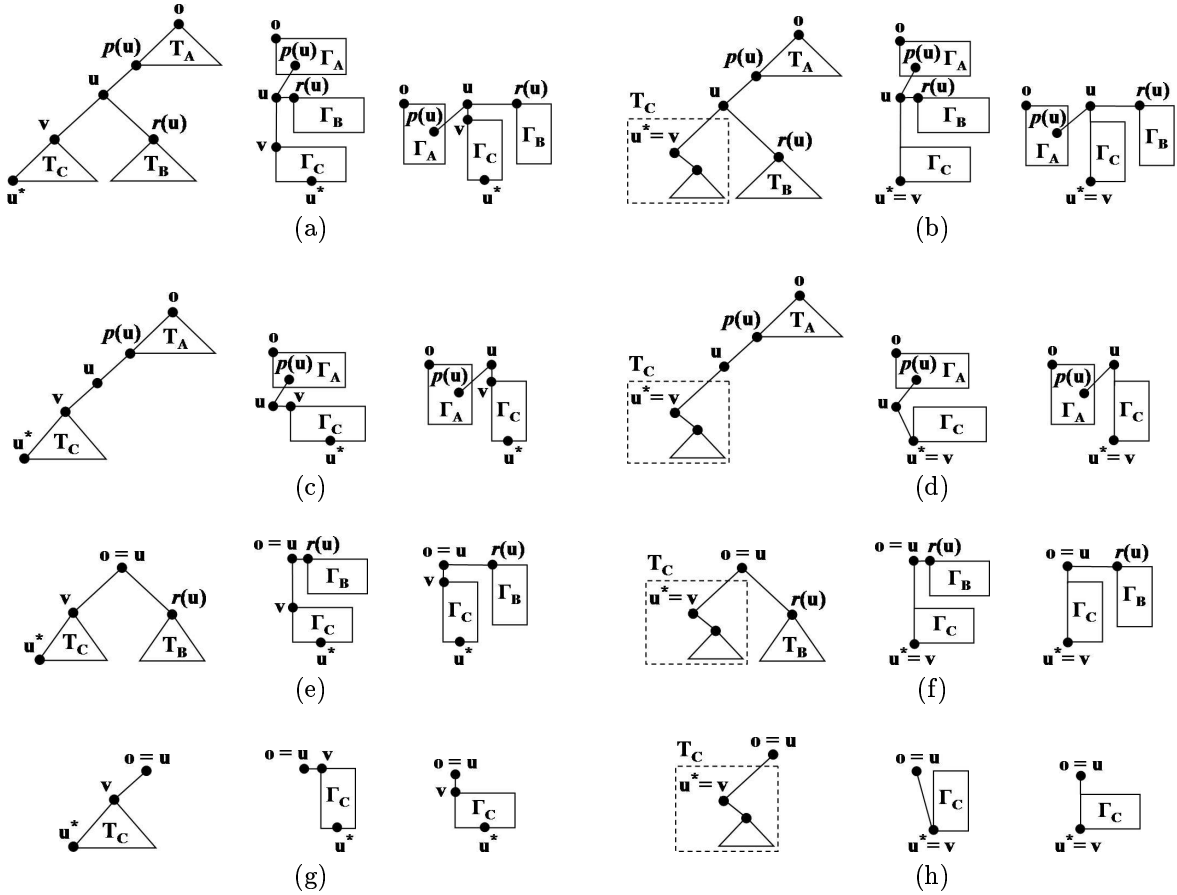
### 5.1 Split Tree

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  $T$ . Let  $o$  be the root of  $T$ . Let  $a$



**Figure 3:** Drawing  $T$  in all the seven subcases of Case 1 (where the separator edge  $(u, v)$  is not in the leftmost path of  $T$ ): (a)  $T_A \neq \emptyset$ ,  $T_C \neq \emptyset$ ,  $u^* \neq l(a)$ , (b)  $T_A \neq \emptyset$ ,  $T_C \neq \emptyset$ ,  $u^* = l(a)$ , (c)  $T_A \neq \emptyset$ ,  $T_C = \emptyset$ ,  $o \neq p(a)$ , (d)  $T_A \neq \emptyset$ ,  $T_C = \emptyset$ ,  $o = p(a)$ , (e)  $T_A = \emptyset$ ,  $T_C = \emptyset$ , (f)  $T_A = \emptyset$ ,  $T_C \neq \emptyset$ ,  $u^* \neq l(a)$ , and (g)  $T_A = \emptyset$ ,  $T_C \neq \emptyset$ ,  $u^* = l(a)$ . 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) and (b), for simplicity,  $p(a)$  is shown to be in the interior of  $\Gamma_A$ , but actually, either it is the same as  $o$ , or if  $A < 1$  ( $A \geq 1$ ), then it is placed at 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.



**Figure 4:** Drawing  $T$  in all the eight subcases of Case 2 (where the separator edge  $(u, v)$  is in the leftmost path of  $T$ ): (a)  $T_A \neq \emptyset$ ,  $T_B \neq \emptyset$ ,  $v \neq u^*$ , (b)  $T_A \neq \emptyset$ ,  $T_B \neq \emptyset$ ,  $v = u^*$ , (c)  $T_A \neq \emptyset$ ,  $T_B = \emptyset$ ,  $v \neq u^*$ , (d)  $T_A \neq \emptyset$ ,  $T_B = \emptyset$ ,  $v = u^*$ , (e)  $T_A = \emptyset$ ,  $T_B \neq \emptyset$ ,  $v \neq u^*$ , (f)  $T_A = \emptyset$ ,  $T_B \neq \emptyset$ ,  $v = u^*$ , (g)  $T_A = \emptyset$ ,  $T_B = \emptyset$ ,  $v \neq u^*$ , and (h)  $T_A = \emptyset$ ,  $T_B = \emptyset$ ,  $v = u^*$ . 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), (b), (c), and (d), for simplicity,  $p(u)$  is shown to be in the interior of  $\Gamma_A$ , but actually, either it is same as  $o$ , or if  $A < 1$  ( $A \geq 1$ ), then it is placed at 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.

be the last node common to the path  $o \rightsquigarrow v$ , and the leftmost path of  $T$ . We define partial trees  $T_A, T_B, T_C, T_\alpha, T_\beta, T_1$  and  $T_2$ , as follows (see Figure 3(a)):

- \* If  $o \neq a$ , then  $T_A$  is the maximal partial tree with root  $o$ , that contains  $p(a)$ , but does not contain  $a$ . If  $o = a$ , then  $T_A = \emptyset$ .
- \*  $T_B$  is the subtree rooted at  $r(a)$ .
- \* If  $u^* \neq a$ , then  $T_C$  is the subtree rooted at  $l(a)$ . If  $u^* = a$ , then  $T_C = \emptyset$ .
- \* If  $s(v)$  exists, i.e., if  $v$  has a sibling, then  $T_1$  is the subtree rooted at  $s(v)$ . If  $v$  does not have a sibling, then  $T_1 = \emptyset$ .
- \*  $T_2$  is the subtree rooted at  $v$ .
- \* If  $u \neq a$ , then  $T_\alpha$  is the subtree rooted at  $u$ . If  $u = a$ , then  $T_\alpha = T_2$ . Note that  $T_\alpha$  is a subtree of  $T_B$ .
- \* If  $u \neq a$  and  $u \neq r(a)$ , then  $T_\beta$  is the maximal partial tree with root  $r(a)$ , that contains  $p(u)$ , but does not contain  $u$ . If  $u = a$  or  $u = r(a)$ , then  $T_\beta = \emptyset$ . Again, note that  $T_\beta$  belongs to  $T_B$ .

