

Straight-line Planar Grid Drawings of High Degree Trees with Linear Area*

Ashim Garg Adrian Rusu

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

Abstract

Trees are usually drawn planar, i.e. without any crossings. In this paper we investigate the area requirement of (non-upward) planar straight-line drawings of high degree trees. A *degree- d tree* is a tree in which each node has at most d children. Let T be a degree- d tree with n vertices, where d is a positive integer, $d = o(n^{1/2})$. We show that T admits a planar straight-line grid drawing with area $O(n)$ and with any prespecified 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

Trees are very common data-structures, which can be used to model information in a variety of applications such as Software Engineering (hierarchies of object-oriented programs), Business Administration (organization charts), and Web-site Design (structure of a Web-site).

A *drawing* Γ of a rooted 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 an *orthogonal* drawing if each edge is a chain of alternating horizontal and vertical segments. Γ is a *grid* drawing if all the vertices and bends along the edges have integer coordinates. Γ is a *planar* drawings 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. The *area* of a drawing Γ is the number of grid points contained in the smallest rectangle with sides parallel to the axes, that covers the entire drawing. We say that R is the *enclosing rectangle* of Γ . The *width* and *height* of Γ are the *width* and *height* of R , respectively. The *enclosing rectangle* of Γ is the smallest rectangle, with sides parallel to the X and Y axes, respectively, that covers the entire drawing. The *aspect ratio* of Γ is the ratio of the longer and shorter sides of its enclosing rectangle. A tree T is a *binary tree* if each node has at most two children. T is an *ordered* tree if the children of each node are assigned a left-to-right order. We denote by $T[v]$, the *subtree* of T rooted at a node v of T . $T(v)$ consists of v and all the descendants of v . Γ has the *subtree separation* [1] property if for any two vertex-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. It is desirable for the drawing to have the subtree separation property [1] since the users can easily detect the subtrees in such a drawing, and also, it allows for focus+context style [8] rendering of the drawing so that if the tree has too many nodes to fit in the given drawing area, then subtrees close to focus can be shown in detail, whereas those further away from the focus can be contracted and simply shown as filled-in rectangles.

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

2 Our Result

It is desirable to construct straight-line drawings because straight-line edges are easier to follow than polyline edges [1] and hence they are more aesthetically pleasing than polyline drawings. It is also desirable to construct planar drawings because they are visually more appealing than non-planar drawings, i.e., drawings with edge-crossings. Grid drawings guarantee at least unit distance separation between nodes, and the integer coordinates of nodes allow such drawings to be rendered without any distortions due to truncation and rounding-off errors. Several display surfaces, such as computer screen and plain paper, have predefined aspect ratio. Besides, the drawing area available to the user in an application may also have a certain fixed aspect ratio. Therefore, it is desirable to give user control over the aspect ratio of a drawing, so that they can draw a tree in the drawing area available to them. Also, as mentioned before, it is also desirable for the drawing to exhibit the subtree separation property. Finally, it is important to minimize the area of a drawing, so that the users can draw a tree in as small drawing area as possible.

In this paper, we investigate the problem of constructing planar straight-line grid drawings of high degree trees with small area. Clearly, any planar grid drawing of a tree with n nodes requires at least n grid-points to place the nodes, and therefore, has area $\Omega(n)$. A long-standing fundamental question, therefore, has been that whether this is a tight bound, i.e., given a 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 drawing of a high degree (degree- d , d is a positive integer, $d = o(n^{1/2})$) 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 $[1, n^\alpha]$. (We define the aspect ratio of a drawing to be the ratio of its longer and shorter dimensions. So its aspect ratio will always be at least 1.) The drawing also exhibits the subtree separation property.

3 Previous Results

Most of the research on drawing trees has dealt with binary trees [4]. 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 [9]. This bound is very close to $O(n)$, but still it does not settle the question whether binary trees can be drawn in this fashion in $O(n)$ area. Thus, our result is significant from a theoretical view-point. In fact, we already know of one category of drawings, namely, planar upward orthogonal grid drawings, for which $n \log \log n$ is a tight bound [5], i.e., any binary tree can be drawn in this fashion in area $O(n \log \log n)$, and there exists a family of binary trees that require area $\Omega(n \log \log n)$ in any such drawing. So, a natural question arises, if $n \log \log n$ is a tight bound for planar straight-line grid drawings also. Of course, our result imply that this is not the case. Besides, our drawing technique and proofs are significantly different from those of [1] and [9].

We now summarize some other known results on planar grid drawings of binary trees. Let T be a binary tree. [5] presents an algorithm for constructing an upward polyline drawing of T in $O(n)$ area. [6] and [11] present algorithms for constructing an orthogonal drawing of T in $O(n)$ area. [7] gives an algorithm for constructing upward layered drawing in $O(n^2)$ area. [2] gives an algorithm for constructing strictly upward drawing of T in $O(n \log n)$ area. If T is a Fibonacci tree, (AVL tree, balanced tree, respectively), then [2, 10] ([3], [2], respectively) give algorithms for constructing an strictly upward drawing of T in $O(n)$ area.

4 Preliminaries

In this section we give definitions and some known results that will be used throughout the paper.

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 - and Y - axes, respectively, passing through the grid-points.

Let T be a degree- d tree, with one distinguished vertex v , which has at most $d - 1$ children. v is called the *link vertex* of T . A *partial tree* of T is a connected subgraph of T . If T is ordered, 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 vertex of p is called the *leftmost* vertex of T . A *sibling* of a node v of T is a node that has the same parent as v . We denote by $n(T)$, the number of vertices in T .

Let R be a rectangle. Let J be the longer dimension of R , and I be its shorter dimension. Aspect ratio of R , denoted by $ar(R)$ is equal to J/I . Thus, $ar(R) \geq 1$. R has a *good* aspect ratio, if its aspect ratio is in the range $[1, n^\alpha]$, where α is a constant, such that $0 \leq \alpha < 1$. Throughout this paper, we will assume that a rectangle has sides parallel to X and Y axes, respectively.

Let Γ be a drawing of T . We denote by $R(\Gamma)$ the rectangle enclosing a drawing Γ of T . By *bottom* (*top*, *left*, and *right*, respectively) of Γ , we will mean the *bottom boundary* (*top*, *left*, and *right* boundaries, respectively) of its enclosing rectangle. Similarly, by *top-left corner* (*top-right*, *bottom-left*, and *bottom-right* corners, respectively) of Γ , we mean the *top-left corner* (*top-right*, *bottom-left*, and *bottom-right* corners, respectively) of its enclosing rectangle. The *aspect ratio* of Γ is same as the aspect ratio of its enclosing rectangle.

Let r be the root of T . Let u^* be the link vertex of T . Suppose T has n nodes. Γ is a *feasible* drawing of T , if it has the following 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 of Γ . Moreover, we can move u^* downwards in its vertical channel by any distance without causing any edge-crossings in Γ .
- **Property 3:** If $u^* = r$, then no other node or edge of T is placed on or crosses the vertical and horizontal channels occupied by r .

We denote by E the *empty drawing*, i.e., drawing of an empty tree, i.e., tree with zero vertices.

Theorem 1 ([11]) *Let n_0 and α be two constants, where $n_0 \geq 1$ 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 $0 < x \leq 1 - \alpha$, $G(xn) \leq x(G(n) - g(n)\sqrt{G(n)})$, and
- $\forall n \leq n_0$, $G(n) \geq m_n$.

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

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

Theorem 2 (Separation Theorem [11]) *In any degree- d tree T , there is an edge e whose removal splits T into two trees T_1 and T_2 , such that for some x , $1/d \leq x \leq (d-1)/d$, $n(T_1) \leq x \cdot n(T)$ and $n(T_2) \leq (1-x) \cdot n(T)$. Edge e is called a separator of T . Moreover, e can be found in $O(n(T))$ time.*

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

- *rotate operation:* rotate Γ counterclockwise by δ degrees.
After a rotation by δ degrees of Γ , node v is located at grid position $(i \cos \delta - j \sin \delta, i \sin \delta + j \cos \delta)$.
In particular, after a rotation by 90° of Γ , node v is located at grid position $(-j, i)$.
- *flip operation:* flip Γ vertically or horizontally.
After a horizontal flip of Γ , node v is located at grid position $(-i, j)$.
After a vertical flip of Γ , node v is located at grid position $(i, -j)$.

4.1 Our Tree Drawing Algorithm

Let T be a degree- d (d is a positive integer, $d = o(n^{1/2})$) tree with a link vertex 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 the divide-and-conquer paradigm to recursively construct a feasible drawing of T within R . In each recursive step, it splits T into at most $2d + 1$ partial trees by removing at most two vertices and the edges incident on these vertices from it. Each partial tree has at most $\frac{d-1}{d}n$ nodes. Correspondingly, it splits R into at most $2d + 1$ smaller rectangles, and assign each smaller

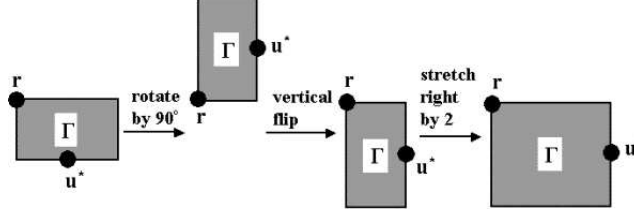


Figure 1: Rotating a drawing Γ first by 90° , followed by flipping it vertically, followed by stretching it to right with the stretch factor of 2. Note that initially vertex u^* was located on the bottom boundary of Γ and after the flip operation u^* is located on the right boundary of Γ .

rectangle to a partial tree. It recursively draws each partial tree into its assigned rectangle, such that the drawings constructed of the partial trees are feasible drawings. Finally, it draws the two vertices and the edges within R , arranging the drawings of the $2d + 1$ partial trees such that there are no edge crossings, and the drawing of the entire tree T fits within R .

The assignment of a rectangle R_k to a partial tree T_k , and the drawing of T_k within R_k is done based on the relative size of T_k compared to the size of T , i.e., on the ratio $x = n_k/n$, where n_k is the number of nodes in T_k . Assume that $H < W$ (the case when $W \leq H$ is analogous). Let $H' = H - 2$, and $W' = W - 2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2$.

- If $n_k < H'$ (i.e., if T_k is a *very small* tree), then $W_k = \log_2 n_k + 4$, and $H_k = H'$. T_k is drawn within R_k using an algorithm called u^* -HV-Draw.
- Otherwise, if $xW'/H' \leq n_k^{-\epsilon}$ (i.e., if T_k is a *moderately large* tree), then $W_k = H'/n_k^\epsilon$, and $H_k = H'$. T_k is drawn recursively within R_k by calling Algorithm *DrawTree* for it.
- Otherwise (i.e., if T_k is a *large* tree), $W_k = xW'$ and $H_k = H'$. T_k is drawn recursively within R_k by calling Algorithm *DrawTree* for it.

This assignment and drawing strategy ensures that the total size of the rectangles assigned to the partial trees is slightly less than the size of R , so that after drawing the partial trees within their respective rectangles, we can put together their drawings, along with the drawings of the removed vertices and the edges, to get a drawing of T that fits within R . It also ensures that, except for the case when T_k has very small size, R_k has a good aspect ratio. This property is crucial for proving that T can be drawn in $O(n)$ area, as shown in the proof of lemma 4. Also, notice that if $H < W$ ($W \leq H$), then all the partial trees will get rectangles with equal height (width), which is equal to $H - 2$ ($W - 2$), and possibly different widths (heights).

