Upward Octagonal Drawings of Ternary Trees

by

Seunghee Lee

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2016

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We explore ways to embed a ternary tree in an integer coordinate grid such that the width of the drawing is minimized. We provide upper and lower bounds on the width requirement of planar, straight-line, upward, order-preserving drawings of ternary trees in an octagonal grid. We present a linear-time algorithm for constructing such octagonal grid drawings of any n-node ternary tree with $O(n^{0.68})$ width. (This bound can be improved to $O(n^{0.631})$ width in the so-called HVA-model.) For ideal octagonal grid drawings of complete n-node ternary trees, we provide an $\Omega(n^{0.411})$ width lower bound.

Acknowledgements

I would like to thank my supervisors Therese Biedl and Timothy Chan for teaching me and inspiring me topics in graph drawing. I am grateful for your patience and guidance throughout my master's.

I would like to thank my committee members Naomi Nishimura and Alex Lopez-Ortiz for taking the time to read and offer feedback.

Finally, I would like to thank my family and friends in Waterloo for their love and support.

Dedication

To my parents: without whom none of my success will be possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Graph drawing

Graphs (made up of nodes and edges such as the Facebook social network graph) are frequently found in our daily lives and help us understand better the relationship among interconnected objects. Using a graph drawing algorithm, we can plot the graph by embedding the nodes in an integer coordinate grid, thereby producing a pictorial representation. Graph visualization allows us to see complicated relationships within a system such as clusters in social networks, disease spreading in biology, and semantic hierarchy in linguistics. Moreover, the analysis and design of graph drawing algorithms offers combinatorial and geometric open problems interesting in their own right.

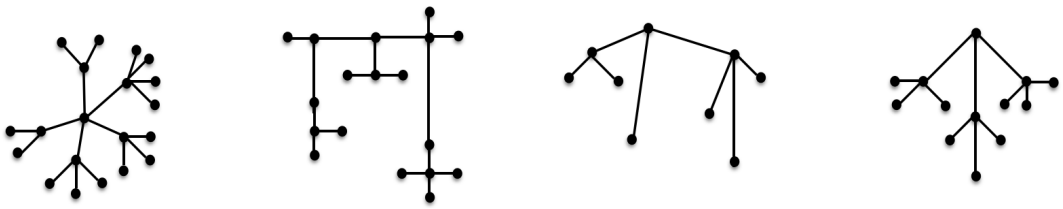


Figure 1.1: Various types of tree drawings are shown: (a) radial drawing, (b) orthogonal drawing, (c) ideal drawing, (d) octagonal drawing.

Among the different types of graphs, (rooted/directed) trees are most suited for representing hierarchical or acyclic relationships. The parent/child nodes associated with each edge and the designated root node provide some orientation and order among the nodes in the tree. Drawing trees is an extensively studied area in graph drawing first initiated by D. E. Knuth [11]. The study of area minimization for drawing binary trees began with a hierarchical style, using a typewriter as a drawing tool, and later inspired the Reingold-Tilford algorithm [12]. Subsequently, different styles of tree drawings emerged (see Figure 1.1) including radial drawings for applications in social sciences, and orthogonal drawings motivated by VLSI circuits. We discuss below the styles considered in this thesis and then give a survey of existing results.

1.2 Ideal drawings

In graph drawing, we first select a set of aesthetically pleasing properties we wish to enforce in the drawing and then select a cost function such as area, width, or aspect ratio (of the bounding box of the resulting drawing) that we wish to minimize. The criteria for an *ideal* drawing vary depending on the type of graph and its application and is left to the author to define. In this thesis, we select the following properties to be necessary for an ideal drawing (these properties are commonly used in tree drawing literature [6]). Throughout the thesis, we identify a vertex/edge with the point/straight-line segment that represents it.

1. **grid drawing**
   All vertices are drawn as points with integer coordinates.

2. **straight-line**
   An edge between two vertices is drawn as a straight-line segment between the points representing the vertices.

3. **planar**
   No two edges cross each other.

4. **upward**
   The parent vertex has same or larger y-coordinate than its child vertex.

5. **order-preserving**
   The line segments from a node to all of its children are sorted by slope from left to right.