We get seven subcases, where subcase (a) is the general case, and subcases (b–g) are special cases:

- (a)  $T_A \neq \emptyset, T_C \neq \emptyset, u^* \neq l(a)$  (see Figure 3(a)), (b)  $T_A \neq \emptyset, T_C \neq \emptyset, u^* = l(a)$  (see Figure 3(b)), (c)  $T_A \neq \emptyset, T_C = \emptyset, o \neq p(a)$  (see Figure 3(c)), (d)  $T_A \neq \emptyset, T_C = \emptyset, o = p(a)$  (see Figure 3(d)), (e)  $T_A = \emptyset, T_C = \emptyset$  (see Figure 3(e)), (f)  $T_A = \emptyset, T_C \neq \emptyset, u^* \neq l(a)$  (see Figure 3(f)), and (g)  $T_A = \emptyset, T_C \neq \emptyset, u^* = l(a)$  (see Figure 3(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,  $u^* = l(a)$  or not, and  $o = p(a)$  or not, we get a total of sixteen cases. From these sixteen cases, we obtain the above seven subcases, by grouping some of them together. For example, the cases  $T_A \neq \emptyset, T_C \neq \emptyset, u^* \neq l(a), o = p(a)$ , and  $T_A \neq \emptyset, T_C \neq \emptyset, u^* \neq l(a), o \neq p(a)$  are grouped together to give Case (a), i.e.,  $T_A \neq \emptyset, T_C \neq \emptyset, u^* \neq l(a)$ . So, Case (a) corresponds to 2 cases. Similarly, Cases (b), (c), (d), (f), and (g) correspond to 2 cases each, and Case (e) corresponds to 4 cases.

In each case, we remove nodes  $a$  and  $u$  (which could be the same node as  $a$ ), and their incident edges, to split  $T$  into at most five partial trees  $T_A, T_C, T_\beta, T_1$ , and  $T_2$ . We also designate  $p(a)$  as the link node of  $T_A$ ,  $p(u)$  as the link node of  $T_\beta$ , and  $u^*$  as the link node of  $T_C$ . We arbitrarily select a node of  $T_1$  that has at most one child (for example, we can select a leaf), and designate it as the link node of  $T_1$ . We arbitrarily select a node of  $T_2$  that has at most one child (for example, we can select a leaf), and designate it as the link node of  $T_2$ .

- *Case 2: The separator edge  $(u, v)$  is in the leftmost path of  $T$ .* Let  $o$  be the root of  $T$ . We can define partial trees  $T_A, T_B$ , and  $T_C$  as follows (see Figure 4(a)):

- \* If  $o \neq u$ , then  $T_A$  is the maximal partial tree with root  $o$ , that contains  $p(u)$ , but does not contain  $u$ . If  $o = u$ , then  $T_A = \emptyset$ .
- \* If  $r(u)$  exists, i.e.,  $u$  has a right child, then  $T_B$  is the subtree rooted at  $r(u)$ . If  $u$  does not have a right child, then  $T_B = \emptyset$ .
- \*  $T_C$  is the subtree rooted at  $v$ .

We get eight subcases, where subcase (a) is the general case, and subcases (b–h) are special cases:

- (a)  $T_A \neq \emptyset, T_B \neq \emptyset, v \neq u^*$  (see Figure 4(a)), (b)  $T_A \neq \emptyset, T_B \neq \emptyset, v = u^*$  (see Figure 4(b)), (c)  $T_A \neq \emptyset, T_B = \emptyset, v \neq u^*$  (see Figure 4(c)), (d)  $T_A \neq \emptyset, T_B = \emptyset, v = u^*$  (see Figure 4(d)), (e)  $T_A = \emptyset, T_B \neq \emptyset, v \neq u^*$  (see Figure 4(e)), (f)  $T_A = \emptyset, T_B \neq \emptyset, v = u^*$  (see Figure 4(f)), (g)  $T_A = \emptyset, T_B = \emptyset, v \neq u^*$  (see Figure 4(g)), and (h)  $T_A = \emptyset, T_B = \emptyset, v = u^*$  (see Figure 4(h)).

The reason we get these eight subcases is as follows:  $T_C$  has at least  $n/3$  nodes in it because of Theorem 1. Hence,  $T_C \neq \emptyset$ . Based on whether  $T_A = \emptyset$  or not,  $T_B = \emptyset$  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  $T_A, T_B$ , and  $T_C$ . We also designate  $p(u)$  as the link node of  $T_A$ , and  $u^*$  as the link node of  $T_C$ . We arbitrarily select a node of  $T_B$  that has at most one child (for example, we can select a leaf), and designate it as the link node of  $T_B$ .



*Remark:* In Case 2, from the definition of the separator edge  $(u, v)$  (see Theorem 1), it can be easily shown that  $T_A = \emptyset$  and  $T_B = \emptyset$  can happen simultaneously only if  $T$  has very few nodes in it, namely, at most 5 nodes. Hence, Case 2(g) and Case 2(h) can occur only if  $T$  has at most 5 nodes in it.

## 5.2 Assign Aspect Ratios

Let  $T_k$  be a partial tree of  $T$ , where for Case 1,  $T_k$  is either  $T_A, T_C, T_\beta, T_1$ , or  $T_2$ , and for Case 2,  $T_k$  is either  $T_A, T_B$ , or  $T_C$ . Let  $n_k$  be number of nodes in  $T_k$ .

**Definition:**  $T_k$  is a *large* partial tree of  $T$  if:

- $A \geq 1$  and  $n_k \geq (n/A)^{1/(1+\epsilon)}$ , or
- $A < 1$  and  $n_k \geq (An)^{1/(1+\epsilon)}$ ,

and is a *small* partial tree of  $T$  otherwise.

In Step *Assign Aspect Ratios*, we assign a desirable aspect ratio  $A_k$  to each non-empty  $T_k$  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  $T$ )  $A_k = n_k^\epsilon$ .

Intuitively, the above assignment strategy ensures that each partial tree gets a good desirable aspect ratio.

## 5.3 Draw Partial Trees

First, we change the values of  $A_A$  and  $A_\beta$  in some situations, as follows: (recall that  $A_A$  and  $A_\beta$  are the desirable aspect ratios for  $T_A$  and  $T_\beta$ , respectively, when they are non-empty trees)

- In Case 1(c), we change the value of  $A_A$  to  $1/A_A$ . Moreover, in Case 1(c), if  $A \geq 1$ , then we change the value of  $A_\beta$  also to  $1/A_\beta$ .
- In Cases 1(a) and 1(b), if  $A \geq 1$ , then we change the values of  $A_A$  and  $A_\beta$  to  $1/A_A$  and  $1/A_\beta$ , respectively.
- In Cases 1(d), 1(e), 1(f), and 1(g), if  $A \geq 1$ , then we change the values of  $A_\beta$  to  $1/A_\beta$ .
- In Cases 2(a), 2(b), 2(c), and 2(d), if  $A \geq 1$ , then we change the value of  $A_A$  to  $1/A_A$ .

(This is done so because later in Step *Compose Drawings*, when constructing  $\Gamma$ ,

- in Case 1(c), the drawing of  $T_A$  is rotated by  $90^\circ$ , and if  $A \geq 1$ , then the drawing of  $T_\beta$  is also rotated by  $90^\circ$ ,
- in Cases 1(a) and 1(b), if  $A \geq 1$ , then the drawings of  $T_A$  and  $T_\beta$  are rotated by  $90^\circ$ ,
- in Cases 1(d), 1(e), 1(f), and 1(g), if  $A \geq 1$ , then the drawing of  $T_\beta$  is rotated by  $90^\circ$ , and
- in Cases 2(a), 2(b), 2(c), and 2(d), if  $A \geq 1$ , then the drawing of  $T_A$  is rotated by  $90^\circ$ .