Algorithm u^* -HV-Draw is a modification of the classical HV-Drawing algorithm [2], and can be used to construct a feasible drawing of a tree T' with n' nodes within a rectangle with dimensions n' and $\log_2 n' + 4$, where users can specify whether the longer dimension is width or height:

Lemma 1 *Let T be a n -node degree- d 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 $\log_2 n + 4$, as well as within a rectangle with width $\log_2 n + 4$ and height n .*

Details of Algorithm u^* -HV-Draw are given in the Appendix.

Algorithm Place- u^* orders the children of each node of T such that u^* becomes the leftmost vertex of T .

Algorithm Place- u^*

Input: A degree- d tree T with a designated link vertex u^* .

Output: Tree T such that its link vertex u^* is the leftmost vertex.

```

if  $u^*$  is not the leftmost vertex of  $T$  then {
    if  $u^*$  has a leftmost child then remove its leftmost vertex and place it in the next available (empty)
    position;
    Swap all the nodes, except the root, of the path from  $u^*$  to root with their leftmost sibling;
}

```

```

    Return  $T$ ;
}
Return  $T$ ;

```

end Algorithm.

To split T into the partial trees, *DrawTree* first calls Algorithm *Place- u^** to place u^* as the leftmost vertex of T .

Next, it finds a separator edge (u, v) of T , where v is the parent of u . 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 four 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' is the tree rooted at v ,
 - $c_1, \dots, c_j$ are the siblings of v , and $T_1, \dots, T_j$ are the trees rooted at $c_1, \dots, c_j$ respectively, $0 \leq j \leq d-1$,
 - T_α is the tree rooted at u ,
 - w is the parent of u ,
 - a is the last common vertex of the path $r \rightsquigarrow v$ and the leftmost path of T ,
 - f is a child of a , but not the leftmost,
 - 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 g is the leftmost 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 g .
 - $b_1, \dots, b_i$ are the siblings of f and d , and $T^1, \dots, T^i$ are the trees rooted at $b_1, \dots, b_i$ respectively, $0 \leq i \leq d-2$.

In addition to this general case, we get three special cases: (b) when $T_A \neq \emptyset$ and $T_C = \emptyset$ (see Figure 2(b)), (c) when $T_A = \emptyset$ and $T_C \neq \emptyset$ (see Figure 2(c)), and (d) when $T_A = \emptyset$ and $T_C = \emptyset$ (see Figure 2(d)). Note that, since (u, v) is a separator edge, T' has at least $(1/d)n$ nodes and so $T_B \neq \emptyset$. Some of these subcases have further subsubcases. In all the cases, we designate e as the link vertex of T_A , w as the link vertex of T_β , and u^* as the link vertex of T_C . We randomly choose a leaf e_i of T^i , $0 \leq i \leq d-2$, a leaf e_j of T_j , $0 \leq j \leq d-1$, and a leaf e' of T' and designate them as the link vertices of T^i , $0 \leq i \leq d-2$, T_j , $0 \leq j \leq d-1$, and T' respectively.

- *Case 2: The separator edge (u, v) is in the leftmost path of T .* In this case, T has the general form in Figure 3(a), where
 - r is the root of T ,
 - $c_1, \dots, c_j$ are the siblings of v , and $T_1, \dots, T_j$ are the trees rooted at $c_1, \dots, c_j$ respectively, $1 \leq j \leq d-1$,
 - e is the parent of u ,
 - T_A is the maximal tree rooted at r that contains e but not u , and
 - $T_C = T'$ is the tree rooted at v .

In addition to the general case, we get the following three special cases: (b) when $T_A = \emptyset$ and $1 \leq j \leq d-1$ (see Figure 3(b)), (c) when $T_A \neq \emptyset$ and $j = 0$ (see Figure 3(c)), and (d) when $T_A = \emptyset$ and $j = 0$ (see Figure 3(d)). Again, note that, since (u, v) is a separator edge, $T_C \neq \emptyset$. Some of these subcases have further subsubcases. In all these cases, we designate e as the link vertex of T_A , and u^* as the link vertex of T_C . We randomly choose a leaf e_j of T_j and designate it as the link vertex of T_j , $0 \leq j \leq d-1$.

In each case, the at most $2d+1$ partial subtrees that T gets split into are T_A , T_C , T_β , T' , $T^1, \dots, T^i$,

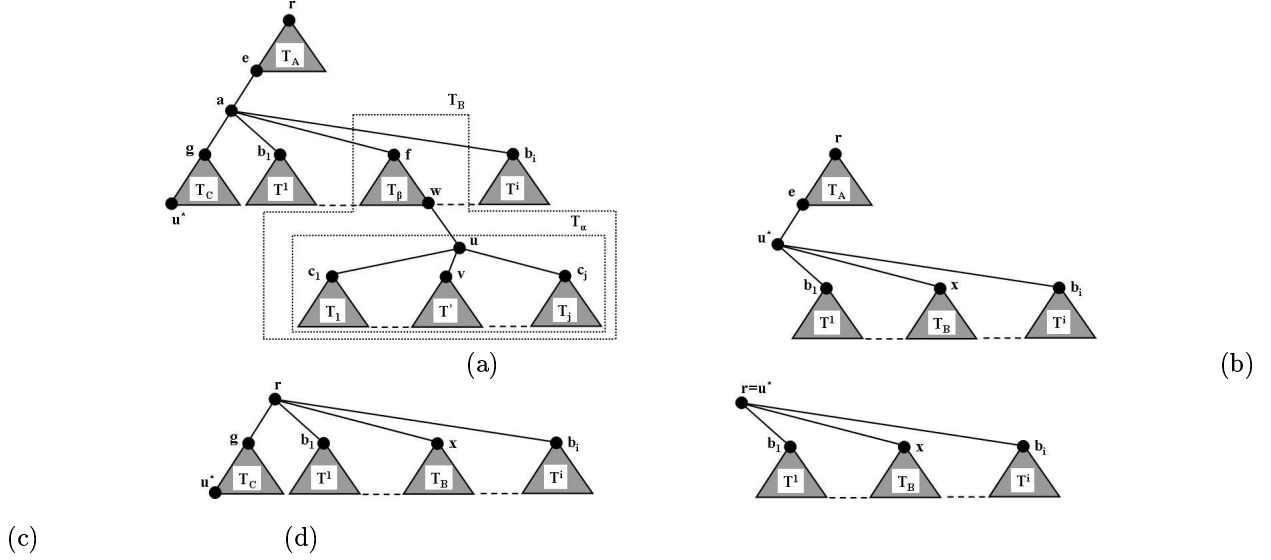


Figure 2: Case 1 (Separator edge (u, v) does not belong to the leftmost path of T), $0 \leq i \leq d - 2$: (a) General form, i.e., $T_A \neq \emptyset$ and $T_C \neq \emptyset$; (b) $T_A \neq \emptyset$ and $T_C = \emptyset$; (c) $T_A = \emptyset$ and $T_C \neq \emptyset$; (d) $T_A = \emptyset$ and $T_C = \emptyset$.

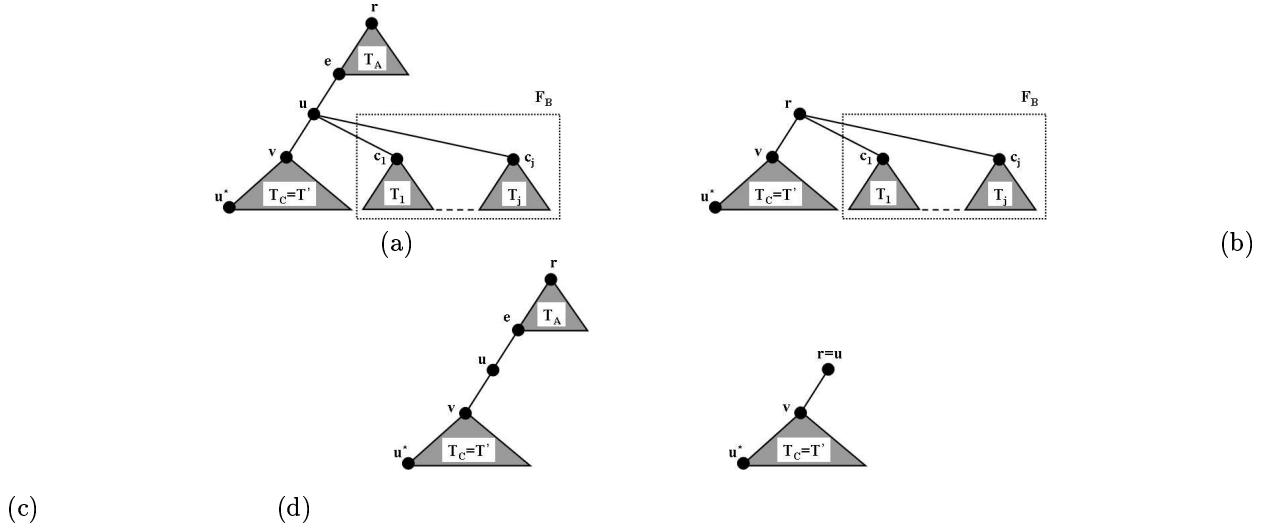


Figure 3: Case 2 (Separator edge (u, v) belongs to the leftmost path of T): (a) General form, i.e., $T_A \neq \emptyset$, $1 \leq j \leq d - 1$; (b) $T_A = \emptyset$ and $1 \leq j \leq d - 1$; (c) $T_A \neq \emptyset$ and $j = 0$; (d) $T_A = \emptyset$, $j = 0$.

$0 \leq i \leq d - 2$, and $T_1, \dots, T_j$, $0 \leq j \leq d - 1$. a and u are the at most two vertices whose removal splits T into these partial trees. Note that the edges that also get removed when we remove a and u are (a, e) , (g, a) , (f, a) , (u, w) , (v, u) , $(c_1, u), \dots, (c_j, u)$, $0 \leq j \leq d - 1$, and $(b_1, a), \dots, (b_i, a)$, $0 \leq i \leq d - 2$.

We give below a pseudocode for Algorithm *DrawTree*. In this pseudocode, it assigns rectangles to T_A , T_C , T_B , $T^1, \dots, T^i$, $0 \leq i \leq d - 2$, and $T_1, \dots, T_j$, $0 \leq j \leq d - 1$ by calling Algorithm *AssignRectangle*. It calls itself recursively to draw T_A , T_C , and $T^1, \dots, T^i$, $0 \leq i \leq d - 2$, and calls Algorithm *DrawTreeB* to draw tree T_B . In the end, it calls Algorithm *Comp* to put together the drawings of T_A , T_B , $T^1, \dots, T^i$, $0 \leq i \leq d - 2$, and T_C , and to draw a and its incident edges, to construct a drawing of T . If $W \leq H$, then *Comp* stacks the drawings of T_A , T_B , $T^1, \dots, T^i$, $0 \leq i \leq d - 2$, and T_C one above the other, and places a appropriately such that there are no edge crossings when drawing edges incident on a , as shown in the second column of Figures 6 and 7. If $H < W$, then *Comp* puts the drawings of T_A , T_B , $T^1, \dots, T^i$, $0 \leq i \leq d - 2$,

and T_C side-by-side, and places a appropriately such that there are no edge crossings when drawing edges incident on a , as shown in the third column of Figures 6 and 7. Details of Algorithm *Comp* are given in the Appendix.

