Straight-Line Drawings of Binary Trees with Linear Area and Arbitrary Aspect Ratio [⋆] (Extended Abstract)

Ashim Garg and Adrian Rusu

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

Abstract. Trees are usually drawn planar, i.e. without any crossings. In this paper, we investigate the area requirement of (non-upward) planar straight-line grid drawings of binary trees. Let T be a binary tree with n nodes. We show that T admits a planar straight-line grid drawing with area $O(n)$ and with any pre-specified aspect ratio in the range $[1, n^\alpha]$, where α is a constant such that $0 \leq \alpha < 1$. We also show that such a drawing can be constructed in $O(n \log n)$ time.

1 Introduction

A *drawing* Γ 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. Γ is a *straight-line* drawing if each edge is drawn as a single straight-line segment. Γ is a *polyline* drawing if each edge is drawn as a connected sequence of one or more line-segments, where the meeting point of consecutive line-segments is called a *bend*. Γ is a *grid* drawing if all the nodes and edge-bends have integer coordinates. Γ is a *planar* drawing if edges do not intersect each other in the drawing. Γ is an *upward* drawing if a parent is always assigned either the same or higher y-coordinate than its children. 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 longer and shorter sides. R is the *enclosing rectangle* of Γ, if it is the smallest rectangle that covers the entire drawing. The *width*, *height*, *area*, and *aspect ratio* of Γ 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 descendents of v. Γ 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.

⋆ Research supported by NSF CAREER Award IIS-9985136 and NSF CISE Research Infrastructure Award No. 0101244.

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 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 is easier for the user to detect the subtrees in 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.

We, therefore, investigate the problem of constructing (non-upward) 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 question, therefore, has been 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 $O(n)$?

In this paper, we answer this question in affirmative, by giving an algorithm that constructs a planar straight-line grid drawing of a binary tree with n nodes with $O(n)$ area in $O(n \log n)$ time. Moreover, the drawing can be parameterized for its aspect ratio, i.e., for any constant α, where $0 \leq \alpha < 1$, the algorithm can construct a drawing with any user-specified aspect ratio in the range $[1, n^\alpha]$. The drawing also exhibits the subtree separation property.

3 Previous Results

Previously, the best-known bound on area of planar straight-line grid drawings of binary trees was $O(n \log \log n)$, which was shown in [1] and [7].

We now summarize some other known results on planar grid drawings of binary trees (for more results, see [4]). Let T be a binary tree. [5] presents an algorithm for constructing an upward polyline drawing of T in $O(n)$ area. [6] and [9] present algorithms for constructing an orthogonal polyline drawing of T in $O(n)$ area. [2] gives an algorithm for constructing upward straight-line drawing of T in $O(n \log n)$ area. If T is a Fibonacci tree, (AVL tree, balanced tree, respectively), then [2,8] ([3],[2], respectively) give algorithms for constructing an upward straight-line drawing of T in $O(n)$ area.

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

Let R be a rectangle, such that Γ is entirely contained within R. R has a *good* aspect ratio, if its aspect ratio is in the range $[1, n^\alpha]$, where α is a constant, such that $0 \le \alpha < 1$.

Let r be the root of T. Let u^* be the link node of T. Γ is a *feasible* drawing of T, if it has the following three properties:

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

Theorem 1 ([9]). *Let n_0 and α be two constants, where $n_0 \ge 1$ and $0 < \alpha < 1/2$. Let $m_1, m_2, \dots, m_{n_0}$ be a set of positive numbers. Let $G(n)$, and $g(n)$ be two functions such that*

– *$\forall n > n_0$, and $\forall x$, where $0 < x \le 1-\alpha$, $G(xn) \le x(G(n) - g(n)\sqrt{G(n)})$, and*
– *$\forall n \le n_0$, $G(n) \ge m_n$.*

Then, if $g(n) \le cn^\beta$, where $c > 0$ and β are constants, and $0 \le \beta < 1/2$, then $G(n) = O(n)$. More specifically, $G(n) = \gamma n - \delta\sqrt{n}g(n)$ is a solution, where γ and δ are constants that satisfy the following conditions:

– *$\forall n \le n_0$, $\gamma n - \delta\sqrt{n}cn^\beta \ge m_n$, and*
– *$\delta(\mu^{1/2-\beta} - 1) > \sqrt{\gamma}$, where $\mu = 1/(1-\alpha)$.*

Theorem 2 (Separator Theorem [9]). *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 $1/3 \le x \le 2/3$, $n_1 \le xn$, and $n_2 \le (1-x)n$. More over, e can be found in $O(n)$ time.*

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

- *rotate operation*: rotate Γ counterclockwise by δ degrees around the z-axis passing through r. After a rotation by δ degrees of Γ, node v will be located at grid point $(i\cos\delta - j\sin\delta, i\sin\delta + j\cos\delta)$.
- *flip operation*: flip Γ vertically or horizontally. After a horizontal flip of Γ, node v will be located at grid point $(-i, j)$. After a vertical flip of Γ, node v will be located at grid point $(i, -j)$.

5 Algorithm u^*-*HV-Draw*

We first describe Algorithm u^*-*HV-Draw*, which is used by our main drawing algorithm (described in Section 6).

Algorithm u^*-*HV-Draw* is based on the well-known HV-Drawing algorithm of [2]. Lemma 1 summarizes the main result on HV-Drawing algorithm:

Lemma 1 ([2]). *Let T be an n-node binary tree with root r. Let R be a rectangle with width (height) n and height (width) $\log_2 n + 2$. We can construct a drawing of T within R in $O(n)$ time, such that r is placed at the top-left corner of R.*

Unfortunately, given a tree T with a link node u^*, HV-Drawing algorithm may not construct a feasible drawing of T because it may not place u^* at the bottom of the drawing. Algorithm u^*-*HV-Draw* will, however, construct a feasible drawing.

Algorithm u^*-*HV-Draw* can be described as follows: Let W and H be the width and height of R, respectively.

- Order the children of each node such that u^* becomes the leftmost node of T.
- Let $r = r_1, r_2, \ldots, r_{k-1}, r_k = u^*$, be the nodes in the leftmost path of T (see Figure 1(a)). Let $c = c_1, c_2, \ldots, c_k$ be the right children of $r_1, r_2, \ldots, r_k$, respectively. Let $T_1, T_2, \ldots, T_k$ be the subtrees of T rooted at $c_1, c_2, \ldots, c_k$, respectively. For each i, where $1 \le i \le k$, let n_i be the number of nodes in T_i.
- If $W \le H$ ($H < W$), for each i, where $1 \le i \le k$, construct a drawing Γ_i of T_i, using HV-Drawing algorithm, within a rectangle R_i with height (width) n_i and width (height) $\log_2 n_i + 2$ (see Lemma 1).
- Based on the value of k, tree T is drawn as follows:
  - $k \ge 2$: If $W \le H$, then as shown in Figure 1(b), stack the drawings of $\Gamma_1, \Gamma_2, \ldots, \Gamma_k$ one above the other such that their left boundaries are aligned. Place each r_i, where $1 \le i \le k-1$, at the same height as c_i and at unit horizontal distance from it towards its left. Place u^* to the left of c_k at unit horizontal distance and at unit vertical distance below the bottom boundary of Γ_k. If $H < W$, then first flip Γ_k vertically, and then, as shown in Figure 1(c), place $\Gamma_1, \Gamma_2, \ldots, \Gamma_k$ in a left-to-right order with two units of horizontal separation between them, such that their top boundaries are aligned. Let Γ_h be the drawing among $\Gamma_1, \Gamma_2, \ldots, \Gamma_k$ with maximum height. Place u^* one unit below the bottom boundary of Γ_h, and one unit to the left of c_{k-1}. Move Γ_k down until its bottom boundary is either below, or aligned with the bottom boundary of Γ_{k-1}. Place each r_i, where $1 \le i \le k-1$, at one unit above and one unit to the left of c_i, and draw edges (r_i, c_i), and (r_i, r_{i+1}). Draw the edge (u^*, c_k).