Drawing  $T_A$  and  $T_\beta$  with desirable aspect ratios  $1/A_A$  and  $1/A_\beta$ , respectively, compensates for the rotation, and ensures that the drawings of  $T_A$  and  $T_\beta$  that eventually get placed within  $\Gamma$  are those with desirable aspect ratios  $A_A$  and  $A_\beta$ , respectively.)

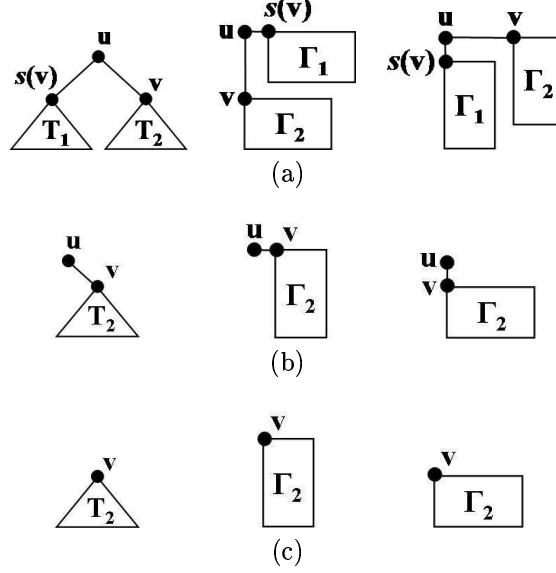
Next, we draw recursively each non-empty partial tree  $T_k$  with  $A_k$  as its desirable aspect ratio. The base case for the recursion happens when  $T_k$  contains exactly one node, in which case, the drawing of  $T_k$  is simply the one consisting of exactly one node.

## 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 drawing  $\Gamma_\alpha$  of the partial tree  $T_\alpha$  by composing  $\Gamma_1$  and  $\Gamma_2$  as shown in Figure 5, then construct a 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 3.

$\Gamma_\alpha$  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  $T_\alpha = T_2$ )



**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.

- If  $u \neq a$  and  $T_1 \neq \emptyset$  (see Figure 5(a)), then:
  - If  $A < 1$ , then 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  $s(v)$ .
  - If  $A \geq 1$ , then 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  $s(v)$  and in the same horizontal channel as  $v$ .

Draw edges  $(u, s(v))$  and  $(u, v)$ .

- If  $u \neq a$  and  $T_1 = \emptyset$  (see Figure 5(b)), then:
  - If  $A < 1$ , then place  $u$  one unit to the left of  $\Gamma_2$  in the same horizontal channel as  $v$ .
  - If  $A \geq 1$ , then place  $u$  one unit above  $\Gamma_2$  in the same vertical channel as  $v$ .

Draw edge  $(u, v)$ .

- If  $u = a$ , then  $\Gamma_\alpha$  is the same as  $\Gamma_2$  (see Figure 5(c)).

$\Gamma_B$  is constructed as follows (see Figure 6):

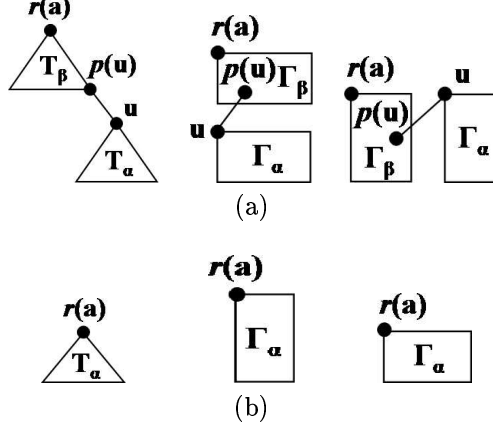
- If  $T_\beta \neq \emptyset$  (see Figure 6(a)) then:
  - if  $A < 1$ , then place  $\Gamma_\beta$  one unit above  $\Gamma_\alpha$  such that the left boundaries of  $\Gamma_\beta$  and  $\Gamma_\alpha$  are aligned.
  - If  $A \geq 1$ , then first rotate  $\Gamma_\beta$  by  $90^\circ$  and then 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  $(p(u), u)$ .

- If  $T_\beta = \emptyset$ , then  $\Gamma_B$  is same as  $\Gamma_\alpha$  (see Figure 6(b)).

$\Gamma$  is constructed from  $\Gamma_A$ ,  $\Gamma_B$ , and  $\Gamma_C$  as follows (see Figure 3):

- In Subcase (a),  $\Gamma$  is constructed as shown in Figure 3(a):
  - If  $A < 1$ , stack  $\Gamma_A$ ,  $\Gamma_B$ , and  $\Gamma_C$  one above the other, such that they are separated by unit distance from each other, and the left boundaries of  $\Gamma_A$  and  $\Gamma_C$  are aligned with each other and are placed one unit to the left of the left boundary of  $\Gamma_B$ . Place  $a$  in the same vertical channel as  $o$  and  $l(a)$ , and in the same horizontal channel as  $r(a)$ .



**Figure 6:** Drawing  $T_B$  when: (a)  $T_\beta \neq \emptyset$ , and (b)  $T_\beta = \emptyset$ . For each case, we first show the structure of  $T_B$  for that case, then its drawing when  $A < 1$ , and then its drawing when  $A \geq 1$ . In Case (a), for simplicity,  $p(u)$  is shown to be in the interior of  $\Gamma_\beta$ , but actually, it is either same as  $r(a)$ , 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.

- If  $A \geq 1$ , then first rotate  $\Gamma_A$  by  $90^\circ$ , and then flip it vertically. Then, place  $\Gamma_A$ ,  $\Gamma_C$ , and  $\Gamma_B$  from left-to-right in that order, separated by unit distances, such that the top boundaries of  $\Gamma_A$  and  $\Gamma_B$  are aligned with each other, and are one unit above the top boundary of  $\Gamma_C$ . Then, move  $\Gamma_C$  down until  $u^*$  becomes the lowest node of  $\Gamma$ . Place  $a$  in the same vertical channel as  $l(a)$  and in the same horizontal channel as  $o$  and  $r(a)$ .

Draw edges  $(p(a), a)$ ,  $(a, r(a))$ , and  $(a, l(a))$ .

- The drawing procedure for Subcase (b) is similar to the one in Subcase (a), except that we also flip  $\Gamma_C$  vertically (see Figure 3(b)).
- In Subcase (c),  $\Gamma$  is constructed as shown in Figure 3(c):
  - If  $A < 1$ , then first flip  $\Gamma_B$  vertically, and then flip it horizontally, so that its root  $r(a)$  gets placed at its lower-right corner. Then, first rotate  $\Gamma_A$  by  $90^\circ$ , and then flip it vertically. Next, place  $\Gamma_A$  above  $\Gamma_B$  at unit distance, such that their left boundaries are aligned. Next move node  $p(a)$  (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  $p(a)$  in this manner will not create any edge-crossings). Place  $u^*$  in the same horizontal channel as  $r(a)$  and one unit to the right of  $p(a)$ .
  - If  $A \geq 1$ , then first rotate  $\Gamma_A$  by  $90^\circ$ , and then flip it vertically. Flip  $\Gamma_B$  vertically. Then, place  $\Gamma_A$ ,  $u^*$ , and  $\Gamma_B$  left-to-right in that order separated by unit distances, such that the top boundaries of  $\Gamma_A$  and  $\Gamma_B$  are aligned, and  $u^*$  is placed in the same horizontal channel as the bottom boundary of the drawing among  $\Gamma_A$  and  $\Gamma_B$  with greater height.