DrawTreeB calls *DrawTree* to draw T_β , and calls Algorithm *DrawTree α* to draw T_α . In the end, it calls Algorithm *CompB* to put together the drawings of T_α and T_β and to draw the edge (u, w) , to construct a feasible drawing of T_B . From Theorem 2, $T_\alpha \neq \emptyset$. Hence, we have two cases: (a) $T_\beta \neq \emptyset$, and (b) $T_\beta = \emptyset$ (see Figure 5). When $T_\beta \neq \emptyset$, then if $W \leq H$ ($H < W$), *CompB* puts the drawings of T_β above (to the left of) T_α at unit distance from it, such that their left (top) boundaries are aligned, as shown in the second (third) column of Figure 5. Then it draws the edge that connects w with the root of T_α . When $T_\beta = \emptyset$, then *CompB* simply returns the drawing of T_α as the drawing of T_B .

DrawTree α calls *DrawTree* to draw T' and $T_1, \dots, T_j$, $0 \leq j \leq d-1$. In the end, it calls Algorithm *Comp α* to put together the drawings of T' and $T_1, \dots, T_j$, $0 \leq j \leq d-1$, and to draw u and its incident edges, to construct a drawing of T_α . From Theorem 2, $T' \neq \emptyset$. We have three cases (see Figure 4): (a) $u \notin$ the leftmost path of T and $1 \leq j \leq d-1$, (b) $u \notin$ the leftmost path of T and $j = 0$, and (c) $u \in$ the leftmost path of T . In Case (a), if $W \leq H$ ($H < W$), as shown in the second (third) column of Figure 4(a), *Comp α* puts the drawings $\Gamma_1, \dots, \Gamma_j$ of $T_1, \dots, T_j$, respectively, $0 \leq j \leq d-1$, and Γ' of T' one above (to the left of) the other at unit distance between them. It places u one unit to the left of and one unit above Γ_1 . Then it draws edges $(u, c_1), \dots, (u, c_j)$, $0 \leq j \leq d-1$, and (u, v) . In Case (b), it just places u one unit to left and one unit above Γ_1 (see Figure 4(b)). Then it draws edge (u, v) . In Case (c), it simply returns Γ' as the drawing of T_α .

Algorithm DrawTree

Input: An n -node degree- d (d is a positive integer, $d = o(n^{1/2})$) tree T with the link vertex u^* , a constant ϵ , where $1/3 < \epsilon < 1$, and a rectangle R with aspect ratio A , where $1 \leq A \leq n^\epsilon$. Let W and H be the width and height, respectively, of R .

Output: A feasible drawing Γ of T within R .

if $n = 0$ then Return E (Base Case 1) ; (Return Empty Drawing)

else if $n = 1$ (Base Case 2) then {

Let Γ be the drawing containing exactly one vertex;

Return Γ ;

}

else if T is a small tree (Base case 3) then {

Let $\Gamma = u^* \text{-HV-Draw}(T, R)$;

Return Γ ;

}

Let $T = \text{Place-}u^*(T)$;

Find a separator edge (u, v) of T .

if $(u, v) \notin$ leftmost path of T then {

Identify trees $T_A, T_B, T_C, T_\alpha, T_\beta, T^1, \dots, T^i$, $0 \leq i \leq d-2$, and $T_1, \dots, T_j$, $0 \leq j \leq d-1$, as in Figure 2;

Designate vertex e as the link vertex of T_A , w as the link vertex of T_β , and u^* as the link vertex of T_C (See Figure 2). Randomly choose a leaf e_i of T_i , $0 \leq i \leq d-2$, a leaf e_j of T_j , $0 \leq j \leq d-1$, and a leaf e' of T' , and designate them as the link vertices of T_i , $0 \leq i \leq d-2$, T_j , $0 \leq j \leq d-1$, and T' , respectively.

}

else ($(u, v) \in$ leftmost path of T) {

Identify trees T_A, T_C , and forest F_B (trees $T_1, \dots, T_j$, $0 \leq j \leq d-1$) as in Figure 3;

Designate e as the link vertex of T_A , and u^* as the link vertex of T_C (See Figure 3). We randomly choose a leaf e_j of T_j , and designate it as the link vertex of T_j , $0 \leq j \leq d-1$.

```

}
Let  $R(T_A) = \text{AssignRectangle}(T_A, T, R, \epsilon)$ ;
Let  $R(T_C) = \text{AssignRectangle}(T_C, T, R, \epsilon)$ ;
if  $(u, v) \notin \text{leftmost path of } T$  then {
    Let  $R(T_\alpha) = \text{AssignRectangle}(T_\alpha, T, R, \epsilon)$ ;
    Let  $R(T_\beta) = \text{AssignRectangle}(T_\beta, T, R, \epsilon)$ ;
    for  $k = 0$  to  $i$  do Let  $R(T^k) = \text{AssignRectangle}(T^k, T, R, \epsilon)$ ;
    Let  $R(T') = \text{AssignRectangle}(T', T, R, \epsilon)$ ;
}
else  $( (u, v) \in \text{leftmost path of } T )$ 
    for  $k = 0$  to  $j$  do Let  $R(T_k) = \text{AssignRectangle}(T_k, T, R, \epsilon)$ ;
if  $(u, v) \notin \text{leftmost path of } T$  then {
    if  $W \leq H$  then if  $T_A \neq \emptyset$ ,  $T_C = \emptyset$ , and  $r \neq e$  then Change the orientation of  $R(T_A)$ ; ( the width of  $R(T_A)$  becomes its height and the height of  $R(T_A)$  becomes its width )
    else  $( H < W )$  {
        if  $T_\beta \neq \emptyset$  then Change the orientation of  $R(T_\beta)$ ; ( the width of  $R(T_\beta)$  becomes its height and the height of  $R(T_\beta)$  becomes its width )
        if  $(T_A \neq \emptyset \text{ and } T_C \neq \emptyset)$  or  $(T_A \neq \emptyset, T_C = \emptyset, \text{ and } r \neq e)$  then Change the orientation of  $R(T_A)$ ; ( the width of  $R(T_A)$  becomes its height and the height of  $R(T_A)$  becomes its width )
    }
}
}
else  $( (u, v) \in \text{leftmost path of } T )$  {
    if  $T_A \neq \emptyset$  and  $H < W$  then Change the orientation of  $R(T_A)$ ; ( the width of  $R(T_A)$  becomes its height and the height of  $R(T_A)$  becomes its width )
}
Let  $\Gamma_A = \text{DrawTree}(T_A, R(T_A), \epsilon)$ ;
if  $(u, v) \notin \text{leftmost path of } T$  then
    Let  $\Gamma_B = \text{DrawTreeB}(T_B, R(T_\alpha), R(T_\beta), R(T^1), \dots, R(T^i), R(T'), W, H, \epsilon)$ ;
Let  $\Gamma_C = \text{DrawTree}(T_C, R(T_C), \epsilon)$ ;
if  $(u, v) \notin \text{leftmost path of } T$  then Let  $\Gamma = \text{Comp}(\Gamma_A, \Gamma_B, \Gamma_C, \Gamma^1, \dots, \Gamma^i, R)$ ;
else  $( (u, v) \in \text{leftmost path of } T )$  Let  $\Gamma = \text{Comp}(\Gamma_A, \Gamma_C, \Gamma_1, \dots, \Gamma_j, R)$ ;
( see the Appendix for description of Algorithm Comp )
Return  $\Gamma$ ;

```

end Algorithm.

Algorithm AssignRectangle

Input: T_k , T , $R(T)$, and ϵ , where $1/3 < \epsilon < 1$. Let I and J be the dimensions of R , where $J \geq I$.

Output: A rectangle $R(T_k)$ whose one dimension is equal to $I - 2$ and the other dimension is either equal to $\log_2 n(T_k) + 4$, or is at least $(I - 2)/n_k^\epsilon$.

```

Let  $I_k = I' = I - 2$ ;
Let  $J' = J - 2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2$ 
Let  $x = n_k/n$ ;
if  $n_k < I'$  then  $J_k = \log_2 n_k + 4$ ; (  $T_k$  is a very small tree )
else if  $xJ'/I' < n_k^{-\epsilon}$  then  $J_k = I'/n_k^\epsilon$ ; (  $T_k$  is a moderately large tree )
else  $J_k = xJ'$ ; (  $T_k$  is a large tree )

```

Let $R(T_k)$ be a rectangle with dimensions I_k and J_k , where if I is the height of R then I_k is the height of R_k and J_k is its width, otherwise I_k is the width of R_k and J_k is its height;
 Return $R(T_k)$;

end Algorithm.

Algorithm DrawTreeB

Input: $T_B, R(T_\alpha), R(T_\beta), R(T_1), \dots, R(T_j), R(T'), W, H$, and ϵ , where $1/3 < \epsilon < 1$ and $0 \leq j \leq d-1$.
Output: A feasible drawing Γ_B of T_B .

Let $\Gamma_\alpha = \text{DrawTree}\alpha(T_\alpha, R(T_1), \dots, R(T_j), R(T'), W, H, \epsilon)$;
 Let $\Gamma_\beta = \text{DrawTree}(T_\beta, R(T_\beta), \epsilon)$;
 Let $\Gamma_B = \text{CompB}(\Gamma_\alpha, \Gamma_\beta, W, H)$;
 Return Γ_B ;

end Algorithm.

Algorithm DrawTree α

Input: $T_\alpha, R(T_1), \dots, R(T_j), R(T'), W, H$, and ϵ , where $1/3 < \epsilon < 1$ and $0 \leq j \leq d-1$.
Output: A feasible drawing Γ_α of T_α .

for $k = 0$ **to** j **Let** $\Gamma_k = \text{DrawTree}(T_k, R(T_k), \epsilon)$;
Let $\Gamma' = \text{DrawTree}(T', R(T'), \epsilon)$;
Let $\Gamma_\alpha = \text{Comp}\alpha(\Gamma_1, \dots, \Gamma_j, \Gamma', W, H)$;
Return Γ_α ;

end Algorithm.

Algorithm Comp α

Input: $\Gamma_1, \dots, \Gamma_j, \Gamma', W$, and H , where $0 \leq j \leq d-1$.
Output: A feasible drawing Γ_α of T_α .

if $u \notin$ the leftmost path of T and $j \neq 0$ (see Figure 4(a)) **then** {
 if $W \leq H$ **then** Place $\Gamma_1, \dots, \Gamma_j$ and Γ' one above the other separated by unit distance such that the left boundaries of $\Gamma_1, \dots, \Gamma_j$, and Γ' are aligned;
 else ($H < W$) Place $\Gamma_1, \dots, \Gamma_j$, and Γ' to the left of the other separated by unit distance such that the top boundaries of $\Gamma_1, \dots, \Gamma_j$ and Γ' are aligned;
 Place u at one unit to the left of and one unit above Γ_1 ;
 Draw edges $(u, c_1), \dots, (u, c_j)$, and (u, v) ;
 Let Γ_α be the drawing thus obtained;
 }
else if $u \notin$ the leftmost path of T and $j = 0$ (see Figure 4(b)) **then** {
 Place u at one unit to the left of and one unit above Γ' ;
 Draw edge (u, v) ;
 Let Γ_α be the drawing thus obtained;
 }
else ($u \in$ the leftmost path of T (see Figure 4(c))) **Let** $\Gamma_\alpha = \Gamma'$;

Return Γ_α ;

end Algorithm.