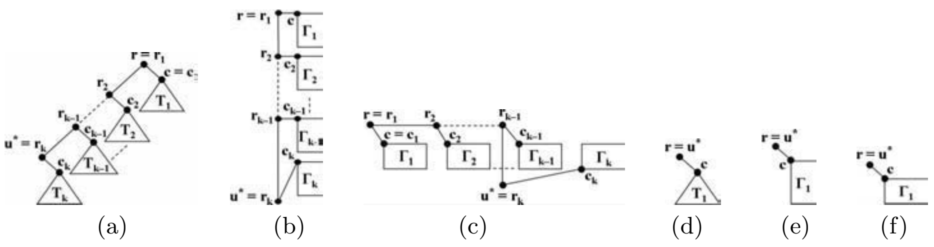

Fig. 1. (a) General form of a binary tree T with link node u^* and root r, where u^* is also the leftmost node of T. This figure uses the notation of Section 5. (b,c) Drawing T when $k \geq 2$: if (b) $W \leq H$, and if (c) $H < W$. (d) Special case when $k = 1$. (e,f) Drawing T when $k = 1$: if (e) $W \leq H$, and if (f) $H < W$. For simplicity, we have shown $\Gamma_1, \Gamma_2, \ldots, \Gamma_k$ as identically sized boxes, but in actuality, they may have different sizes.

- $k = 1$: For both $W \leq H$ and $H < W$, place r one unit above and to the left of the top-left corner of Γ_1 (see Figure 1 (d,e,f)). Draw edge (r, c).

Lemma 2. *Let T be an n-node binary tree with a link vertex u^*. Algorithm u^*-*HV-Draw *can be used to construct in $O(n)$ time, a feasible drawing of T within a rectangle with width n and height* $\log_2 n + 4$*, as well as within a rectangle with width* $\log_2 n + 4$ *and height n.*

6 Our Overall Tree Drawing Algorithm

Let T be a binary tree with a link node u^*. Let n be the number of nodes in T. Let A and ϵ be two numbers such that $1/3 < \epsilon < 1$, and A is in the range $[1, n^\epsilon]$. Let R be a large enough rectangle with aspect ratio A, width W and height H (we will determine later on, how large R should be). Notice that because $1 \leq A < n^\epsilon$, R has a good aspect ratio.

Our tree drawing algorithm, called *DrawTree*, uses a simple divide-and-conquer strategy to recursively construct a feasible drawing Γ of T within R, 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 2 and 3, and described later in Section 6.1.
- *Assign Rectangles*: Correspondingly, split R into at most five smaller rectangles, and assigns each smaller rectangle to a partial tree. The splitting is done by cutting the longer side (height or width) of R.
- *Draw Partial Trees*: Recursively construct a feasible drawing of each partial tree within its assigned rectangle.