Draw edges  $(p(a), a)$  and  $(a, r(a))$  (i.e., the edges  $(p(a), u^*)$  and  $(u^*, r(a))$ ) because in this case,  $u^* = a$ .

- In Subcase (d),  $\Gamma$  is constructed as shown in Figure 3(d):
  - If  $A < 1$ , then first flip  $\Gamma_B$  vertically, then place  $\Gamma_A$  one unit above  $\Gamma_B$ , such that the left boundary of  $\Gamma_A$  is one unit to the left of the left boundary of  $\Gamma_B$ . Place  $u^*$  in the same vertical channel as  $o$  and in the same horizontal channel as  $r(a)$ .
  - If  $A \geq 1$ , then first flip  $\Gamma_B$  vertically, then place  $\Gamma_A$  one unit to the left of  $\Gamma_B$ , 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  $o$  and in the same horizontal channel as  $r(a)$ .

Draw edges  $(o, u^*)$  and  $(u^*, r(a))$  (i.e., the edges  $(p(a), a)$  and  $(a, r(a))$ ) because in this case,  $o = p(a)$

and  $u^* = a$ ). Note that, since  $\Gamma_A$  is a feasible drawing of  $T_A$ , from Property 3 (see Section 4), drawing  $(o, u^*)$  will not create any edge-crossings.

- In Subcase (e), for both  $A < 1$  and  $A \geq 1$ , place node  $o$  one unit above and one unit left of  $\Gamma_B$  (see Figure 3(e)). Draw edge  $(a, r(a))$  (i.e., the edge  $(o, r(o))$  because in this case,  $a = o$ ).
- 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 3(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 3(g)).

In Case 2, we construct  $\Gamma$  by composing  $\Gamma_A$ ,  $\Gamma_B$ , and  $\Gamma_C$ , as follows (see Figure 4):

- The drawing procedures in Subcases (a) and (e) are similar to those in Subcases (a) and (f), respectively, of Case 1 (see Figures 4(a,e)).
- In Subcase (c),  $\Gamma$  is constructed as shown in Figure 4(c):
  - If  $A > 1$ , we place  $\Gamma_A$  one unit above  $\Gamma_C$ , 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  $o$  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  $\Gamma_A$  one unit to the left of  $\Gamma_C$ , 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  $o$ .

Draw edges  $(p(u), u)$ , and  $(u, v)$ .

- The drawing procedure (see Figure 4(g)) in Subcase (g) is similar to that in Case (b) of drawing  $T_\alpha$  (see Figure 5(b)).
- The drawing procedures in Subcases (b), (d), (f), and (h) are similar to those in Subcases (a), (c), (e), and (g), respectively (see Figures 4(b,d,f,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  $T$ .*

**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.

From Figure 5, it can be easily seen that in both the cases,  $A < 1$  and  $A \geq 1$ ,  $\Gamma_\alpha$  is a planar drawing, and the root of  $T_\alpha$  is placed at its top-left corner.

From Figure 6, it can be easily seen that in both the cases,  $A < 1$  and  $A \geq 1$ ,  $r(a)$  is placed at the top-left corner of  $\Gamma_B$ . Note that because  $\Gamma_\beta$  is a feasible drawing of  $T_\beta$  and  $p(u)$  is its link node,  $p(u)$  is either at the bottom of  $\Gamma_\beta$  (from Property 2, see Section 4), or at the top-left corner of  $\Gamma_\beta$  and no other edge or node of  $T_\beta$  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  $(p(u), u)$  will not cause any edge crossings. Also, in Figure 6(a), in the case  $A \geq 1$ , drawing edge  $(p(u), u)$  will not cause any edge crossings because after rotating  $\Gamma_\beta$  by  $90^\circ$  and flipping it vertically,  $p(u)$  will either be at the right boundary of  $\Gamma_\beta$  (because of Property 2), or at the top-left corner of  $\Gamma_\beta$  and no other edge or node of  $T_\beta$  will be placed on, or cross the horizontal channel occupied by it (because of Properties 1 and 3). It therefore follows that in both the cases,  $A < 1$  and  $A \geq 1$ ,  $\Gamma_B$  will be a planar drawing.

Finally, by considering each of the seven subcases shown in Figure 3 one-by-one, we can show that  $\Gamma$  is a feasible drawing of  $T$ :

- *Subcase (a):* See Figure 3(a).  $\Gamma_A$  is a feasible drawing of  $T_A$ , and  $p(a)$  is the link node of  $T_A$ . Hence,  $p(a)$  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  $(p(a), a)$  will not create any edge-

crossings. In the case  $A \geq 1$  also, drawing edge  $(p(a), a)$  will not create any edge-crossings because after rotating  $\Gamma_A$  by  $90^\circ$  and flipping it vertically,  $p(a)$  will either be at the right boundary of  $\Gamma_A$  (because of Property 2), or at the top-left corner of  $\Gamma_B$  and no other edge or node of  $T_A$  will be placed on, or cross the horizontal channel occupied by it (because of Properties 1 and 3).

Nodes  $r(a)$  and  $l(a)$  are placed at the top-left corner of  $\Gamma_B$  and  $\Gamma_C$ , respectively. Hence, drawing edges  $(a, r(a))$  and  $(a, l(a))$  will not create any edge-crossings in both the cases,  $A < 1$  and  $A \geq 1$ .

In both the cases,  $A < 1$  and  $A \geq 1$ ,  $o$  gets placed at the top-left corner of  $\Gamma$ . Hence,  $\Gamma$  satisfies Property 1.

Since  $u^* \neq o$ , Property 3 is satisfied by  $\Gamma$  vacuously.

We now show that Property 2 is satisfied by  $\Gamma$ . In both the cases,  $A < 1$  and  $A \geq 1$ ,  $u^*$  gets placed at the bottom of  $\Gamma$ .  $\Gamma_C$  is a feasible drawing,  $l(a)$  is the root of  $T_C$ , and  $u^* \neq l(a)$ . Hence, from Property 2, we can move  $u^*$  downwards in its vertical channel by any distance without causing any edge-crossings in  $\Gamma_C$ . Hence, in  $\Gamma$  also, we can move  $u^*$  downwards in its vertical channel by any distance without causing any edge-crossings in  $\Gamma$ . Thus, Property 2 is satisfied by  $\Gamma$ .

We therefore conclude that in both the cases,  $A < 1$  and  $A \geq 1$ ,  $\Gamma$  is a feasible drawing of  $T$ .