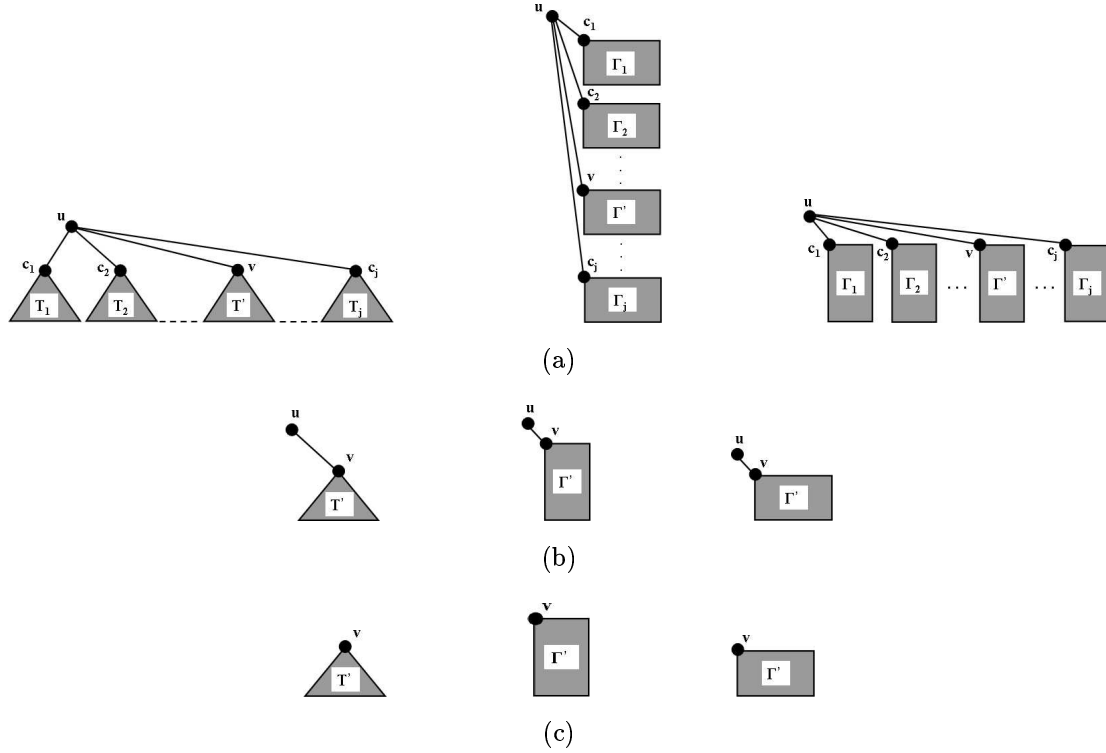


Figure 4: Drawing T_α , when: (a) u does not belong to the leftmost path of T and $1 \leq j \leq d-1$, (b) u does not belong to the leftmost path of T and $j = 0$, and (c) $u \in$ leftmost path of T . The first column shows the three cases. The second column shows the drawings when $W \leq H$. The third column shows the drawings when $H < W$. Here, Γ_j is the drawing of T_j , $1 \leq j \leq d-1$, and Γ' is the drawing of T' . For convenience, we have shown $\Gamma_1, \dots, \Gamma_j$, and Γ' as having same the width in the second column and the same height in the third column. In actuality, their heights and widths may be different.

Algorithm CompB

Input: $\Gamma_\alpha, \Gamma_\beta, W$, and H .

Output: A feasible drawing Γ_B of T_B .

```

if  $T_\beta \neq \emptyset$  (see Figure 5(a)) then {
  if  $W \leq H$  then Place  $\Gamma_\beta$  above  $\Gamma_\alpha$  separated by unit distance such that the left boundaries of  $\Gamma_\beta$ 
  and  $\Gamma_\alpha$  are aligned;
  else (  $H < W$  ) then {
    First rotate by  $90^\circ$  and then flip  $\Gamma_\beta$  vertically;
    Place  $\Gamma_\beta$  to the left of  $\Gamma_\alpha$  separated by unit distance such that the top boundaries of  $\Gamma_\beta$  and  $\Gamma_\alpha$ 
    are aligned;
  }
  Draw edge  $(w, y)$ ;
  Let  $\Gamma_B$  be the drawing obtained;
}

```

else ($T_\beta = \emptyset$ (see Figure 5(b))) Let $\Gamma_B = \Gamma_\alpha$;
Return Γ_B ;

end Algorithm.

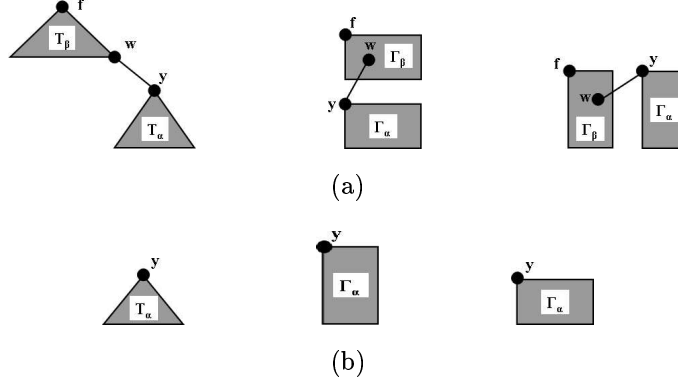


Figure 5: Drawing T_B when: (a) $T_\beta \neq \emptyset$, and (b) $T_\beta = \emptyset$. Here, Γ_β and Γ_α are the drawings of T_β and T_α , respectively. The first column shows the two cases. The second column shows the drawings when $W \leq H$. The third column shows the drawings when $H < W$. For convenience, we have shown Γ_β , and Γ_α as having same the width in the second column and the same height in the third column. In actuality, their height and widths may be different. For convenience, we have shown w as being somewhere inside Γ_β . Actually, since Γ_β is a feasible drawing, either $w = f$, or w will be at the bottom of Γ_β .

Lemma 2 (Planarity) *Let T be a degree- d (d is a positive integer, $d = o(n^{1/2})$) tree with a link vertex u^* given as input to Algorithm DrawTree. Algorithm DrawTree will construct a feasible drawing Γ of T .*

Proof: Proof is by induction over the size of T , i.e., over the number of nodes n in T .

When T has at most one vertex, this is trivially true (this corresponds to Base Case 2 of Algorithm DrawTree).

If T gets drawn in Base Case 3 of Algorithm DrawTree, then it is drawn using Algorithm u^* -HV-Draw. From Lemma 1, this drawing is a feasible drawing.

In the general recursive step, we have two cases: Case 1 (when the separator edge (u, v) is not in the leftmost path), and Case 2 (when the separator edge (u, v) is in the leftmost path).

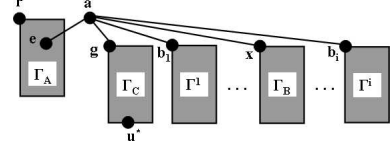
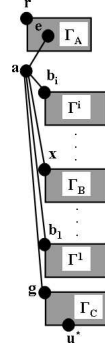
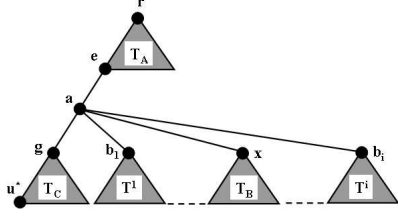
Consider Case 1. Assume that we have already constructed feasible drawings $\Gamma_A, \Gamma_C, \Gamma_\beta, \Gamma^1, \dots, \Gamma^i$, $0 \leq i \leq d-2$, $\Gamma_1, \dots, \Gamma_j$, $0 \leq j \leq d-1$, and Γ' of the partial trees $T_A, T_C, T_\beta, T^1, \dots, T^i$, $0 \leq i \leq d-2$, $T_1, \dots, T_j$, $0 \leq j \leq d-1$, and T' , respectively, of T . Recall that a feasible drawing has Properties 1, 2, and 3, as given in the definition of feasible drawing in Section 4.

Because $\Gamma_1, \dots, \Gamma_j$, $0 \leq j \leq d-1$, and Γ' are feasible drawings, in all the cases, as shown in Figure 4, Algorithm CompA can draw edges $(u, c_1), \dots, (u, c_j)$, $0 \leq j \leq d-1$, and (u, v) without any edge crossings, and as a result, Drawing Γ_α constructed by Algorithm DrawTree α is a feasible drawing.

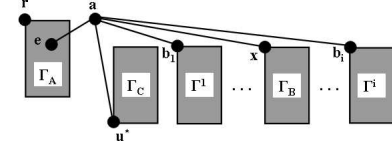
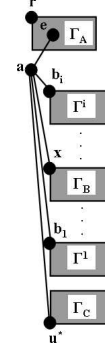
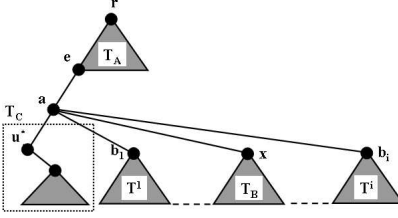
Because Γ_β and Γ_α are feasible drawings, in all the cases, as shown in Figure 5, Algorithm CompB can draw the edge (w, y) without any edge crossings, and as a result, Drawing Γ_B constructed by Algorithm DrawTreeB is also a feasible drawing. Note that in Figure 5, we have shown w to be somewhere inside Γ_β , but because of Properties 2 and 3, it is either the same as the root of T_β , or is at the bottom of Γ_β .

Because $\Gamma_A, \Gamma_B, \Gamma_C$, and $\Gamma^1, \dots, \Gamma^i$, $0 \leq i \leq d-2$, are feasible drawings, in all the cases, as shown in Figure 6, Algorithm Comp can draw the edges (e, a) , (a, g) , (a, x) , and $(a, b_1), \dots, (a, b_i)$, $0 \leq i \leq d-2$, without any edge crossings, and as a result, Drawing Γ of T constructed by Algorithm DrawTree is also a feasible drawing. Note that in Figure 6 we have shown e to be somewhere inside Γ_A , but because of Properties 1 and 2, it is either the same as the root of T , or is at the bottom of Γ_A .