The above are required properties in our drawing. It is also possible to enforce a stronger restriction on the upward/order-preserving properties of drawings. Since we discuss some results in strictly upward and/or strictly order-preserving drawings in the background section, we give the definitions here.

1. **strictly upward**
   The parent vertex has strictly larger y-coordinate than its child vertex.

2. **strictly order-preserving**
   The line segment from a node to its leftmost/rightmost child is monotone decreasing/increasing in the x-direction. The line segments from a node to all of its children are sorted by slope from left to right.

Our objective is to minimize the width of the drawing. We measure height, width, and area of a drawing by the number of columns, rows, and grid points in its bounding box (i.e., smallest axes-parallel rectangle). Since it is not possible to produce upward ternary tree drawings with $o(n)$ height in general (consider a chain of nodes where each node in the chain has three children), we choose to minimize the width.

1.3 Drawing models

In this section, we present two drawing models: *octagonal* and *HVA*. The octagonal drawing model is particularly attractive since the restriction on angle allows the output to be displayed as an ASCII file (using the three characters —, /, and \ for the edges). Our main results are in the octagonal drawing model; the HVA drawing model does not have a restriction on angle and serves as a warm up to the construction of the width upper bound for the octagonal drawing model.

Definition 1. *A drawing of a rooted tree is an* octagonal drawing *(see Figure 1.2) if the angle between every pair of edges incident to the same node is a multiple of 45 degrees. In particular, we may assume that every edge is either vertical, horizontal, or on one of the two 45 degree diagonals.*

Recently, Batzill and Biedl [3] proposed the so-called HVA model for tree drawing. They used this to achieve (non-upward) tree drawings of small area. We will use this model as well, because our construction is easier to explain for HVA drawings than for octagonal drawings.

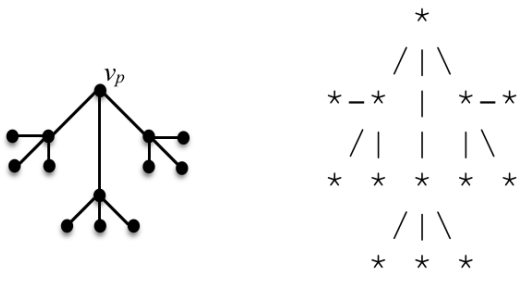


Figure 1.2: (a) An octagonal drawing of a complete ternary tree where the root node is labeled v_p. (b) An octagonal drawing of a complete ternary tree using ASCII characters.

Definition 2. *A drawing of a rooted tree is an* HVA drawing *(see Figure 1.3) if for all edges in the rooted tree, the child vertex is one of the following:*

1. ***horizontal**:*
   The child vertex is drawn with the same y-coordinate as its parent vertex.
2. ***vertical**:*
   The child vertex is drawn with the same x-coordinate as its parent vertex.
3. ***adjacent**:*
   The child vertex is drawn with an y-coordinate one less than its parent vertex (i.e., drawn one row below its parent vertex).

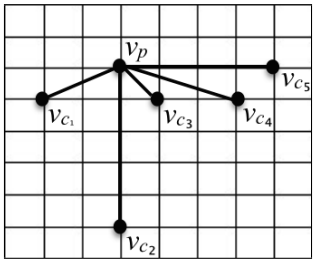


Figure 1.3: An example HVA drawing where the parent vertex v_p has five children $v_{c_1}, \ldots, v_{c_5}$. Vertex v_{c_5} is horizontal, v_{c_2} is vertical, and $v_{c_1}, v_{c_3}, v_{c_4}$ are adjacent.

1.4 Background

In the literature, the area requirements for planar straight-line drawings of ternary trees varies depending on the restriction on the angle between edges and whether the drawing is upward and/or order-preserving. For tree drawings with arbitrary angles, Chan [6] has shown an upper bound on the area requirement of a planar straight-line upward order-preserving grid drawing of an n-node k-ary tree, namely $O(n4^{\sqrt{2\log n}})$, which was later improved by Biedl to $O(kn\log n)$ [4]. Relaxing the upward requirement, Garg and Rusu have shown that any n-node k-ary tree admits a planar straight-line non-upward order-preserving grid drawing with area $O(n\log n)$ [10].