- *Subcase (b)*: See Figure 3(b). The proof is similar to the one for Subcase (a), except that in this case, because  $u^* = l(a)$ , we use the fact that  $\Gamma_C$  satisfies Property 3 to prove that  $\Gamma$  satisfies Property 2. To elaborate, since  $u^* = l(a)$ ,  $l(a)$  is the root of  $\Gamma_C$ , and  $\Gamma_C$  is a feasible drawing, from Property 3, we can move  $u^*$  upwards in its vertical channel by any distance without causing any edge-crossings in  $\Gamma_C$ . We flip  $\Gamma_C$  vertically before placing it in  $\Gamma$ . Hence, it follows that in  $\Gamma$ , we can move  $u^*$  downwards in its vertical channel by any distance without causing any edge-crossings in  $\Gamma$ .
- *Subcase (c)*: See Figure 3(c).  $\Gamma_A$  is a feasible drawing of  $T_A$ ,  $p(a)$  is the link node of  $T_A$ , and  $p(a) \neq o$ . Hence, from Property 2,  $p(a)$  is located at the bottom of  $\Gamma_A$ . Rotating  $\Gamma_A$  by  $90^\circ$  and flipping it vertically will move  $p(a)$  to the right boundary of  $\Gamma_A$ . Moving  $p(a)$  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 edges  $(p(a), u^*)$  and  $(u^*, r(a))$  will not create any edge-crossings, and  $\Gamma$  will be a feasible drawing of  $T$ .
- *Subcase (d)*: See Figure 3(d).  $\Gamma_A$  is a feasible drawing of  $T_A$ ,  $p(a)$  is the link node of  $T_A$ , and  $p(a) = o$ . Hence, from Properties 1 and 3,  $p(a)$  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  $(p(a), u^*)$  will not create any edge-crossings, and  $\Gamma$  will be a feasible drawing of  $T$ .
- *Subcase (e)*: See Figure 3(e). Because  $r(a)$  is placed at the top-left corner of  $\Gamma_B$ , drawing edge  $(a, r(a))$  will not cause any edge-crossings in both the cases,  $A < 1$  and  $A \geq 1$ . It can be easily seen that  $\Gamma$  is a feasible drawing of  $T$  in both the cases when  $A < 1$  and  $A \geq 1$ .
- *Subcase (f)*: See Figure 3(f). It is straightforward to see that  $\Gamma$  is a feasible drawing of  $T$  in both the cases,  $A < 1$  and  $A \geq 1$ .
- *Subcase (g)*: See Figure 3(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  $(o, u^*)$  will not create any edge-crossings. From Property 3, we can move  $u^*$  upwards in its vertical channel by any distance without causing any edge-crossings in  $\Gamma_C$ . We flip  $\Gamma_C$  vertically before placing it in  $\Gamma$ . Hence, in  $\Gamma$ , we can move  $u^*$  downwards in its vertical channel by any distance without causing any edge-crossings in  $\Gamma$ . It therefore follows that  $\Gamma$  is a feasible drawing of  $T$ .

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

**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  $O(n) \cdot O(\log n) = O(n \log n)$ .  $\square$

**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,  $W = \sqrt{AD}$ .  $H = W/A = \sqrt{AD}/A = \sqrt{D/A}$ .  $\square$

**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  $[n^{-\epsilon}, n^\epsilon]$ . 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  $D(n) \leq c_1 n - c_2 n^\beta$  for all  $n \geq n_0$ , where  $n_0, c_1, c_2, \beta$  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  $P_k = c_1 n - c_2 n^\beta / x_k^{1-\beta}$ . From Theorem 1, it follows that  $n_k \leq (2/3)n$ , and hence,  $x_k \leq 2/3$ . Hence,  $P_k \leq c_1 n - c_2 n^\beta / (2/3)^{1-\beta} = c_1 n - c_2 n^\beta (3/2)^{1-\beta}$ . Let  $P' = c_1 n - c_2 n^\beta (3/2)^{1-\beta}$ . 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  $D(n_k) \leq c_1 n_k - c_2 n_k^\beta$ . Since  $x_k = n_k/n$ ,  $D(n_k) \leq c_1 n_k - c_2 n_k^\beta = c_1 x_k n - c_2 (x_k n)^\beta = x_k (c_1 n - c_2 n^\beta / x_k^{1-\beta}) = x_k P_k \leq x_k P'$ .

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,  $A_k = 1/n_k^\epsilon$ . From Lemma 3, we have that  $W_k = \sqrt{A_k D(n_k)} \leq \sqrt{(1/n_k^\epsilon)(x_k P')} = \sqrt{(1/n_k^\epsilon)(n_k/n)P'} = \sqrt{n_k^{1-\epsilon} P' / n}$ . Since  $n_k < (n/A)^{1/(1+\epsilon)}$ , it follows that  $W_k < \sqrt{(n/A)^{(1-\epsilon)/(1+\epsilon)} P' / n} = \sqrt{(1/A^{(1-\epsilon)/(1+\epsilon)}) P' / n^{2\epsilon/(1+\epsilon)}} \leq \sqrt{P' / n^{2\epsilon/(1+\epsilon)}}$ , since  $A \geq 1$ .  
From Lemma 3,  $H_k = \sqrt{D(n_k)/A_k} \leq \sqrt{x_k P' / (1/n_k^\epsilon)} = \sqrt{(n_k/n) P' n_k^\epsilon} = \sqrt{n_k^{1+\epsilon} P' / n}$ . Since  $n_k < (n/A)^{1/(1+\epsilon)}$ ,  $H_k < \sqrt{(n/A)^{(1+\epsilon)/(1+\epsilon)} P' / n} = \sqrt{(n/A) P' / n} = \sqrt{P' / A}$ .
- $T_k$  is a large partial tree of  $T$ : Then, as explained in Section 5.2,  $A_k = x_k A$ . From Lemma 3,  $W_k = \sqrt{A_k D(n_k)} \leq \sqrt{x_k A x_k P'} = x_k \sqrt{A P'}$ .  
From Lemma 3,  $H_k = \sqrt{D(n_k)/A_k} \leq \sqrt{x_k P' / (x_k A)} = \sqrt{P' / A}$ .

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  $T$ . For example, in Case 1(e), if  $u \neq a$  and  $T_1 = \emptyset$ , then we use one additional horizontal channel and one additional vertical channel for placing  $a$  (see Figure 3(e)), and one additional horizontal channel for placing  $u$  (see Figure 5(b)).

Hence,  $\Gamma$  can fit inside a rectangle  $R'$  with width  $W'$  and height  $H'$ , respectively, where,

$$H' \leq \max_{T_k \text{ is a partial tree of } T} \{H_k\} + 2 \leq \sqrt{P'/A} + 2,$$

and

$$\begin{aligned}
W' &\leq \sum_{T_k \text{ is a large partial tree}} W_k + \sum_{T_k \text{ is a small partial tree}} W_k + 1 \\
&\leq \sum_{T_k \text{ is a large partial tree}} x_k \sqrt{AP'} + \sum_{T_k \text{ is a small partial tree}} \sqrt{P'/n^{2\epsilon/(1+\epsilon)}} + 1 \\
&\leq \sqrt{AP'} + 5\sqrt{P'/n^{2\epsilon/(1+\epsilon)}} + 1
\end{aligned}$$

(because  $\sum_{T_k \text{ is a large partial tree}} x_k \leq 1$ , and  $T$  has at most 5 partial trees).

$R'$  might 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

$$W = \sqrt{AP'} + 5\sqrt{P'/n^{2\epsilon/(1+\epsilon)}} + 1 + 2A,$$

and

$$H = \sqrt{P'/A} + 2 + (5/A)\sqrt{P'/n^{2\epsilon/(1+\epsilon)}} + 1/A$$

Hence,  $D(n) = WH = (\sqrt{AP'} + 5\sqrt{P'/n^{2\epsilon/(1+\epsilon)}} + 1 + 2A)(\sqrt{P'/A} + 2 + (5/A)\sqrt{P'/n^{2\epsilon/(1+\epsilon)}} + 1/A) \leq P' + c_3P'/\sqrt{An^{2\epsilon/(1+\epsilon)}} + c_4\sqrt{AP'} + c_5P'/(An^{2\epsilon/(1+\epsilon)}) + c_6\sqrt{P'/n^{2\epsilon/(1+\epsilon)}} + c_7A + c_8 + c_9/A + c_{10}\sqrt{P'/A} + c_{11}\sqrt{P'/n^{2\epsilon/(1+\epsilon)}}/A$ , where  $c_3, c_4, \dots, c_{11}$  are some constants.