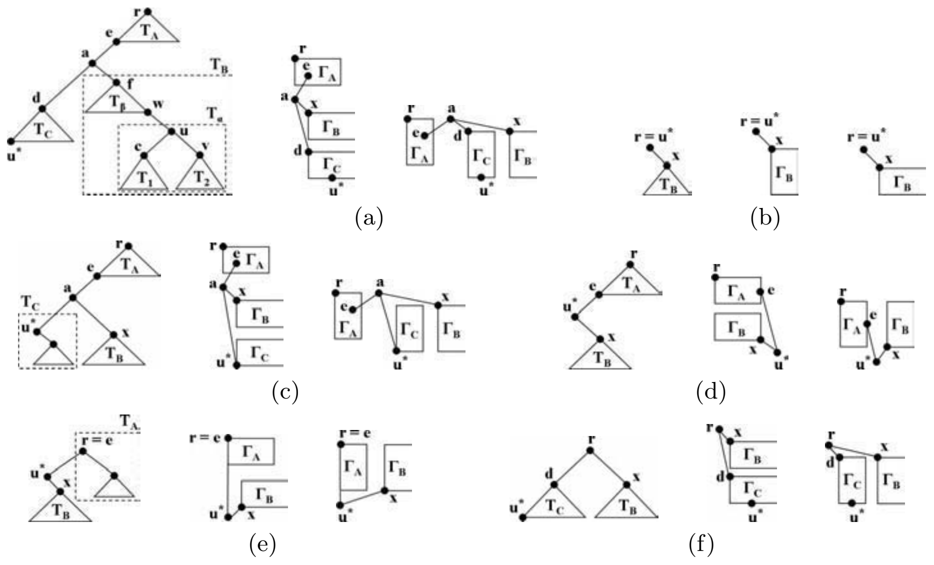

Fig. 2. Drawing T in all the subcases of Case 1 (when the separator (u, v) is not in the leftmost path of T): (a) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $d \neq u^*$, (b) $T_A = \emptyset$, $T_C = \emptyset$, (c) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $d = u^*$, (d) $T_A \neq \emptyset$, $T_C = \emptyset$, $r \neq e$, (e) $T_A \neq \emptyset$, $T_C = \emptyset$, $r = e$, (f) $T_A = \emptyset$, $T_C \neq \emptyset$, $d \neq u^*$. For each subcase, we first show the structure of T for that subcase, then its drawing when $W \leq H$, and then its drawing when $H < W$. In Subcases (a) and (c), for simplicity, e is shown to be in the interior of Γ_A, but actually, either it is the same as r, or if $W \leq H$ $(H < W)$, then it is placed on the bottom (right) boundary of Γ_A. For simplicity, we have shown Γ_A, Γ_B, and Γ_C as identically sized boxes, but in actuality, they may have different sizes.

- *Compose Drawings*: Within R, 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 Γ 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.

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

6.1 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 2, 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 six subcases: (a) In the general case, T has the form as shown in Figure 2(a). In this figure: 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, T_α is the

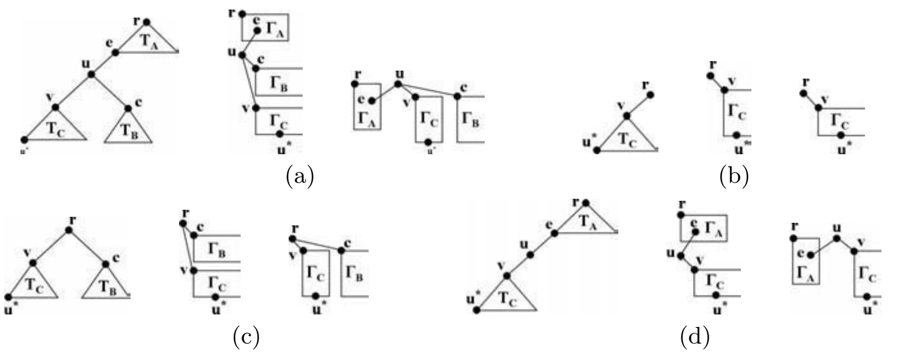

Fig. 3. Drawing T in all the subcases of Case 2 (when the separator (u, v) is in the leftmost path of T): (a) $T_A \neq \emptyset$, $T_B \neq \emptyset$. (b) $T_A = \emptyset$, $T_B = \emptyset$. (c) $T_A = \emptyset$, $T_B \neq \emptyset$. (d) $T_A \neq \emptyset$, $T_B = \emptyset$. For each subcase, we first show the structure of T for that subcase, then its drawing when $W \leq H$, and then its drawing when $H < W$. In Subcases (a) and (d), for simplicity, e is shown to be in the interior of Γ_A, but actually, either it is same as r, or if $W \leq H$ ($H < W$), then it is placed on the bottom (right) boundary of Γ_A. For simplicity, we have shown Γ_A, Γ_B, and Γ_C as identically sized boxes, but in actuality, they may have different sizes.

subtree rooted at u, w is the parent of u, a is the last common node of the path $r \rightsquigarrow v$ and the leftmost path of T, f is the right child of a, T_β is the maximal tree rooted at f that contains w but not u, T_B is the tree consisting of the trees T_α and T_β, 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, and T_C is the tree rooted at d.

In addition to this general case, we get five 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) $T_A \neq \emptyset$, $T_C = \emptyset$, $r \neq e$ (see Figure 2(d)), (e) $T_A \neq \emptyset$, $T_C = \emptyset$, $r = e$ (see Figure 2(e)), and (f) $T_A = \emptyset$, $T_C \neq \emptyset$, $d \neq u^*$ (see Figure 2(f)). 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_β, T_1, and T_2. We also designate e as the link node of T_A, w as the link node of T_β, and u^* as the link node of T_C. We randomly 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 four 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, T_B is the subtree of T rooted at c, e is the parent of u, T_A is the maximal tree rooted at r that contains e but not u, and T_C is the tree rooted at v.