Readability and aesthetics of tree drawings improve with restrictions on angle and this has sparked interest in orthogonal, hexagonal, and octagonal grids drawings (where the angle between any two edges connected to the same parent node is a multiple of 90, 60, and 45 degrees respectively). Straight-line orthogonal drawings of binary trees are well-studied [7, 8, 9]. For straight-line octagonal drawings of binary trees, Chan has shown $O(n^{1.48})$ area for upward order-preserving drawings [6]. For straight-line orthogonal drawings of ternary trees, the upward requirement cannot be met since all four directions on the orthogonal grid are required for complete ternary trees [8]. Frati has shown a tight area bound of $\Theta(n^2)$ for straight-line orthogonal non-upward order-preserving drawings for n-node ternary trees [8]. For straight-line orthogonal non-upward non-order-preserving drawings, Frati has shown an upper bound of $O(n^{1.262})$ and $O(n^{1.631})$ area for complete and arbitrary n-node ternary trees [8].

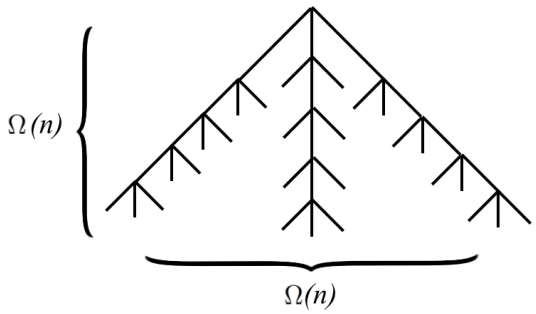


Figure 1.4: An example of an n-node ternary tree composed of three chains which results in $\Omega(n^2)$ area in straight-line hexagonal upward order-preserving drawings.

	Angle	Upward	Area Bound	Ref.
Complete Ternary	orthogonal		$O(n^{2\log_3 2}) = O(n^{1.262})$	[8]
Complete Ternary	hexagonal	✓	$\Theta(n^{2\log_3 2}) = O(n^{1.262})$	[1]
Complete Ternary	octagonal		$O(n^{(\log_3 100)/4}) = O(n^{1.048})$	[2]
Complete Ternary	octagonal	✓	$\Omega(n^{0.411})$ width	Thm. 6

Table 1.1: Summary of area bounds for planar straight-line drawings of complete ternary trees.

	Angle	Upward	Order-preserving	Width Bound	Ref.
Ternary	any	✓(strictly)	✓	$O(\log n)$	[4]
Ternary	any (HVA)		✓	$O(\log n)$	[3]
Ternary	orthogonal			$O(n^{\log_3 2}) = O(n^{0.631})$	[8]
Ternary	orthogonal		✓	$\Theta(n)$	[8]
Ternary	hexagonal	✓		$\Theta(n^{\log_3 2}) = O(n^{0.631})$	[1]
Ternary	any (HVA)	✓(strictly)	✓(strongly)	$O(n^{\log_3 2}) = O(n^{0.631})$	Thm. 1
Ternary	octagonal	✓	✓(strongly)	$O(n^{0.68})$	Thm. 3

Table 1.2: Summary of width bounds for planar straight-line drawings of arbitrary ternary trees.

As we wish to avoid arbitrary angles, the natural next step is hexagonal or octagonal grid drawings. Bachmaier et al. [1] have shown that a ternary tree admits a planar straight-line upward non-order-preserving hexagonal grid drawing with area $O(n^{1.262})$ and $O(n^{1.631})$ for complete and arbitrary n-node ternary trees. It can be seen in Figure 1.4 that enforcing upward and order-preserving properties for hexagonal drawings of ternary trees require $\Omega(n^2)$ area. In the octagonal case, Bachmaier and Matzner showed that an n-node complete ternary tree admits a planar straight-line non-upward order-preserving octagonal grid drawing in $O(n^{1.048})$ area with aspect ratio 1 [2]. There are no previous results on ideal octagonal drawings of arbitrary ternary trees.