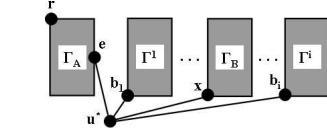
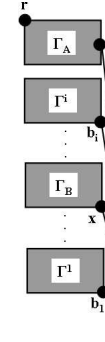
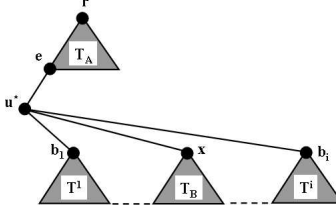
Now Consider Case 2. Assume that we have already constructed feasible drawings Γ_A, Γ_C , and $\Gamma_1, \dots, \Gamma_j$, $0 \leq j \leq d-1$, of T_A, T_C , and $T_1, \dots, T_j$, $0 \leq j \leq d-1$, respectively. Using the same reasoning as above, we



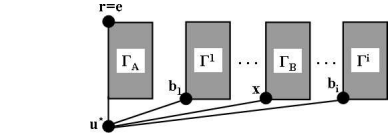
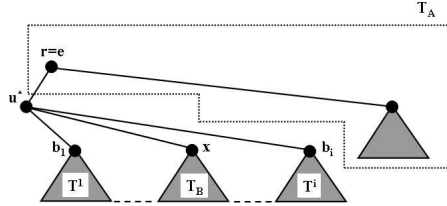
(a)



(b)



(c)



(d)

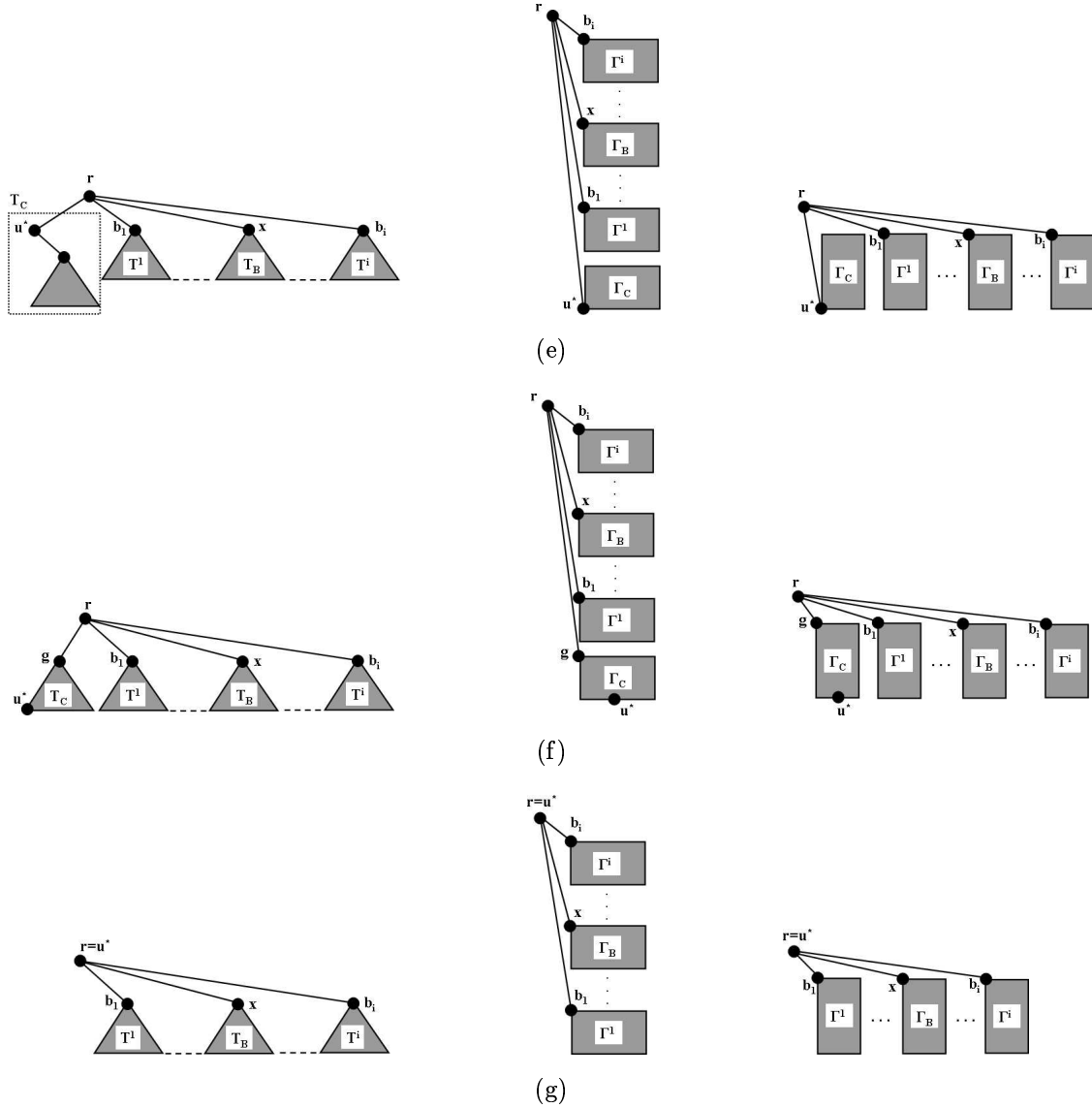


Figure 6: Drawing T in all the subcases of Case 1 (when the separator edge (u, v) is not in the leftmost path of T): (a) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $g \neq u^*$, $0 \leq i \leq d-2$. (b) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $g = u^*$, $0 \leq i \leq d-2$. (c) $T_A \neq \emptyset$, $T_C = \emptyset$, $r \neq e$, $0 \leq i \leq d-2$. (d) $T_A \neq \emptyset$, $T_C = \emptyset$, $r = e$, $0 \leq i \leq d-2$. (e) $T_A = \emptyset$, $T_C \neq \emptyset$, $g = u^*$, $0 \leq i \leq d-2$. (f) $T_A = \emptyset$, $T_C \neq \emptyset$, $g \neq u^*$, $0 \leq i \leq d-2$. (g) $T_A = \emptyset$, $T_C = \emptyset$, $0 \leq i \leq d-2$. Node x shown here is either node f or u . Here, the first column shows the subcases. The second column shows the drawings when $W \leq H$. The third column shows the drawings when $H < W$. Here, Γ_A , Γ_B and Γ_C are the drawings of T_A , T_B and T_C , respectively. For convenience, we have shown Γ_A , Γ_B and Γ_C as having same the width in the second column and the same height in the third column. Note that when $W \leq H$ ($H < W$), the assigned rectangles of T_A , T_B and T_C will have the same width, but the widths (height) of their actual drawings Γ_A , Γ_B and Γ_C may be different from each other. For convenience, we have shown e as being somewhere inside Γ_A . Actually, since Γ_A is a feasible drawing, either $e = r$, or e will be at the bottom of Γ_A .

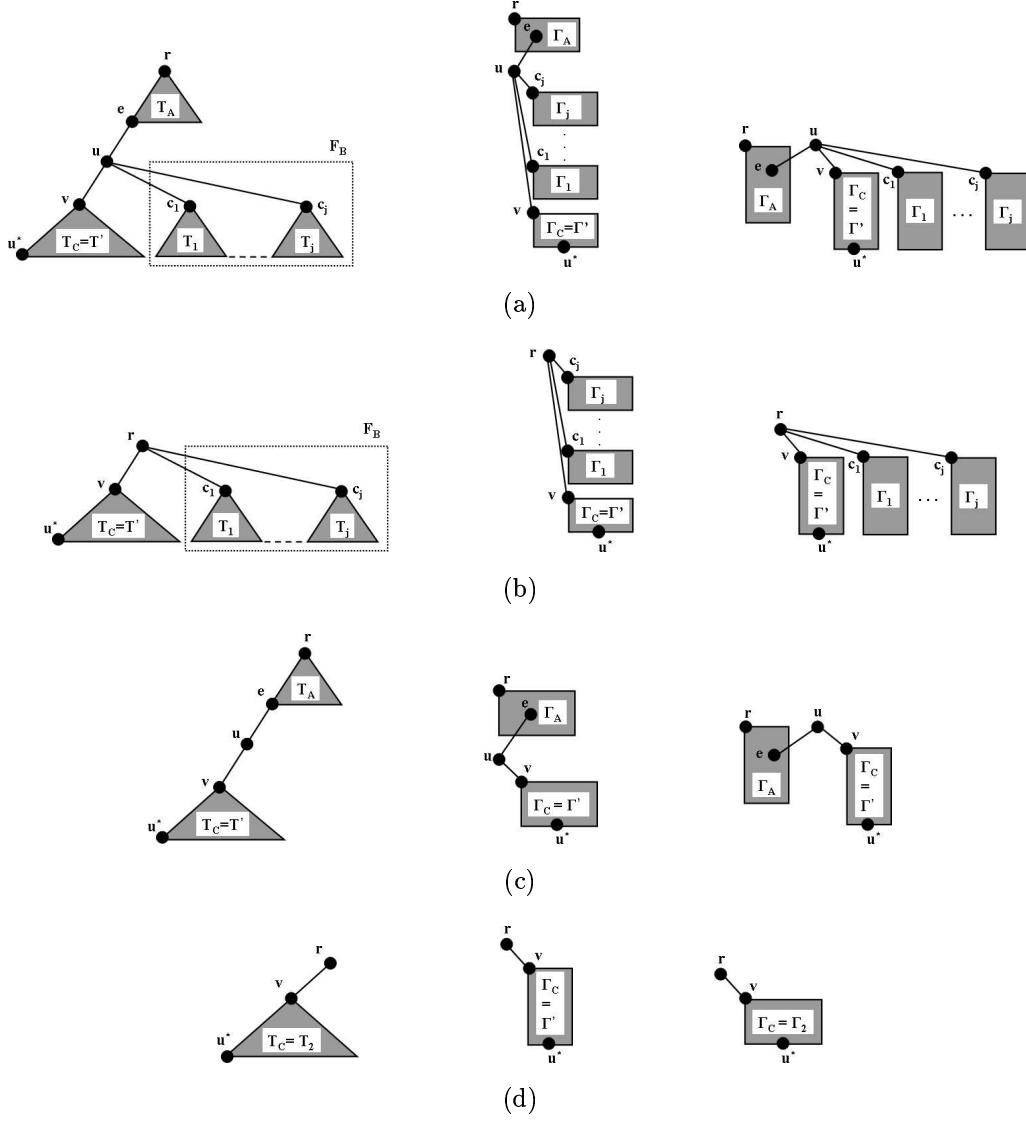


Figure 7: Drawing T in all the subcases of Case 2 (when none of the vertices u and v is in the leftmost path of T): (a) $T_A \neq \emptyset$, $1 \leq j \leq d-1$. (b) $T_A = \emptyset$, $1 \leq j \leq d-1$. (c) $T_A \neq \emptyset$, $j = 0$. (d) $T_A = \emptyset$, $j = 0$. Here, the first column shows the subcases. The second column shows the drawings when $W \leq H$. The third column shows the drawings when $H < W$. Here, Γ_A , Γ_B and Γ_C are the drawings of T_A , T_B and T_C , respectively. For convenience, we have shown Γ_A , Γ_B and Γ_C as having same the width in the second column and the same height in the third column. Note that when $W \leq H$ ($H < W$), the assigned rectangles of T_A , T_B and T_C will have the same width, but the widths (height) of their actual drawings Γ_A , Γ_B and Γ_C may be different from each other. For convenience, we have shown e as being somewhere inside Γ_A . Actually, since Γ_A is a feasible drawing, either $e = r$, or e will be at the bottom of Γ_A .

can prove that Algorithm *Comp* will construct a feasible drawing of T in this case also. $\square$

Lemma 3 (Time Complexity) *Let T be a n -node degree- d (d is a positive integer, $d = o(n^{1/2})$) tree with a link vertex u^* given as input to Algorithm *DrawTree*. Algorithm *DrawTree* will construct a drawing Γ of T in $O(n \log n)$ time.*

Proof: By Theorem 2, in each recursive step, each of the trees $T'_A, T'_\beta, T^{1'}, \dots, T^{i'}, 0 \leq i \leq d-2, T'_1, \dots, T'_j, 0 \leq j \leq d-1$, and T'_C obtained by splitting a tree T' with n' nodes and a link vertex u' , contains at most $\frac{d-1}{d}n'$ nodes. Hence, in the worst case (d is a constant), the depth of the recursion of Algorithm *DrawTree* is $O(\log n)$. In each recursive step, it can make u' the leftmost vertex of T in $O(\frac{n'}{d-1})$ time, assign rectangles to $T'_A, T'_\beta, T^{1'}, \dots, T^{i'}, 0 \leq i \leq d-2, T'_1, \dots, T'_j, 0 \leq j \leq d-1$, and T'_C , and combine their drawings to obtain a drawing of T' in $O(d)$ time. Hence, the overall running time of *DrawTree* is $O(n \log n)$. $\square$

Lemma 4 (Area) *Let T be a n -node degree- d (d is a positive integer, $d = o(n^{1/2})$) tree with a link vertex u^* given as input to Algorithm *DrawTree*. Algorithm *DrawTree* can construct a drawing of T within a rectangle with $O(n)$ area.*

Proof:

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

Consider the case when $H < W$ (the case $W \leq H$ is analogous).