  In addition to the general case, we get the following three special cases: (b) $T_A = \emptyset$, $T_B = \emptyset$ (see Figure 3(b)), (c) $T_A = \emptyset$, $T_B \neq \emptyset$ (see Figure 3(c)), and (d) $T_A \neq \emptyset$, $T_B = \emptyset$ (see Figure 3(d)). 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 e as the link node of T_A, and u^* as the link node of T_C. We randomly select a leaf of T_B and designate it as the link node of T_B.

6.2 Assign Rectangles

Let T_k be a partial tree of T, where for Case 1, T_k is either T_A, T_C, T_β, 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. Algorithm *DrawTree* assigns a rectangle R_k, with width W_k and height H_k, to T_k, where W_k and H_k are as follows: Let W and H be the width and height, respectively, of R. Assume that $H < W$ (the case when $W \le H$ is handled similarly). Let $H' = H - 2$, and $W' = W - 3\max\{n^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2$. Let $x = n_k/n$.

- If $n_k < H'$ (i.e., if T_k is a *very small* tree), then $W_k = \log_2 n_k + 4$, and $H_k = H'$;
- Otherwise, if $xW'/H' \le n_k^{-\epsilon}$ (i.e., if T_k is a *moderately large* tree), then $W_k = H'/n_k^{\epsilon}$, and $H_k = H'$;
- Otherwise (i.e., if T_k is a *large* tree), $W_k = xW'/H'$ and $H_k = H'$.

This assignment strategy ensures that when the drawings of all the partial trees are composed together to obtain a drawing Γ of T in the *Compose Drawings* step, then Γ will fit inside R. It also ensures that, except when T_k is a very small tree, R_k has a good aspect ratio.

6.3 Draw Partial Trees

In Subcase (d) of Case 1, and if $H < W$, then in Subcases (a) and (c) of Case 1, and Subcases (a) and (d) of Case 2, we first change the orientation of R_A, so that its height becomes its width, and its width becomes its height. Also, in Case 1, if $H < W$, and $T_\beta \neq \phi$, we change the orientation of R_β. This is done so because, in these situations, as explained later in Subsection 6.4, we need to rotate R_A and R_β during the *Compose Drawings* step. In all other situations, we do not change the orientation of the rectangle assigned to a partial tree.

Next we draw each partial tree T_k within its assigned rectangle R_k. If T_k is a *large* tree or a *moderately large* tree (see their definitions in Subsection 6.2), then T_k is drawn within R_k by calling Algorithm *DrawTree* recursively for it, and if T_k is a *very small* tree, then T_k is drawn within R_k by calling Algorithm u^*-*HV-Draw* for it. Note that recursion for *DrawTree* stops when either T is empty, or contains exactly one node, or gets designated as a very small tree (in which case, it is drawn using u^*-*HV-Draw*).

6.4 Compose Drawings

Let Γ_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 Γ of T from the drawings of the partial trees in both Cases 1 and 2.

In Case 1, we first construct a feasible drawing Γ_α of the partial tree T_α by composing Γ_1 and Γ_2 as shown in Figure 4, then construct a feasible drawing Γ_B of T_B by composing Γ_α and Γ_β as shown in Figure 5, and finally construct Γ by composing Γ_A, Γ_B and Γ_C as shown in Figure 2.

Γ_α is constructed as follows (see Figure 4):

- If u does not belong to the leftmost path of T, and $T_1 \neq \emptyset$ (see Figure 4(a)), then, if $W \leq H$, place Γ_1 above Γ_2 such that the left boundaries of Γ_1 and Γ_2 are aligned; otherwise (i.e., if $W > H$) place Γ_1 to the left of Γ_2, such that the top boundaries of Γ_1 and Γ_2 are aligned. Place u at one unit to the left and one unit above Γ_1. Draw edges (u, c) and (u, v).
- If u does not belong to the leftmost path of T, and $T_1 = \emptyset$ (see Figure 4(b)), then for both $W \leq H$ and $H < W$, place u at one unit to the left and one unit above Γ_2. Draw edge (u, v).
- Otherwise (i.e., if $u \in$ leftmost path of T), Γ_α is same as Γ_2 (see Figure 4(c)).