Table 1.1 compares best known area bounds for complete ternary trees with results presented in this thesis. While our lower bound result is in the width of a complete ternary tree (as opposed to the area), we can easily extend Theorem 6 to obtain a lower bound on the area of some ternary tree using a simple construction (see Figure 1.5). In a similar way, it is always possible to enforce $\Omega(n)$ height for arbitrary trees; therefore we focus on results on bounding the width for arbitrary ternary trees. Table 1.2 compares best known width bounds with results presented in this thesis. We present worst-case asymptotic upper and lower bounds on width as a function of n.

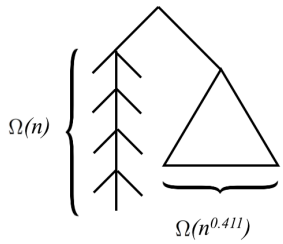


Figure 1.5: A ternary tree that requires $\Omega(n^{1.411})$ area where a single chain and a complete ternary tree are attached to the root node.

1.5 Our results

We present the first sublinear upper bound and the first nontrivial lower bound on the width of ideal octagonal drawings of arbitrary ternary trees. Our results are organized as follows:

- Section 2.1: Any ternary tree admits a planar straight-line strictly upward order-preserving HVA drawing of width $O(n^{\log_3 2}) = O(n^{0.631})$.
- Section 2.2: Any ternary tree admits a planar straight-line upward order-preserving octagonal grid drawing of width $O(n^{0.68})$.
- Chapter 3: For complete ternary trees, the width of any planar straight-line upward order-preserving octagonal grid drawing is $\Omega(n^{0.411})$.

The techniques used in this thesis are a mix of new approaches as well as modifications of existing methods in the literature. For example, in the upper bound (Chapter 2), we use a recursive algorithm to draw an HVA/octagonal grid drawing which is inspired from a previous paper by Biedl [4]. We modify the existing algorithm and introduce vertically stretchable drawings which allows us to take advantage of their small width. In the lower bound (Chapter 3), we use a counting argument to argue the minimum width of a ternary tree drawing. Unlike many of the previous results, the exponent that arises in this width upper bound $O(n^{0.68})$ and width lower bound $\Omega(n^{0.411})$ are determined numerically.

Chapter 2

Width Upper Bound

What is the best way to draw ternary trees with minimum width? In the case of complete ternary trees, we can achieve $O(n^{\log_3 2}) \approx O(n^{0.631})$ width quite easily for ideal octagonal drawings: Let T be the given complete ternary tree with root v. Suppose we have already constructed ideal octagonal drawings of the left subtree L, middle subtree M, and right subtree R. We can combine the three drawings into an ideal octagonal drawing of T by vertically aligning v with the root of M, placing the bounding box of L immediately to the left of v using a diagonal edge, placing the bounding box of R immediately to the right of v using a diagonal edge, and placing the bounding box of M immediately below that of L and R (see Figure 2.1).

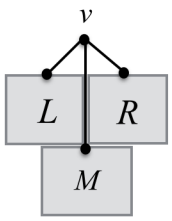


Figure 2.1: We can easily achieve $O(n^{0.631})$ width for ideal octagonal drawings of complete ternary trees.

Then the width satisfies the recurrence

$$W(n) = 2W\left(\frac{n}{3}\right) + 1$$

which can be solved as $W(n) \in O(n^{\log_3 2}) \subseteq O(n^{0.631})$. In Section 2.1, we show that achieving $O(n^{\log_3 2})$ width for unbalanced ternary trees is trickier albeit possible for HVA drawings. In Section 2.2, we study octagonal drawings of unbalanced ternary trees; here we could not achieve width $O(n^{\log_3 2})$ but came close. We present two versions of the so-called Overhang Algorithm: the first version constructs an ideal HVA drawing for any n-node ternary tree with $O(n^{\log_3 2})$ width and the second version constructs an ideal octagonal drawing for any n-node ternary tree with $O(n^{0.68})$ width.

The Overhang Algorithm uses the divide-and-conquer paradigm; in each recursive step, the algorithm breaks the given ternary tree T into several subtrees, draws each subtree recursively using either the Overhang Algorithm or the L-Algorithm (to be defined), and then combines their drawings to obtain a final drawing *overhang*(T) of T. The L-Algorithm is a tree drawing algorithm which produces a so-called L-drawing (defined formally below) of a given ternary tree.