In each subcase of Case 1 (see Figure 2) T is divided into at most $2d + 1$ partial trees: $T_A, T_C, T_\beta, T^1, \dots, T^i, 0 \leq i \leq d-2, T_1, \dots, T_j, 0 \leq j \leq d-1$, and T' . Similarly, in each subcase of Case 2 (see Figures 3), T is divided into at most $d + 1$ subtrees: T_A, T_C , and $T_1, \dots, T_j, 0 \leq j \leq d-1$. In each of these subcases, for Algorithm *DrawTree* to succeed, it is necessary that the area of the rectangle R_k that is assigned to each tree T_k (where T_k is $T_A, T_C, T_\beta, T^1, \dots, T^i, 0 \leq i \leq d-2, T_1, \dots, T_j, 0 \leq j \leq d-1$, or T'), should be at least $D(n_k)$, where n_k is the number of nodes in T_k . Let H_k be the height and W_k be the width of R_k .

The algorithm assigns R_k to T_k , and draws T_k inside R_k as follows. Let $x = n_k/n$. Let $H' = H - 2$ and $W' = W - 2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2$.

1. If T_k has a very small size, i.e., if $n_k < H'$ then $W_k = \log_2 n_k + 4$, and $H_k = H'$. T_k is drawn in R_k using Algorithm u^* -HV-Draw. Since $W_k = \log_2 n_k + 4$ and $H_k = H' > n_k$, from Lemma 1, Algorithm u^* -HV-Draw will succeed in drawing T_k within R_k .
2. Otherwise, if T_k has a moderate size, i.e., if $xW'/H' \leq n_k^{-\epsilon}$, then $W_k = H'/n_k^\epsilon$ and $H_k = H'$. In this case, the area of R_k is equal to $H' \cdot H'/n_k^\epsilon \geq H'xW' = xW'H'$ (since $xW'/H' < n_k^{-\epsilon}$). T_k is drawn recursively in R_k . Thus, Algorithm *DrawTree* can successfully draw T_k within R_k , if $D(n_k) \leq xW'H'$.
3. Otherwise, if T_k has a large enough size, i.e., if $xW'/H' > n_k^{-\epsilon}$, $H_k = H'$ and $W_k = xW'$. In this case, area of R_k is equal to $xH'W'$. T_k is drawn recursively in R_k . Thus, for Algorithm *DrawTree* to successfully draw T_k within R_k , it is necessary that $D(n_k) \leq xW'H'$.

Thus, we get that, for Algorithm *DrawTree* to successfully draw T_k within R_k , it is necessary that $D(n_k) \leq xH'W'$, i.e., $D(xn) \leq x(W - 2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2)(H - 2)$. (Since $n_k = xn$, $H' = H - 2$ and $W' = W - 2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2$.)

The condition that $D(xn) \leq x(W - 2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2)(H - 2)$ is also a sufficient condition for Algorithm *DrawTree* to draw T within R . To see this, notice that 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 vertical channels to combine drawings of the partial trees to obtain a drawing of T . At least one partial tree of T , namely the subtree rooted at v , where (u, v) is the separator of T , has a large enough size, since it contains at least n/d nodes. Thus, T can have at most $2d$ very small and moderately sized partial trees. We can show that if T_k is a moderately large tree, then it has at most $(2n)^{\frac{1}{1+\epsilon}}$ nodes in it, and $W_k \leq (2n)^{\frac{1-\epsilon}{1+\epsilon}}$. Each very-small partial tree is assigned a rectangle with height $H - 2$ and width $\log_2 n + 4$. Hence, if each partial tree T_k can be drawn within its assigned rectangle R_k , then the total area assigned to very small and moderately large partial trees

is at most $2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\}(H - 2)$ area. Thus, the total area within R available for drawing all the large partial trees is at least $W'H' = (W - 2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2)(H - 2)$. Therefore, if each large partial tree with xn nodes can be drawn within area $xW'H'$, i.e., within its assigned rectangle, then the entire drawing of T can fit within R . Each very-small partial tree can be drawn within its assigned rectangle using Algorithm u^* -HV-Draw. If $D(xn) \leq x(W - 2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2)(H - 2)$ then each moderately large and large partial tree can also be drawn within its assigned rectangle. We therefore conclude the if $D(xn) \leq x(W - 2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2)(H - 2)$, then T can be drawn within R .

Thus, $D(xn) \leq x(W - 2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2)(H - 2)$ is a necessary and 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 , $H \leq \sqrt{D(n)}$, and since $A = W/H \leq n^\epsilon$, $W \leq n^{\epsilon/2} \sqrt{D(n)}$. Substituting W and H by these bounds in $D(xn) \leq x(W - 2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} - 2)(H - 2)$ and with some mathematical manipulation, we get that $D(xn) \leq x(D(n) - (2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} + 2n^{\epsilon/2} + 2)\sqrt{D(n)})$ 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 $1/2$. If $2d \cdot \max\{(2n)^{\frac{1-\epsilon}{1+\epsilon}}, \log_2 n + 4\} + 2n^{\epsilon/2} + 2 \leq pn^\beta$, for some constants p and β , where $p > 0$, and $0 \leq \beta < 1/2$, then, by Theorem 1, we get that $D(n) = O(n)$.

We observe that $(2n)^{\frac{1-\epsilon}{1+\epsilon}}$ is maximum (resp. minimum) when ϵ is minimum (resp. maximum). Also, $(2n)^{\frac{1-\epsilon}{1+\epsilon}}$ maximum (resp. minimum) implies d minimum (resp. maximum). Since $1/3 < \epsilon < 1$ can be rewritten as $0 < \frac{1-\epsilon}{1+\epsilon} < \frac{1}{2}$, $d = o(n^{1/2})$. As $d = o(n^{1/2})$ grows, ϵ decreases in the range $(\frac{1}{3}, 1)$. □

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

Proof: Let ϵ be a constant such that $1 \leq A \leq n^\epsilon$ 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 as defined in proof of Lemma 4. Note that from the proof of Lemma 4, $D(n) = O(n)$. Construct a drawing Γ of T in R by calling Algorithm *DrawTree* with T , R and ϵ as input. From Lemmas 2, 3, and 4, 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. A preliminary version appeared in Proceedings of Graph Drawing '96, Lecture Notes in Computer Science, volume 1190, pages 63-75, 1997.
- [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] C. E. Leiserson. Area-efficient graph layouts (for VLSI). In *Proc. 21st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 270-281, 1980.
- [7] E. Reingold and J. Tilford. Tidier drawing of trees. *IEEE Trans. Softw. Eng.*, SE-7(2):223-228, 1981.
- [8] M. Sarkar and M. H. Brown. Graphical fisheye views. *Commun. ACM*, 37(12):73-84, 1994.

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

Appendix

4.2 Algorithm u^* -HV-Draw

The classical HV-drawing algorithm [2] can be used to construct a drawing of a binary tree T with n nodes in a rectangle with dimensions n and $\log_2 n + 2$. Moreover, users can specify whether the longer dimension is the height or the width of a rectangle. This is formally stated in the following lemma:

Lemma 5 ([2]) *Let T be a binary tree with n nodes. Let R be a rectangle with width (height) n and height (width) $\log_2 n + 2$. Then, we can construct a drawing of T within R in $O(n)$ time, such that the root of T is placed at the top-left corner of R .*

This result stands even for degree- d trees:

Lemma 6 *Let T be a degree- d tree with n nodes. We can construct a drawing of T within a rectangle R with dimensions n and $\log_2 n + 2$ in $O(n)$ time, such that the root of T is placed at the top-left corner of R .*

However, it is possible that HV-drawing algorithm may not construct a feasible drawing of T , if T has a link vertex u^* , because it may not place u^* at the bottom of the drawing. Here, we give Algorithm u^* -HV-Draw, which will construct a feasible drawing of T , if T has a link vertex u^* , within a rectangle R with width (height) n and height (width) $\log_2 n + 4$. Let W and H be the width and height, respectively, of R .