Γ_B is constructed as follows (see Figure 5): Let y be the root of T_α. Note that $y = u$ if u does not belong to the leftmost path of T, and $y = v$ otherwise.

- if $T_\beta \neq \emptyset$ (see Figure 5(a)) then, if $W \leq H$, then place Γ_β above Γ_α such that the left boundaries of Γ_β and Γ_α are aligned; otherwise (i.e., if $W > H$), first rotate Γ_β by 90° and then flip it vertically, then place Γ_β to the left of Γ_α such that the top boundaries of Γ_β and Γ_α are aligned. Draw edge (w, y).
- Otherwise (i.e., if $T_\beta = \emptyset$), Γ_B is same as Γ_α (see Figure 5(b)).

Γ is constructed from Γ_A, Γ_B and Γ_C as follows (see Figure 2): Let x be the root of T_B. Note that $x = f$ if $T_\beta \neq \emptyset$, and $x = y$ otherwise.

- In Subcase (a), as shown in Figure 2(a), if $W \leq H$, stack Γ_A, Γ_B, and Γ_c one above the other and place node a between Γ_A and Γ_B, such that Γ_A, a, Γ_B, and Γ_C are separated by unit vertical distance from each other, left boundaries of Γ_B and Γ_C are aligned with each other and are placed at unit horizontal distance to the right of the left boundary of Γ_A, and a is aligned with the left boundary of Γ_A. If $H < W$, then first rotate Γ_A by 90°, and then flip it vertically. Then, place Γ_A, a, Γ_C, and Γ_B from left-to-right in that order, separated by unit horizontal distances, such that a is aligned with the top boundary of Γ_A, the top boundaries of Γ_B and Γ_C are aligned, and are at unit vertical distance below the top boundary of Γ_A. Then, move Γ_C down until u^* becomes the lowest node of Γ. Draw edges (e, a), (a, x), and (a, d).
- In Subcase (b), for both $W \leq H$ and $H < W$, place node r one unit above and left of the top boundary of Γ_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 Γ_C vertically (see Figure 2(c)).
- In Subcase (d), as shown in Figure 2(d), if $W \leq H$, first flip Γ_B vertically, and then flip it horizontally, so that its root x gets placed at its lower-right corner. Then, first rotate Γ_A by 90°, and then flip it vertically. Next, place Γ_A above Γ_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

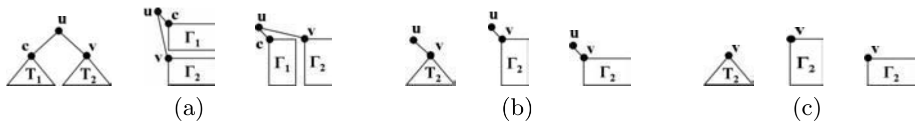

Fig. 4. Drawing T_α, when: (a) u does not belong to the leftmost path of T and $T_1 \neq \emptyset$, (b) u does not belong to the leftmost path of T and $T_1 = \emptyset$, and (c) $u \in$ leftmost path of T. For each case, we first show the structure of T_α for that case, then its drawing when $W \leq H$, and then its drawing when $H < W$. For simplicity, we have shown Γ_1 and Γ_2 as identically sized boxes, but in actuality, their sizes may be different.

it is either to the right of, or aligned with the right boundary of Γ_B (since Γ_A is a feasible drawing, by Property 2, as given in Section 4, moving e will not create any edge-crossings), and then place u^* one unit to the right of and below x. If $H < W$, first rotate Γ_A by 90°, and then flip it vertically. Then, place Γ_A, u^*, and Γ_B left-to-right in that order separated by unit horizontal distances, such that the top boundaries of Γ_A and Γ_B are aligned, and u^* is placed one unit below the bottom boundary of the drawing among Γ_A and Γ_B with greater height. Draw edges (u^*, e) and (u^*, x).