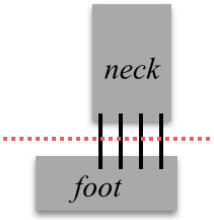


Figure 2.2: An example of an L-drawing. The bounding box of the neck, the bounding box of the foot, the vertical edges connecting the neck and foot, and horizontal line separating the neck and the foot are shown. Since all edges connecting the neck and foot are vertical, an L-drawing is vertically stretchable while preserving width.

Definition 3. *A tree drawing is an L-drawing if there exists a horizontal line that intersects the drawing such that the horizontal line only crosses vertically drawn edges of the tree drawing. The part above this line is called the* neck *and the part below the line is called the* foot *of this drawing.*

Note that the bounding box of the neck sits strictly above the bounding box of the foot (i.e. all y-coordinates are larger) (see Figure 2.2). L-drawings are desirable since we can draw the foot an arbitrary vertical distance below the neck without changing the width of the drawing.

Due to the recursive and asymmetric structure of the Overhang Algorithm, the user can specify the Overhang Algorithm to produce either a left-corner drawing *overhang-left*(T)

of T, or a right-corner drawing *overhang-right*(T) of T. Here a *left-corner drawing* is one in which the root of the tree is drawn on the top-left corner of the bounding box of the drawing. *Right-corner drawing* is symmetrically defined.

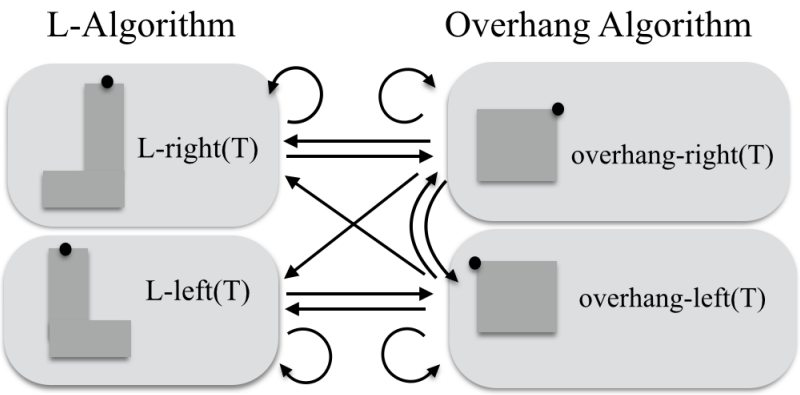


Figure 2.3: For a chosen drawing model (either HVA or octagonal), the Overhang Algorithm can either produce a right-corner overhang drawing or a left-corner overhang drawing. The L-Algorithm can either produce an L-right drawing or L-left drawing. Both algorithms recursively call each other as a subroutine. The arrow $A \to B$ means algorithm which draws A uses algorithm which draws B as a subroutine.

As with the Overhang algorithm, the L-Algorithm uses a divide-and-conquer paradigm where in each recursive step, subtrees are drawn using either the Overhang Algorithm or the L-Algorithm, then combined to obtain the final drawing. The user can specify the L-Algorithm to produce either an L-right-drawing or an L-left-drawing and the two are symmetrically equivalent. Here, an *L-right drawing* is an L-drawing where no node in the foot of the drawing has an x-coordinate larger than any of the nodes in the neck of the drawing and an *L-left drawing* is an L-drawing where no node in the foot of the drawing has an x-coordinate smaller than any of the nodes in the neck of the drawing. For L-drawings, the root does not have to be placed anywhere specific as long as it is placed in the top row. See Figure 2.3.

Below we give common definitions used in this chapter then we describe the Overhang Algorithm and L-Algorithm for HVA drawings.

Definition 4. *For any node v that is not a leaf node, the* heavy child *of v is the child of v with the largest subtree size, breaking ties arbitrarily.*

Definition 5. *The* heavy path *of the ternary tree T is the root-to-leaf path obtained by starting at the root and always taking the heavy child until we reach a leaf.*

2.1 Overhang Algorithm for HVA Drawings

In this section, the Overhang Algorithm and L-Algorithm refer to the version that produces an HVA drawing (as opposed to the version that produces an octagonal drawing).

The L-Algorithm

We will give here only the description of the L-right algorithm, i.e, the version of the L-Algorithm which produces an L-right-drawing; the other algorithm is symmetric.