1. Order the children of each node of T such that u^* becomes the leftmost node of T . As Described in Section 4.1, in the description of Algorithm *DrawTree*, this can be easily done in $O(\frac{n}{d-1})$ time.
2. Let r be the root of T . Let $p = r_1(=r), r_2, \dots, r_k(=u^*)$ be the leftmost path of T (see Figure 8). Let $c_j^1, \dots, c_j^{i_j}$ be the rest of the children of r_j , $1 \leq j \leq k$, $0 \leq i_j \leq d-1$. (If a vertex r_j has only one child, then $i_j = 0$). Let $T_j^1, \dots, T_j^{i_j}$ be the subtrees rooted at $c_j^1, \dots, c_j^{i_j}$, $1 \leq j \leq k$, $0 \leq i_j \leq d-1$.
3. If $W \geq H$, then draw each T_j^i using the HV-Drawing algorithm of [2] within a rectangle of width $n(T_j^i)$ and height $\log_2 n(T_j^i) + 4$. Otherwise (i.e., if $H > W$), draw T_j^i using the HV-Drawing algorithm of [2] within a rectangle of width $\log_2 n(T_j^i) + 4$ and height $n(T_j^i)$. Let Γ_j^i be the drawing constructed of T_j^i . Note that by Lemma 6, c_j^i will be at the top-left corner of Γ_j^i .
4. We have three cases, based on the value of k :
 - (a) $k = 1$: As shown in Figure 11(b),(c), place $\Gamma_1^1, \dots, \Gamma_1^{i_1}$ in a left-to-right order such that their top boundaries are aligned. Place r one unit to the left and one unit above Γ_1^1 . Draw the edges $(r, c_1^1), \dots, (r, c_1^{i_1})$.
 - (b) $k = 2$: If $W \geq H$, then flip $\Gamma_k^1, \dots, \Gamma_k^{i_k}$ vertically, and place $\Gamma_1^1, \dots, \Gamma_1^{i_1}$ in a left-to-right order such that their top boundaries are aligned, to the left of Γ_k^1 , separated by unit horizontal distance such that their bottom boundaries are aligned (see Figure 10(b)); otherwise (i.e., if $H > W$), stack the drawings of $\Gamma_1^1, \dots, \Gamma_1^{i_1}, \Gamma_k^1, \dots, \Gamma_k^{i_k}$ one above the other, separated by unit vertical distance such that their left boundaries are aligned (see Figure 10(c)). Place r one unit to the left and one unit above Γ_1^1 , and u^* one unit to the left and one unit below $\Gamma_k^{i_k}$. Draw the edges $(r, c_1^1), \dots, (r, c_1^{i_1}), (r, u^*)$, and $(u^*, c_k^1), \dots, (u^*, c_k^{i_k})$.
 - (c) $k \geq 3$: If $W \geq H$, then first flip $\Gamma_k^1, \dots, \Gamma_k^{i_k}$ vertically, and then, as shown in Figure 9(a), place $\Gamma_1^1, \dots, \Gamma_1^{i_1}, \dots, \Gamma_{k-1}^1, \dots, \Gamma_{k-1}^{i_{k-1}}, \Gamma_k^1, \dots, \Gamma_k^{i_k}$ in a left-to-right order such that their top boundaries are aligned, $\Gamma_1^1, \dots, \Gamma_1^{i_1}, \dots, \Gamma_{k-1}^1, \dots, \Gamma_{k-1}^{i_{k-1}}$ are separated by unit horizontal distance, and $\Gamma_{k-1}^{i_{k-1}}$ and Γ_k^1 are separated by two units of horizontal distance. Place $r, r_2, \dots, r_{k-1}$ at one unit above these drawings at unit vertical distance, such that $r, r_2, \dots, r_{k-2}$ are aligned with the left boundaries of $\Gamma_1^1, \Gamma_2^1, \dots, \Gamma_{k-2}^1$, respectively, and r_{k-1} is aligned with the right boundary of $\Gamma_{k-1}^{i_{k-1}}$. Place u^* between $\Gamma_{k-1}^{i_{k-1}}$ and Γ_k^1 such that it is separated by unit horizontal distance from both, and is placed at unit vertical distance below the bottom boundary of Γ_j , where Γ_j is the subtree among $\Gamma_j^1, \Gamma_j^2, \dots, \Gamma_j^{i_j}$, $1 \leq j \leq d-1$, $0 \leq i_j \leq k$ with greatest height. If $H > W$, then as shown in Figure 9(b), stack the drawings of $\Gamma_1^1, \dots, \Gamma_1^{i_1}, \dots, \Gamma_{k-1}^1, \dots, \Gamma_{k-1}^{i_{k-1}}, \Gamma_k^1, \dots, \Gamma_k^{i_k}$ one above the other such that their left boundaries are aligned. Place each r_j , where $1 \leq j \leq k-1$ at the same height as c_j^1 at unit horizontal distance from it towards its left. Place u^* to the left of $c_k^{i_k}$ at unit

horizontal distance and at unit vertical distance below the bottom boundary of $\Gamma_k^{i_k}$.

Lemma 7 *Let T be a n -node degree- d 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 $\log_2 n + 4$, as well as within a rectangle with width $\log_2 n + 4$ and height n .*

Proof: Immediate from Lemma 6, and the way Algorithm u^* -HV-Draw constructs its drawings. $\square$

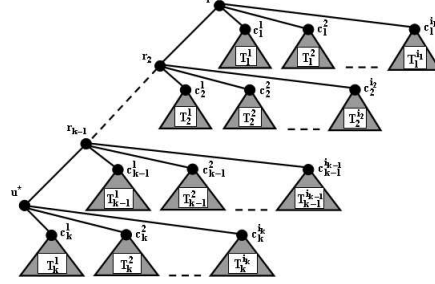


Figure 8: General form of a degree- d tree T with link vertex u^* and root r , where u^* is also the leftmost vertex of T .

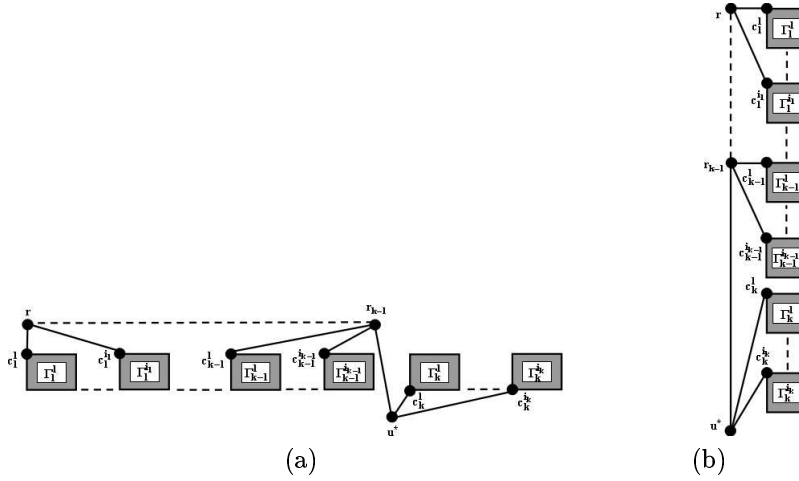


Figure 9: Drawing T of the Figure 8 when $k \geq 3$: (a) When $W \geq H$ (b) when $H > W$.

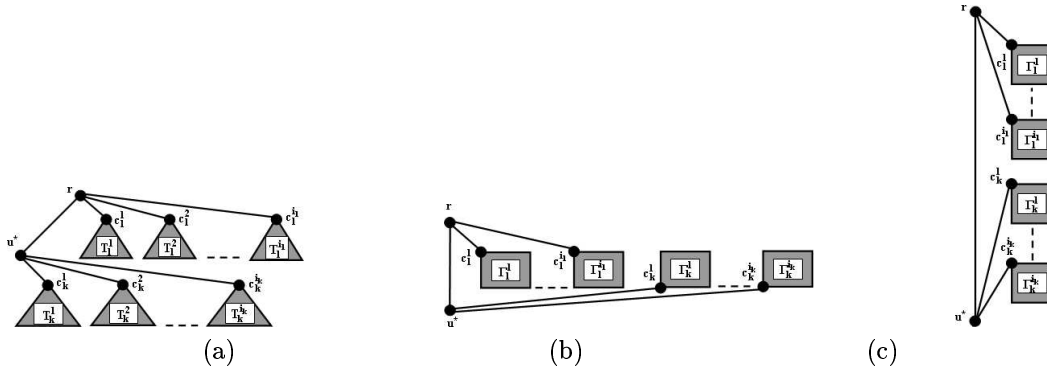


Figure 10: Drawing T of the Figure 8 when $k = 2$: (a) Tree T in this case; (b) Drawing T when $W \geq H$; (c) Drawing T when $H > W$.

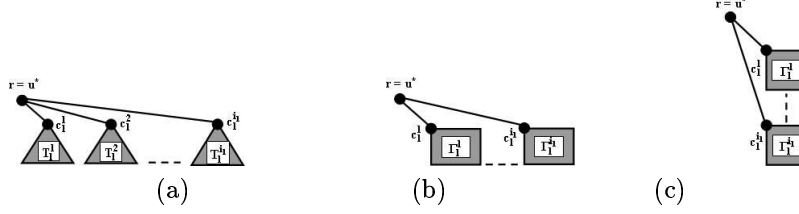


Figure 11: Drawing T of the Figure 8 when $k = 1$: (a) Tree T in this case; (b) Drawing T when $W \geq H$; (c) Drawing T when $H > W$.

4.3 Algorithm *Comp*

We now give the pseudocode for Algorithm *Comp*. We use the same notation as in Section 4.1. Let

$$x = \begin{cases} f & \text{if } T_\beta \neq \emptyset \\ y & \text{otherwise} \end{cases}$$

Let D and D' be two feasible drawings of some trees H and H' . The operation $D \ominus_1 D'$ places D to the left of D' separated from it by unit horizontal distance. The operation $D \ominus_2 D'$ places D to the left of D' separated from it by two units of horizontal distance. The operation $D \oplus_1 D'$ places D above D' separated from it by unit vertical distance. The operation $D \oplus_2 D'$ places D above D' separated from it by two units of vertical distance.

If the separator edge (u, v) does not belong to the leftmost path of T then algorithm *Comp* constructs Γ as illustrated in Figure 6, otherwise *Comp* constructs Γ as illustrated in Figure 7.

Algorithm *Comp*

Input: $\Gamma_A, \Gamma_B, \Gamma_C, \Gamma^1, \dots, \Gamma^i, 0 \leq i \leq d-2, R$ or $\Gamma_A, \Gamma_1, \dots, \Gamma_j, 0 \leq j \leq d-1, \Gamma_C, R$.

Output: A feasible drawing Γ of T .

Let W and H be the width and height, respectively, of R ;

Let (i, j) be the coordinates of the top-left corner of R ;

if (u, v) does not belong to the leftmost path of T (see Figure 6) **then** {

if $T_A \neq \emptyset$ and $T_C \neq \emptyset$ (see Figure 6(a),(b)) **then** {

if $g = u^*$ **then** Flip Γ_C vertically;

if $W \leq H$ **then** Let $\Gamma' = \Gamma_A \oplus_2 (\Gamma_i \oplus_1 \dots \oplus_1 \Gamma_B \oplus_1 \dots \oplus_1 \Gamma_1 \oplus_1 \Gamma_C)$ such that the left boundaries of $\Gamma_1, \dots, \Gamma_i, \Gamma_B$, and Γ_C are aligned and are placed one unit to the right of the left boundary of Γ_A ;

else ($H < W$) {

 Rotate Γ_A by 90° ;

 Flip Γ_A vertically;

 Let $\Gamma' = \Gamma_A \ominus_2 (\Gamma_C \ominus_1 \Gamma^1 \ominus_1 \dots \ominus_1 \Gamma_B \ominus_1 \dots \ominus_1 \Gamma^i)$ such that the top boundaries of $\Gamma^1, \dots, \Gamma^i, \Gamma_B$, and Γ_C are aligned and are placed one unit below the top boundary of Γ_A ;

 Move Γ_C down such that u^* becomes the bottommost vertex of Γ' ;

 }

 Place Γ' such that r is at grid location (i, j) ;

if $W \leq H$ **then** Place a at grid location $(i, j - (\text{height}(R(T_A)) + 1))$;

else ($H < W$) Place a at grid location $(i + \text{width}(R(T_A)) + 1, j)$;

 Connect a with e, x, g , and $b_1, \dots, b_i$;