Since,  $1 \leq A \leq n^\epsilon$ , we have that

$$\begin{aligned}
D(n) &\leq P' + c_3P'/\sqrt{n^{2\epsilon/(1+\epsilon)}} + c_4\sqrt{n^\epsilon P'} + c_5P'/n^{2\epsilon/(1+\epsilon)} + c_6\sqrt{P'/n^{2\epsilon/(1+\epsilon)}} \\
&\quad + c_7n^\epsilon + c_8 + c_9 + c_{10}\sqrt{P'} + c_{11}\sqrt{P'/n^{2\epsilon/(1+\epsilon)}}
\end{aligned}$$

Since  $P' < c_1n$ ,

$$\begin{aligned}
D(n) &< P' + c_3c_1n/\sqrt{n^{2\epsilon/(1+\epsilon)}} + c_4\sqrt{n^\epsilon c_1n} + c_5c_1n/n^{2\epsilon/(1+\epsilon)} \\
&\quad + c_6\sqrt{c_1n/n^{2\epsilon/(1+\epsilon)}} + c_7n^\epsilon + c_8 + c_9 + c_{10}\sqrt{c_1n}^{1/2} \\
&\quad + c_{11}\sqrt{c_1n/n^{2\epsilon/(1+\epsilon)}} \\
&\leq P' + c_3c_1n^{1/(1+\epsilon)} + c_4\sqrt{c_1}n^{(1+\epsilon)/2} + c_5c_1n^{(1-\epsilon)/(1+\epsilon)} \\
&\quad + c_6\sqrt{c_1}n^{(1-\epsilon)/(2(1+\epsilon))} + c_7n^\epsilon + c_8 + c_9 + c_{10}\sqrt{c_1}n^{1/2} \\
&\quad + c_{11}\sqrt{c_1}n^{(1-\epsilon)/(2(1+\epsilon))} \\
&\leq P' + c_{12}n^{1/(1+\epsilon)} + c_{13}n^{(1+\epsilon)/2}
\end{aligned}$$

where  $c_{12}$  and  $c_{13}$  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  $1/2 < (1+\epsilon)/2$ ).

$P' = c_1n - c_2n^\beta(3/2)^{1-\beta} = c_1n - c_2n^\beta(1 + c_{14})$ , where  $c_{14}$  is a constant such that  $1 + c_{14} = (3/2)^{1-\beta}$ .

Hence,  $D(n) \leq c_1n - c_2n^\beta(1 + c_{14}) + c_{12}n^{1/(1+\epsilon)} + c_{13}n^{(1+\epsilon)/2} = c_1n - c_2n^\beta - (c_2c_{14}n^\beta - c_{12}n^{1/(1+\epsilon)} - c_{13}n^{(1+\epsilon)/2})$ . Thus, for a large enough constant  $n_0$ , and large enough  $\beta$ , where  $1 > \beta > \max\{1/(1+\epsilon), (1+\epsilon)/2\}$ , for all  $n \geq n_0$ ,  $c_2c_{14}n^\beta - c_{12}n^{1/(1+\epsilon)} - c_{13}n^{(1+\epsilon)/2} \geq 0$ , and hence  $D(n) \leq c_1n - c_2n^\beta$ .

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  $D(n)$ :

$$\begin{aligned}
W_k &\leq \sqrt{AP'} \\
H_k &\leq \sqrt{P'/n^{2\epsilon/(1+\epsilon)}} \text{ if } T_k \text{ is a small partial tree}
\end{aligned}$$

$$\begin{aligned}
&\leq x_k \sqrt{P'/A} \text{ if } T_k \text{ is a large partial tree} \\
W' &\leq \sqrt{AP'} + 2 \\
H' &\leq \sqrt{P'/A} + 5\sqrt{P'/n^{2\epsilon/(1+\epsilon)}} + 1 \\
W &\leq \sqrt{AP'} + 2 + 5A\sqrt{P'/n^{2\epsilon/(1+\epsilon)}} + A \\
H &\leq \sqrt{P'/A} + 5\sqrt{P'/n^{2\epsilon/(1+\epsilon)}} + 1 + 2/A \\
D(n) &\leq P' + c_{12}n^{1/(1+\epsilon)} + c_{13}n^{(1+\epsilon)/2}
\end{aligned}$$

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  $A > 1$  and  $A < 1$  are symmetrical to each other.)  $\square$

**Theorem 2 (Main Theorem)** *Let  $T$  be a binary tree with  $n$  nodes. Given two numbers  $A$  and  $\epsilon$ , where  $\epsilon$  is a constant, such that  $0 < \epsilon < 1$ , and  $n^{-\epsilon} \leq A \leq n^\epsilon$ , we can construct in  $O(n \log n)$  time, a planar straight-line grid drawing of  $T$  with  $O(n)$  area and aspect ratio  $A$ .*

**Proof:** Arbitrarily select a node of  $T$  that has at most one child (for example, we can select any leaf), and designate it the link node of  $T$ . Construct a drawing  $\Gamma$  of  $T$  by invoking Algorithm *DrawTree* with  $T$ ,  $A$ , and  $\epsilon$  as input. From Lemmas 1, 2, and 4,  $\Gamma$  will be a planar straight-line grid drawing contained entirely within a rectangle with  $O(n)$  area and aspect ratio  $A$ .  $\square$

**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  $A = 1$ , and  $\epsilon$  any constant, such that  $0 < \epsilon < 1$ .  $\square$

## 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  $T_0, T_1, \dots, T_n$  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  $p = u_0 u_1 \dots u_m$  of  $T_{i-1}$ , where  $u_0$  is the root of  $T_{i-1}$ , and  $u_m$  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  $u_0 u_1 \dots u_{i-1}$ , 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  $A$  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.

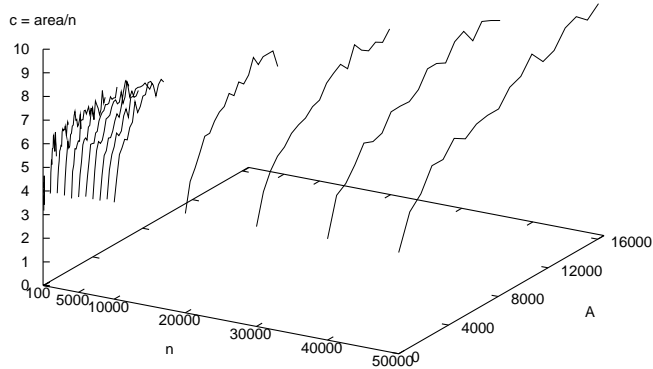
We also discovered that  $c$  increases with  $A$  for a given  $n$  and  $\epsilon$ . However, the rate of increase is very small. Consequently, for a given  $n$  and  $\epsilon$ , the range for  $c$  over all the values of  $A$  is small (see Figure 7(b,d,f,h,j), and Figure 8(b,d,f,h,j)). For example, 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  $\epsilon$ .