- In Subcase (e), as shown in Figure 2(e), if $W \leq H$, first flip Γ_B vertically, then place Γ_A, Γ_B, and u^* one above the other with unit vertical separation, such that the left boundary of Γ_A is at unit horizontal distance to the left of the left boundary of Γ_B, and u^* is aligned with the left boundary of Γ_A. If $H < W$, then first flip Γ_B vertically, place Γ_A to the left of Γ_B at unit horizontal distance, such that their top boundaries are aligned. Next, move Γ_B down until its bottom boundary is either aligned with or below the bottom boundary of Γ_A. Then, place u^* one unit below the bottom boundary of Γ_B, such that it is aligned with left boundary of Γ_A. Draw edges (u^*, r) and (u^*, x). Note that, since Γ_A is a feasible drawing, 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 Γ_A here (see Figure 2(f)).

In Case 2, we construct Γ by composing Γ_A, Γ_B and Γ_C, as follows (see Figure 3):

- The drawing procedures in Subcases (a), (b), and (c) are similar to those in Subcases (a), (b), and (f), respectively, of Case 1 (see Figures 3(a,b,c)).
- In Subcase (d), as shown in Figure 3(d), if $W \leq H$, we place Γ_A, u, and Γ_C one above the other separated by unit vertical distances such that u is aligned with the left boundary of Γ_A, and the left boundary of Γ_C is one unit to the right of the left boundary of Γ_A. If $H < W$, then first rotate Γ_A by 90°, and then flip it vertically. Then, place Γ_A, u, and Γ_C from left to right in that order, separated by unit horizontal distances, such that u is aligned with the top boundary of Γ_A, and the top boundary of Γ_C is one unit below the top boundary of Γ_A. Draw edges (u, v) and (u, e).

Lemma 3 (Planarity). *Given a binary tree T with a link node u^*, Algorithm* DrawTree *will construct a feasible drawing Γ of T.*

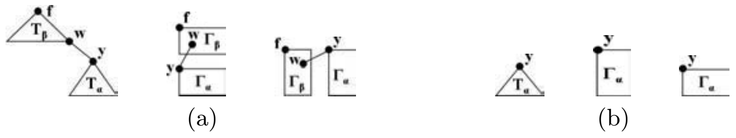

Fig. 5. Drawing T_B when: (a) $T_\beta \neq \emptyset$, and (b) $T_\beta = \emptyset$. Node y shown here is either node u or v. For each case, we first show the structure of T_B for that case, then its drawing when $W \leq H$, and then its drawing when $H < W$. In Case (a), for simplicity, w is shown to be in the interior of Γ_β, but actually, it is either same as f, or if $W \leq H$ $(H < W)$, then is placed on the bottom (right) boundary of Γ_β. For simplicity, we have shown Γ_β and Γ_α as identically sized boxes, but in actuality, their sizes may be different.

Lemma 4 (Time). *Given an n-node binary tree T with a link node u^*, Algorithm DrawTree will construct a drawing Γ of T in $O(n \log n)$ time.*

Lemma 5 (Area). *Given an n-node binary tree T with a link node u^*, Algorithm* DrawTree *can construct a drawing of T within a rectangle with $O(n)$ area.*

Sketch of Proof. The proof is similar to the proof of Theorem 5 of [9]. Let R be a rectangle assigned to T, with aspect ratio A, such that A is in the range $[1, n^\epsilon]$, where ϵ is a constant, such that $1/3 < \epsilon < 1$. Let $D(n)$ be some function such that the area of R should be at least $D(n)$ for the algorithm to succeed in drawing T entirely within R. Let H be the height and W be the width of R. We have two cases: $H < W$ and $W \leq H$.

Consider the case when $H < W$ (the case $W \leq H$ is analogous). Let T_k be a partial tree of T into which T is split, as described in Section 6.1. Let R_k, n_k, x, H_k, W_k, W', and H' be as defined in Section 6.2.