Let T be the given ternary tree with root vertex v, left subtree T_ℓ, middle subtree T_m, and right subtree T_r. The L-right algorithm uses the *left rule*, *middle rule*, and *right rule* to combine ideal drawings of T_ℓ, T_m, and T_r into an ideal drawing of T. (See Figure 2.4.)

First, we describe which rule to apply at the current root, and which algorithm to use to recursively draw the three subtrees. Let L-*right*(T) denote the drawing of ternary tree T obtained by using the L-right algorithm. Let *overhang-right*(T) denote the right-corner drawing of T obtained by using the Overhang Algorithm.

- If $|T| \leq 1$, return the trivial drawing. Otherwise, compare the sizes of subtrees T_ℓ, T_m, and T_r.
- If the heavy child is the root of T_ℓ, recursively compute L-*right*(T_m), L-*right*(T_r), and *overhang-right*(T_ℓ). Then combine the drawings using the left rule (explained below).
- If the heavy child is the root of T_m, recursively compute L-*right*(T_ℓ), L-*right*(T_r), and *overhang-right*(T_m). Then combine the drawings using the middle rule.
- If the heavy child is the root of T_r, recursively compute L-*right*(T_ℓ), L-*right*(T_m), and *overhang-right*(T_r). Then combine the drawings using the right rule.

Now we describe how to combine the three drawings for the left rule, middle rule, and right rule. For a subtree T, let $D(T)$ denote the drawing of subtree T. (When we describe where to place the y-position of the neck drawing, we refer to the y-position of the top row of the neck.)

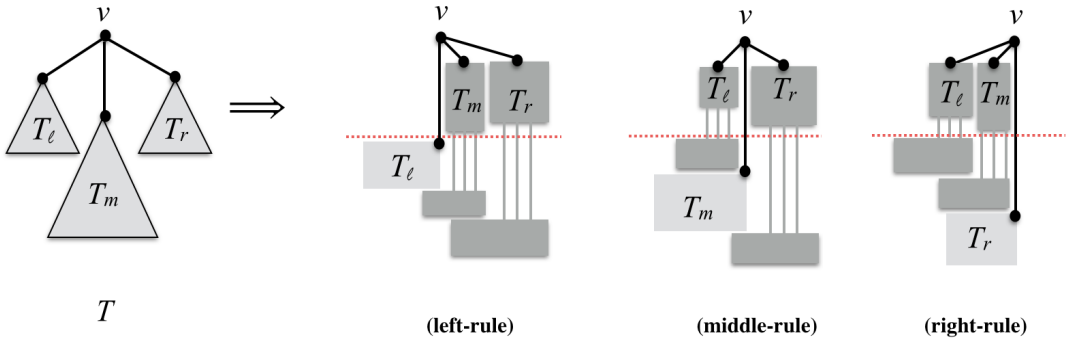


Figure 2.4: L-right uses the *left rule*, *middle rule*, and *right rule*. Dark grey denotes subtrees drawn using L-right and light grey denotes subtrees drawn using the Overhang Algorithm.

- In the left rule, we place the neck of $D(T_m)$ one unit below and immediately to the right of v and place the neck of $D(T_r)$ one unit below v and immediately to the right of the neck of $D(T_m)$. Then we place the root of $D(T_\ell)$ below the necks of $D(T_m)$ and $D(T_r)$ and vertically aligned with v (since we used a right-corner drawing, this puts all of $D(T_\ell)$ in the column of v or further left). We place the foot of $D(T_m)$ immediately below $D(T_\ell)$, and place the foot of $D(T_r)$ immediately below the foot of $D(T_m)$.
- In the middle rule, we place the neck of $D(T_\ell)$ one unit below and one unit to the left of v and we place the neck of $D(T_r)$ one unit below and one unit to the right of v. We place the foot of $D(T_\ell)$ below the necks of $D(T_\ell)$ and $D(T_r)$. We place the root of $D(T_m)$ immediately below the foot of $D(T_\ell)$ and vertically aligned with v. Then we place the foot of $D(T_r)$ immediately below $D(T_m)$.
- In the right rule, we place the neck of $D(T_m)$ one unit below and one unit to the left of v and we place the neck of $D(T_\ell)$ one unit below v and immediately to the left of the neck of $D(T_m)$. Then, we place the foot of $D(T_\ell)$ below the necks of $D(T_\ell)$ and $D(T_m)$ and place the foot of $D(T_m)$ immediately below the foot of $D(T_\ell)$. We place the root of $D(T_r)$ immediately below the foot of $D(T_m)$ and vertically aligned with v.