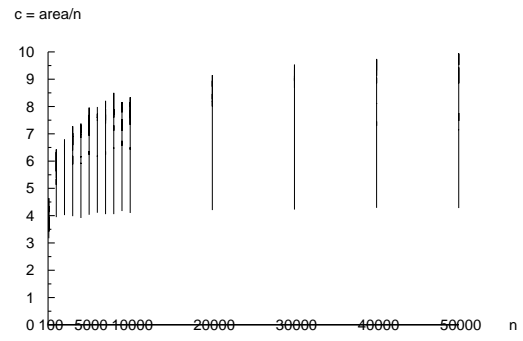
Finally, we would like to comment that the aspect ratio of the drawing constructed is, in general, different from the input aspect ratio  $A$ . We computed the ratio  $r$  of the aspect ratio of the drawing constructed by the algorithm and input aspect ratio  $A$ . We discovered that  $r$  is close to 1 for  $A = 1$ , generally decreases as we increase  $A$ , and can get as low as 0.1 for  $A = n^\epsilon$ . 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, 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

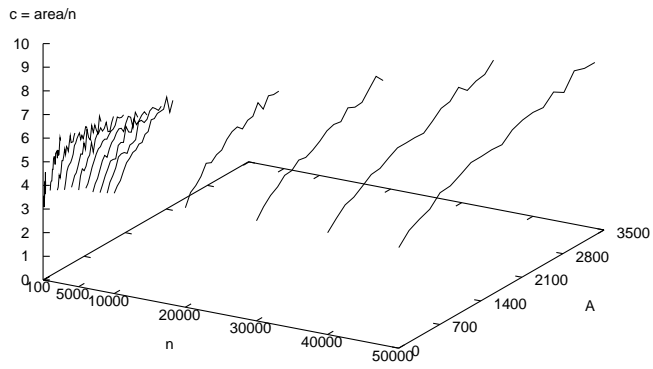
- [1] T. Chan, M. Goodrich, S. Rao Kosaraju, and R. Tamassia. Optimizing area and aspect ratio in straight-line orthogonal tree drawings. *Computational Geometry: Theory and Applications*, 23:153–162, 2002.
- [2] P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Comput. Geom. Theory Appl.*, 2:187–200, 1992.
- [3] P. Crescenzi, P. Penna, and A. Piperno. Linear-area upward drawings of AVL trees. *Comput. Geom. Theory Appl.*, 9:25–42, 1998. (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia).
- [4] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.
- [5] A. Garg, M. T. Goodrich, and R. Tamassia. Planar upward tree drawings with optimal area. *Internat. J. Comput. Geom. Appl.*, 6:333–356, 1996.
- [6] 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 *Lecture Notes in Computer Science*, pages 320–331. Springer-Verlag, 2002.
- [7] C. E. Leiserson. Area-efficient graph layouts (for VLSI). In *Proc. 21st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 270–281, 1980.
- [8] C.-S. Shin, S.K. Kim, S.-H. Kim, and K.-Y. Chwa. Area-efficient algorithms for straight-line tree drawings. *Comput. Geom. Theory Appl.*, 15:175–202, 2000.
- [9] L. Trevisan. A note on minimum-area upward drawing of complete and Fibonacci trees. *Inform. Process. Lett.*, 57(5):231–236, 1996.
- [10] L. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Comput.*, C-30(2):135–140, 1981.