We show that the condition $D(xn) \leq xW'H' = x(W - 3\max\{n^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2)(H - 2)$ is a sufficient condition for Algorithm *DrawTree* to draw T within R. For T to be drawn succesfully within R, each partial tree T_k should be drawn successfully within its assigned rectangle, and their drawings should be succesfully combined to lie entirely within R. If T_k is a very small tree, then it can be drawn within its assigned rectangle R_k using Algorithm u^*-*HV-Draw*, as shown in Lemma 2. If T_k is a moderately large partial tree, then R_k has area equal to $(H'/n_k^\epsilon)H' \geq xW'H'$, and if it is a large partial tree, then R_k has area exactly equal to $xW'H'$. Thus, if $D(xn) \leq xW'H'$, we can draw T_k within its assigned rectangle. In both Case 1 and Case 2, as shown in Figures 2, 3, 4, and 5, Algorithm *DrawTree* uses at most 2 extra horizontal channels and 2 extra vertical channels to combine the drawings of the partial trees to obtain a drawing of T. The rectangles assigned to all the partial trees have equal height, namely, $H' = H - 2$. At least two of the partial trees contain $n/12$ nodes each. Hence, at most three partial trees can get designated as very small or moderately large. We can show that if T_k is a moderately large tree, then it has at most $n^{\frac{1}{1+\epsilon}}$ nodes in it, and $W_k \leq n^{\frac{1-\epsilon}{1+\epsilon}}$. Hence, the total area of the rectangles assigned to the very small and moderately large trees is at most $3\max\{n^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\}(H - 2)$.

Thus, the total area within R available for drawing all the large partial trees is at least $W'H' = (W - 3\max\{n^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2)(H - 2)$. Therefore, if each large partial tree T_k can be drawn within area $xW'H'$, i.e., within its assigned rectangle, then the entire drawing of T can fit within R.

Thus, $D(xn) \le x(W - 3\max\{n^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2)(H - 2)$ is a sufficient condition for drawing T within R.

Since we are interested in finding $D(n)$, suppose the rectangle R assigned to T has area equal to $D(n)$. Then, because H is the shorter side of R, $H \le \sqrt{D(n)}$, and since $A = W/H \le n^\epsilon$, $W \le n^{\epsilon/2}\sqrt{D(n)}$. Substituting W and H by these bounds in $D(xn) \le x(W - 3\max\{n^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2)(H - 2)$ and with some mathematical manipulation, we get that $D(xn) \le x(D(n) - (3\max\{n^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} + 2n^{\epsilon/2} + 2)\sqrt{D(n)})$ is a sufficient condition for drawing T within R. Since $1/3 < \epsilon < 1$, $(1-\epsilon)/(1+\epsilon)$ and $\epsilon/2$ are both less than $1/2$. Hence, $3\max\{n^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} + 2n^{\epsilon/2} + 2 \le cn^\beta$, for some constants c and β, where $c > 0$, and $0 \le \beta < 1/2$. Hence, from Theorem 1, we get that $D(n) = O(n)$.

Theorem 3 (Main Theorem). *Let T be a binary tree with n nodes. Given any number A, where $1 \le A \le n^\alpha$, for some constant α, where $0 \le \alpha < 1$, we can construct in $O(n \log n)$ time, a planar straight-line grid drawing Γ of T with $O(n)$ area, and aspect ratio A.*

Proof. Let ϵ be a constant such that $1 \le A \le n^\epsilon$ and $1/3 < \epsilon < 1$. Assign a rectangle R with area $D(n)$ and aspect ratio A to T, where $D(n)$ is the function defined in the proof of Lemma 5. Note that from the proof of Lemma 5, $D(n) = O(n)$. Construct a drawing Γ of T in R by calling Algorithm *DrawTree* with T, R and ϵ as input. From Lemmas 3, 4, and 5, Γ will be a planar straight-line grid drawing of T contained entirely within R.

References

1. T. Chan, M. Goodrich, S. Rao Kosaraju, and R. Tamassia. Optimizing area and aspect ratio in straight-line orthogonal tree drawings. *Comput. Geom. Theory Appl.* to appear. Prel. version in Proc. Graph Drawing'96, LNCS, vol. 1190, pp. 63-75.
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.
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. C. E. Leiserson. Area-efficient graph layouts (for VLSI). In *Proc. 21st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 270–281, 1980.
7. 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.*, 15:175–2002, 2000.

8. L. Trevisan. A note on minimum-area upward drawing of complete and Fibonacci trees. *Inform. Process. Lett.*, 57(5):231–236, 1996.
9. L. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Comput.*, C-30(2):135–140, 1981.