 }

else if $T_A \neq \emptyset, T_C = \emptyset$ and $r \neq e$ (see Figure 6(c)) **then** {

```

if  $W \leq H$  then {
    Rotate  $\Gamma_A$  by  $90^\circ$ ;
    Flip  $\Gamma_A$  vertically;
    Flip  $\Gamma^1, \dots, \Gamma^i$ , and  $\Gamma_B$  horizontally;
    Then Flip  $\Gamma^1, \dots, \Gamma^i$ , and  $\Gamma_B$  vertically;
    Let  $\Gamma' = \Gamma_A \oplus_1 \Gamma^1 \oplus_1 \dots \oplus_1 \Gamma_B \oplus_1 \dots \oplus_1 \Gamma^1$  such that the left boundaries of  $\Gamma_A, \Gamma^1, \dots, \Gamma^i$ 
    and  $\Gamma_B$  are aligned;
    Stretch  $\Gamma_A$  to the right such that  $e$  becomes the rightmost vertex of  $\Gamma'$ ;
    Place  $\Gamma'$  such that  $r$  is at grid location  $(i, j)$ ;
    Place  $u^*$  at the bottom-right corner of  $R(T)$ ;
}

else (  $H < W$  ) {
    Rotate  $\Gamma_A$  by  $90^\circ$ ; Flip  $\Gamma_A$  vertically;
    Flip  $\Gamma^1, \dots, \Gamma^i$  and  $\Gamma_B$  vertically;
    Let  $\Gamma' = \Gamma_A \ominus_2 (\Gamma^1 \ominus_1 \dots \ominus_1 \Gamma_B \ominus_1 \dots \ominus_1 \Gamma^i)$ , such that the top boundaries of  $\Gamma_A, \Gamma_B$ , and
     $\Gamma^1, \dots, \Gamma^i$  are aligned;
    Place  $\Gamma'$  such that  $r$  is at grid location  $(i, j)$ ;
    Place  $u^*$  at grid location  $(i + \text{width}(R(T_A)) + 1, j - \text{height}(R))$ ;
}

Connect  $u^*$  with  $e, x$ , and  $b_1, \dots, b_i$ ;
}

else if  $T_A \neq \emptyset, T_C = \emptyset$  and  $r = e$  (see Figure 6(d)) then {
    if  $W \leq H$  then {
        Flip  $\Gamma^1, \dots, \Gamma^i$  and  $\Gamma_B$  vertically;
        Let  $\Gamma' = \Gamma_A \oplus_1 (\Gamma^i \oplus_1 \dots \oplus_1 \Gamma_B \oplus_1 \dots \oplus_1 \Gamma^1)$  such that the left boundaries of  $\Gamma^1, \dots, \Gamma^i$ 
        and  $\Gamma_B$  are aligned and are placed one unit to the right of the left boundary of  $\Gamma_A$ ;
    }

    else (  $H < W$  ) {
        Flip  $\Gamma^1, \dots, \Gamma^i$  and  $\Gamma_B$  vertically;
        Let  $\Gamma' = \Gamma_A \ominus_1 (\Gamma^1 \ominus_1 \dots \ominus_1 \Gamma_B \ominus_1 \dots \ominus_1 \Gamma^i)$  such that the bottom boundaries of  $\Gamma_A, \Gamma_B$ ,
        and  $\Gamma^1, \dots, \Gamma^i$ , are aligned;
    }

    Place  $\Gamma'$  such that  $r$  is at grid location  $(i, j)$ ;
    Place  $u^*$  at the bottom-left corner of  $R(T)$ ;
    Connect  $u^*$  with  $r, x$ , and  $b_1, \dots, b_i$ ;
}

else if  $T_A = \emptyset, T_C \neq \emptyset$ , and  $g = u^*$  (see Figure 6(e)) then {
    Place  $r$  at grid location  $(i, j)$ ;
    if  $W \leq H$  then {
        Flip  $\Gamma_C$  vertically;
        Let  $\Gamma' = \Gamma^i \oplus_1 \dots \oplus_1 \Gamma_B \oplus_1 \dots \oplus_1 \Gamma^1 \oplus_1 \Gamma_C$  such that the left boundaries of  $\Gamma^1, \dots, \Gamma^i, \Gamma_B$ ,
        and  $\Gamma_C$  are aligned;
        Place  $\Gamma'$  such that  $b_i$  is at grid location  $(i + 1, j - 1)$ ;
    }

    else (  $H < W$  ) {
        Flip  $\Gamma_C$  vertically;
    }
}

```

Let $\Gamma' = \Gamma_C \ominus_1 \Gamma^1 \ominus_1 \dots \ominus_1 \Gamma_B \ominus_1 \dots \ominus_1 \Gamma^i$ such that the top boundaries of $\Gamma^1, \dots, \Gamma^i, \Gamma_B$, and Γ_C are aligned;
 Place Γ' such that b_1 is at grid location $(i+1, j-1)$;
 Move Γ_C down such that u^* becomes the bottommost vertex of Γ' ;
 }
 Connect r with $b_1, \dots, b_i, x$, and u^* ;
 }
else if $T_A = \emptyset, T_C \neq \emptyset$, and $g \neq u^*$ (see Figure 6(f)) **then** {
 Place r at grid location (i, j) ;
if $W \leq H$ **then**
 Let $\Gamma' = \Gamma^i \oplus_1 \dots \oplus_1 \Gamma_B \oplus_1 \dots \oplus_1 \Gamma^1 \oplus_1 \Gamma_C$ such that the left boundaries of $\Gamma^1, \dots, \Gamma^i, \Gamma_B$, and Γ_C are aligned;
 Place Γ' such that b_i is at grid location $(i+1, j-1)$;
 }
else ($H < W$)
 Let $\Gamma' = \Gamma_C \ominus_1 \Gamma^1 \ominus_1 \dots \ominus_1 \Gamma_B \ominus_1 \dots \ominus_1 \Gamma^i$ such that the top boundaries of $\Gamma^1, \dots, \Gamma^i, \Gamma_B$, and Γ_C are aligned;
 Place Γ' such that g is at grid location $(i+1, j-1)$;
 Move Γ_C down such that u^* becomes the bottommost vertex of Γ' ;
 }
 Connect r with x, g , and $b_1, \dots, b_i$;
 }
else ($T_A = \emptyset, T_C = \emptyset$ (see Figure 6(g))) {
 Place r at grid location (i, j) ;
if $W \leq H$ **then** {
 Let $\Gamma' = \Gamma^i \oplus_1 \dots \oplus_1 \Gamma_B \oplus_1 \dots \oplus_1 \Gamma^1$ such that the left boundaries of $\Gamma^1, \dots, \Gamma^i$, and Γ_B are aligned;
 Place Γ' such that b_i is at the grid location $(i+1, j-1)$;
 }
else ($H < W$) {
 Let $\Gamma' = \Gamma^1 \ominus_1 \dots \ominus_1 \Gamma_B \ominus_1 \dots \ominus_1 \Gamma^i$ such that the top boundaries of $\Gamma^1, \dots, \Gamma_B, \dots, \Gamma^i$ are aligned;
 Place Γ' such that b_1 is at the grid location $(i+1, j-1)$;
 }
 Connect r with x and $b_1, \dots, b_i$;
 }
 }
else ((u, v) belongs to the leftmost path of T (see Figure 7)) {
if $T_A \neq \emptyset, F_B \neq \emptyset$ (see Figure 7)(a) **then** {
if $W \leq H$ **then** Let $\Gamma' = \Gamma_A \oplus_2 (\Gamma_j \oplus_1 \dots \oplus_1 \Gamma_1 \oplus_1 \Gamma_C)$ such that the left boundaries of $\Gamma_1, \dots, \Gamma_j$, and Γ_C are aligned and are placed one unit to the right of the left boundary of Γ_A ;
else ($H < W$) {
 Rotate Γ_A by 90° ; Flip Γ_A vertically;
 Let $\Gamma' = \Gamma_A \oplus_2 (\Gamma_C \ominus_1 \Gamma_1 \ominus_1 \dots \ominus_1 \Gamma_j)$ such that the top boundaries of Γ_C and $\Gamma_1, \dots, \Gamma_j$ are aligned and are placed one unit below the top boundary of Γ_A ;
 Move Γ_C down such that u^* becomes the bottommost vertex of Γ' ;
 }
 }
 }
 }

```

    Place  $\Gamma'$  such that  $r$  is at grid location  $(i, j)$ ;
    if  $W \leq H$  then Place  $u$  at grid location  $(i, j - (\text{height}(R(T_A)) + 1))$ ;
    else (  $H < W$  ) Place  $u$  at grid location  $(i + \text{width}(R(T_A)) + 1, j)$ ;
    Connect  $u$  with  $e, c_1, \dots, c_j$ , and  $v$ ;
}
else if  $T_A = \emptyset, F_B \neq \emptyset$  (see Figure 7(b)) then {
    Place  $r$  at grid location  $(i, j)$ ;
    if  $W \leq H$  then {
        Let  $\Gamma' = \Gamma_j \oplus_1 \dots \oplus_1 \Gamma_1 \oplus_1 \Gamma_C$  such that the left boundaries of  $\Gamma_1, \dots, \Gamma_j$  and  $\Gamma_C$  are aligned;
        Place  $\Gamma'$  such that  $c_j$  is at grid location  $(i + 1, j - 1)$ ;
    }
    else (  $H < W$  ) {
        Let  $\Gamma' = \Gamma_C \ominus_1 \Gamma_1 \ominus_1 \dots \ominus_1 \Gamma_j$  such that the top boundaries of  $\Gamma_1, \dots, \Gamma_j$  and  $\Gamma_C$  are aligned;
        Place  $\Gamma'$  such that  $v$  is at grid location  $(i + 1, j - 1)$ ;
        Move  $\Gamma_C$  down such that  $u^*$  becomes the bottommost vertex of  $\Gamma'$ ;
    }
    Connect  $r$  with  $c_1, \dots, c_j$  and  $v$ ;
}
else if  $T_A \neq \emptyset, F_B = \emptyset$  (see Figure 7(c)) then {
    if  $W \leq H$  then {
        Let  $\Gamma' = \Gamma_A \oplus_2 \Gamma_C$  such that the left boundary of  $\Gamma_C$  is one unit to the right of the left
        boundary of  $\Gamma_A$ ;
        Place  $\Gamma'$  such that  $r$  is at grid location  $(i, j)$ ;
        Place  $u$  at grid location  $(i, j - (\text{height}(R(T_A)) + 1))$ ;
    }
    else (  $H < W$  ) {
        Rotate  $\Gamma_A$  by  $90^\circ$ ;
        Flip  $\Gamma_A$  vertically;
        Let  $\Gamma' = \Gamma_A \ominus_2 \Gamma_C$  such that the top boundary of  $\Gamma_C$  is one unit below the top boundary of
         $\Gamma_A$ ;
        Place  $\Gamma'$  such that  $r$  is at grid location  $(i, j)$ ;
        Place  $u$  at grid location  $(i + \text{width}(T_A) + 1, j)$ ;
        Move  $\Gamma_C$  down such that  $u^*$  becomes the bottommost vertex of  $\Gamma'$ ;
    }
    Connect  $u$  with  $e$  and  $v$ ;
}
else (  $T_A = \emptyset, F_B = \emptyset$  ) (see Figure 7(d)) {
    Place  $r$  at grid location  $(i, j)$ ;
    Place  $\Gamma_C$  such that  $v$  is at grid location  $(i + 1, j - 1)$ ;
    Connect  $r$  with  $v$ ;
}
}
Let  $\Gamma$  be the drawing obtained;
Return  $\Gamma$ ;

end Algorithm.

```