It is easy to see that the left rule, middle rule, and right rule produce an ideal HVA drawing since the resulting drawing is planar, straight-line, strictly upward, order-preserving, and only uses vertical or adjacent edges.

Let the *neck width*, denoted $W_{\text{neck}}(n)$, be the maximum possible width of the neck and let the *foot width*, denoted $W_{\text{foot}}(n)$, be the maximum possible width of the foot that could be created with this L-drawing algorithm on a ternary tree with n nodes.

Lemma 1. *Any ordered ternary tree of size n admits an HVA L-drawing of neck width $O(n^{\log_3 2})$.*

Proof. The *left rule*, *middle rule*, and *right rule* give rise to a sequence where $W_{\text{neck}}(0) = 0$, $W_{\text{neck}}(1) = 1$ and

$$W_{\text{neck}}(n) \leq W_{\text{neck}}(n_1) + W_{\text{neck}}(n_2) + 1, \text{ where } n_1 + n_2 \leq \lfloor \frac{2n}{3} \rfloor$$

since the largest of the three subtrees uses an Overhang right-corner drawing and therefore contributes only one unit to the neck width. Note that Hölder's inequality states

$$\sum_{k=1}^{m} |x_k y_k| \leq \left(\sum_{k=1}^{m} |x_k|^p \right)^{\frac{1}{p}} \left(\sum_{k=1}^{m} |y_k|^q \right)^{\frac{1}{q}}, \quad \text{for } p, q \geq 1 \text{ with } \frac{1}{p} + \frac{1}{q} = 1$$

and using Hölder's inequality with values $m = 2$, $(x_1, x_2) = ({n_1}^{\log_3 2}, {n_2}^{\log_3 2})$, $(y_1, y_2) = (1, 1)$, $p = \frac{1}{\log_3 2}$, $q = \frac{1}{1-\log_3 2}$, we can show that $n_1 + n_2 \leq \frac{2n}{3}$ implies $n_1^{\log_3 2} + n_2^{\log_3 2} \leq n^{\log_3 2}$ as follows:

$$\begin{aligned} n_1^{\log_3 2} + n_2^{\log_3 2} &\leq \left((n_1^{\log_3 2})^{\frac{1}{\log_3 2}} + (n_2^{\log_3 2})^{\frac{1}{\log_3 2}} \right)^{\log_3 2} \cdot \left(1 + 1 \right)^{1-\log_3 2} \\ &\leq (n_1 + n_2)^{\log_3 2} \cdot 2^{1-\log_3 2} \\ &\leq (\frac{2n}{3})^{\log_3 2} \cdot \frac{2}{2^{\log_3 2}} \\ &= \frac{(2n)^{\log_3 2}}{2^{\log_3 2}} \\ &= n^{\log_3 2}. \end{aligned}$$

Now we will prove by induction that for all $n \geq 1$,

$$W_{\text{neck}}(n) \leq 2n^{\log_3 2} - 1. \tag{2.1}$$

Base case: When $n = 1$, the left side of (2.1) is $W_{\text{neck}}(1) = 1$ and the right side is $2(1^{\log_3 2}) - 1 = 1$.
When $n = 2$, the left side of (2.1) is $W_{\text{neck}}(2) = 1$ and the right side is $2(2^{\log_3 2}) - 1 \geq 1$.