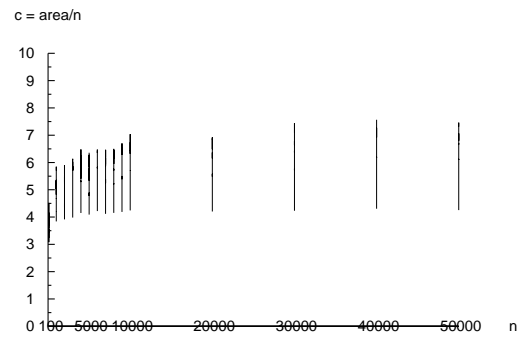
(a)  $\epsilon = 0.9$



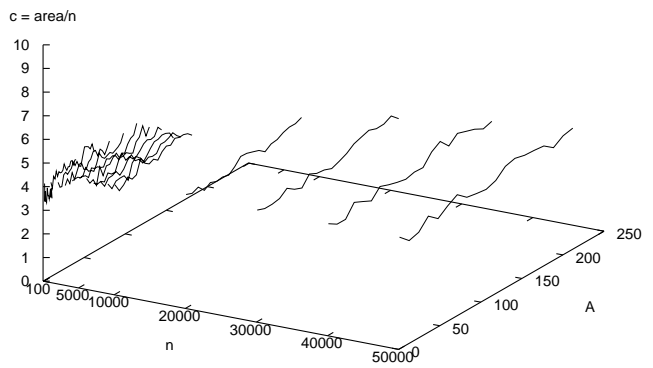
(b)  $\epsilon = 0.9$



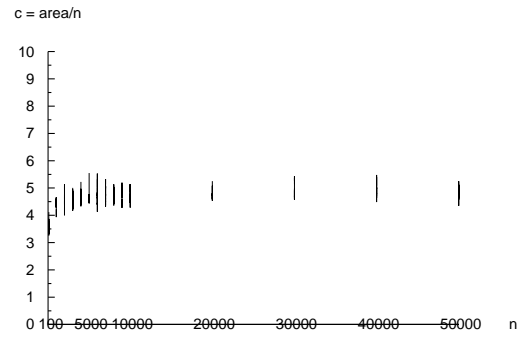
(c)  $\epsilon = 0.75$



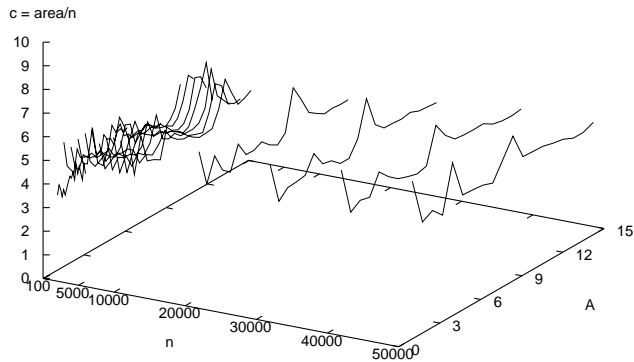
(d)  $\epsilon = 0.75$



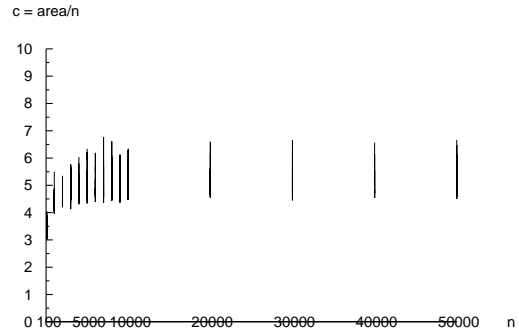
(e)  $\epsilon = 0.5$



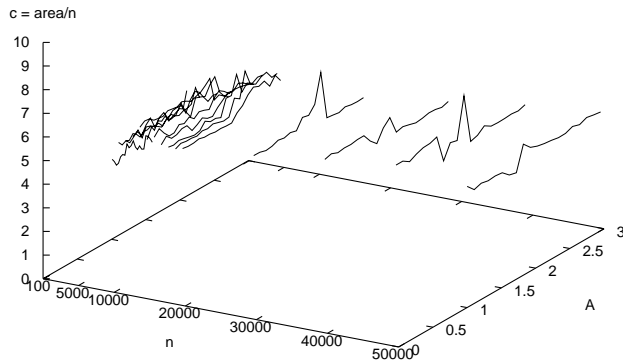
(f)  $\epsilon = 0.5$



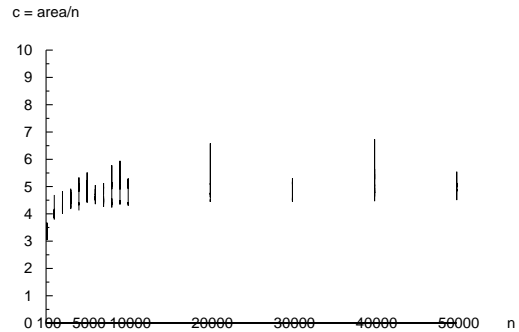
(g)  $\epsilon = 0.25$



(h)  $\epsilon = 0.25$

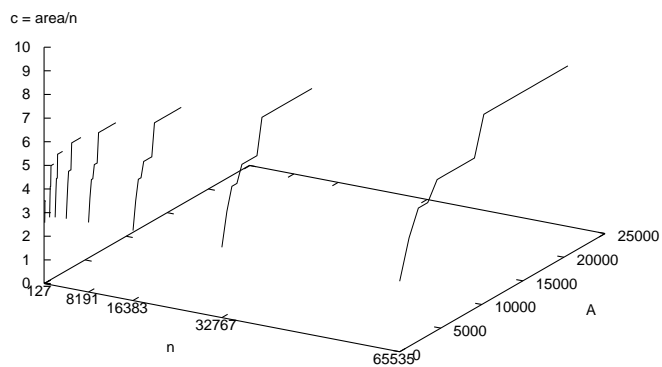


(i)  $\epsilon = 0.1$

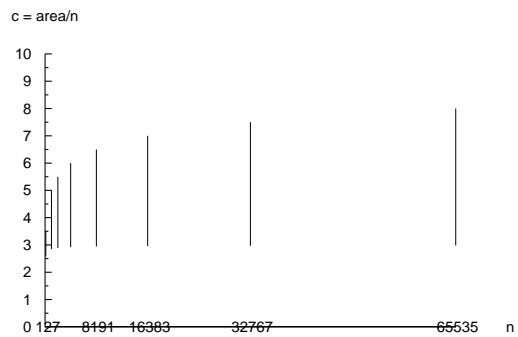


(j)  $\epsilon = 0.1$

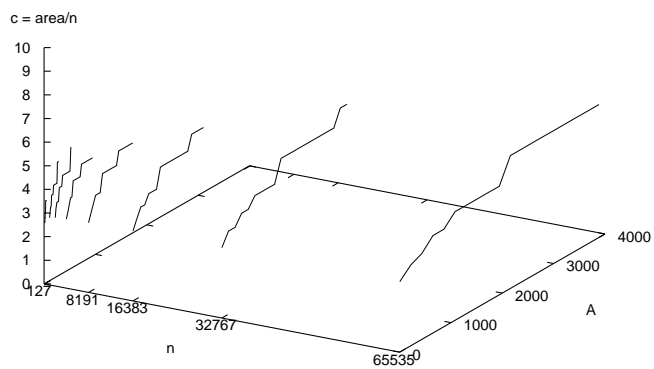
**Figure 7:** Performance of the algorithm, as given by the value of  $c$ , for drawing a randomly-generated binary tree  $T$  with different values of  $A$  and  $\epsilon$ , where  $c = \text{area of drawing} / \text{number of nodes } n \text{ in } T$ : (a)  $\epsilon = 0.9$ , (c)  $\epsilon = 0.75$ , (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  $n$ .



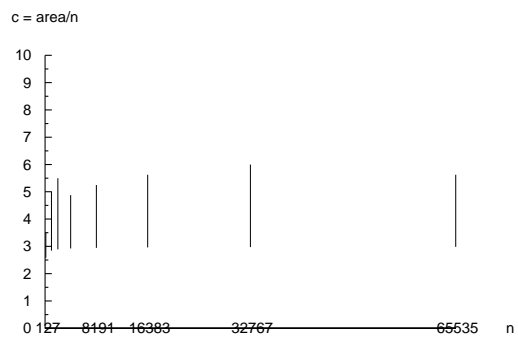
(a)  $\epsilon = 0.9$



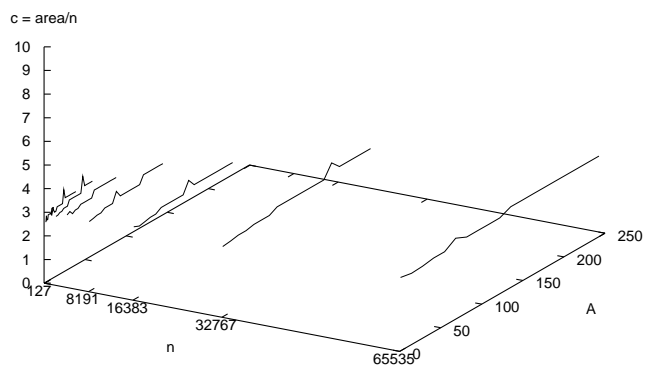
(b)  $\epsilon = 0.9$



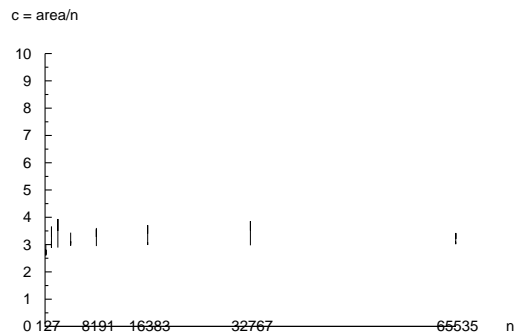
(c)  $\epsilon = 0.75$



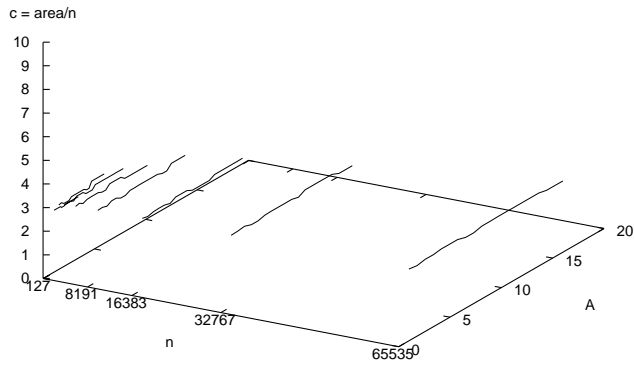
(d)  $\epsilon = 0.75$



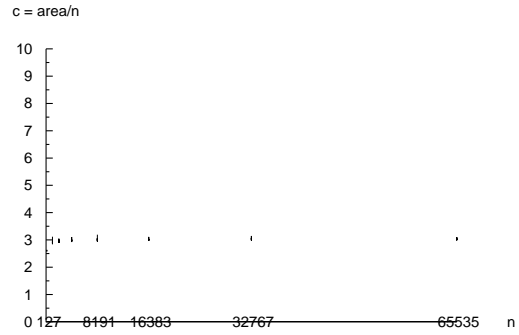
(e)  $\epsilon = 0.5$



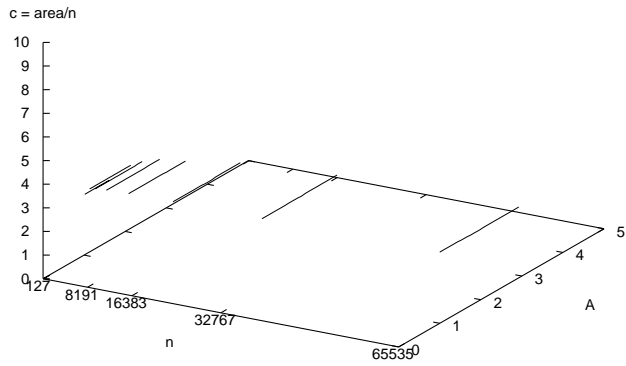
(f)  $\epsilon = 0.5$



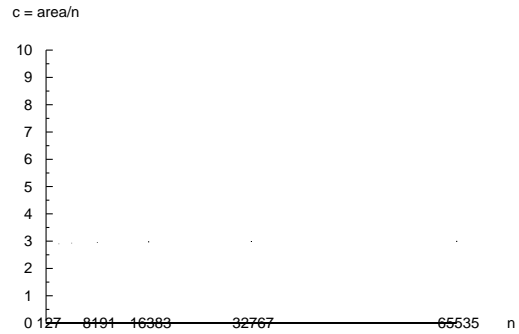
(g)  $\epsilon = 0.25$



(h)  $\epsilon = 0.25$



(i)  $\epsilon = 0.1$



(j)  $\epsilon = 0.1$

**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  $c = \text{area of drawing} / \text{number of nodes } n \text{ in } T$ : (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  $n$ .