Inductive step: Let $N \geq 3$ and assume (2.1) is true for $1 \leq n < N$. Then,

$$\begin{aligned} W_{\text{neck}}(N) &\leq W_{\text{neck}}(n_1) + W_{\text{neck}}(n_2) + 1 \text{ where } n_1 + n_2 \leq \lfloor \frac{2N}{3} \rfloor \\ &\leq 2n_1^{\log_3 2} - 1 + 2n_2^{\log_3 2} - 1 + 1 \\ &\leq 2(n_1^{\log_3 2} + n_2^{\log_3 2}) - 1 \\ &\leq 2N^{\log_3 2} - 1. \end{aligned}$$

Therefore by induction, $W_{\text{neck}}(n) \leq 2n^{\log_3 2} - 1$ and $W_{\text{neck}}(n) \in O(n^{\log_3 2})$. □

We cannot yet analyze the foot width of L-drawings since it depends on the width obtained with the Overhang Algorithm. However we state the recurrence here for future reference for $W_{\text{foot}}(n)$:

Observation 1. *The left rule, middle rule, and right rule give rise to the following recurrence for $n > 1$.*

$$W_{foot}(n) \leq W_{neck}(n_2) + W_{neck}(n_3) + \max\left\{W_{overhang}(n_1), W_{foot}(n_2), W_{foot}(n_3)\right\}$$

for some n_1, n_2, n_3 where $n_1 + n_2 + n_3 \leq n$, $n_2 \leq n_1$, and $n_3 \leq n_1$ and $W_{overhang}(n_1)$ denotes the maximum possible width of the Overhang drawing for a ternary tree with n_1 nodes.

The Overhang Algorithm

We now describe the Overhang Algorithm. In this algorithm, some subtrees are recursively drawn using the Overhang Algorithm and some subtrees are recursively drawn using the L-Algorithm. The Overhang Algorithm can either produce a left-corner drawing or a right-corner drawing depending on whether we want the root vertex to be on the top-left corner or the top-right corner of the bounding box of the drawing. We describe here only the algorithm to produce *overhang-left*(T), the left-corner drawing of T obtained by using the Overhang Algorithm; the other algorithm is symmetric.

The algorithm proceeds by labeling each vertex in the heavy path as one of *left-knee*, *right-knee*, *ordinary-left*, *ordinary-right*, *switch-left*, or *switch-right*. Based on the label of v_i, the algorithm determines how to draw and place the left subtree L_i of v_i, middle subtree M_i of v_i, and right subtree R_i of v_i. Then the algorithm decides on the label of the heavy child v_{i+1} of v_i.

In the labelling of nodes in the heavy path, we will ensure that the following holds:

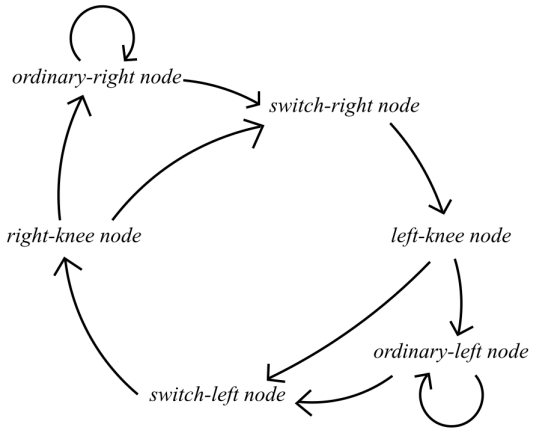


Figure 2.5: The labelling of nodes in the heavy path. Label A can follow label B if there is an in-bounding edge from A to B.

Invariant (I) An ordinary-left node does not have children to the left of its heavy child. Similarly, an ordinary-right node does not have children to the right of its heavy child.

Invariant (II) A switch-left node has a non-heavy child as a leftmost child. Similarly, a switch-right node has a non-heavy child as a rightmost child.

Invariant (III) The order of labels in the heavy path follows the restrictions shown in Figure 2.5.

To readers familiar with the literature, we remark that these labels have different definitions from Garg and Rusu's paper [10]. Our Overhang Algorithm is inspired by [10] in that we combine already constructed drawings of subtrees around the heavy path. Our method, however, is different in that we benefit from using vertically stretchable L-drawings.

1. If T is a single node, return the trivial drawing. Else, let $P = \langle v_0, v_1, v_2, \ldots, v_m \rangle$ be the *heavy path* (recall Definition 5) of the ternary tree T. For ease of notation, we assume that all nodes on the heavy path have exactly three children where the non-heavy children may be present or be empty subtrees.
2. Case for v_0.
   (a) First, assign v_1 as a *right-knee* node.

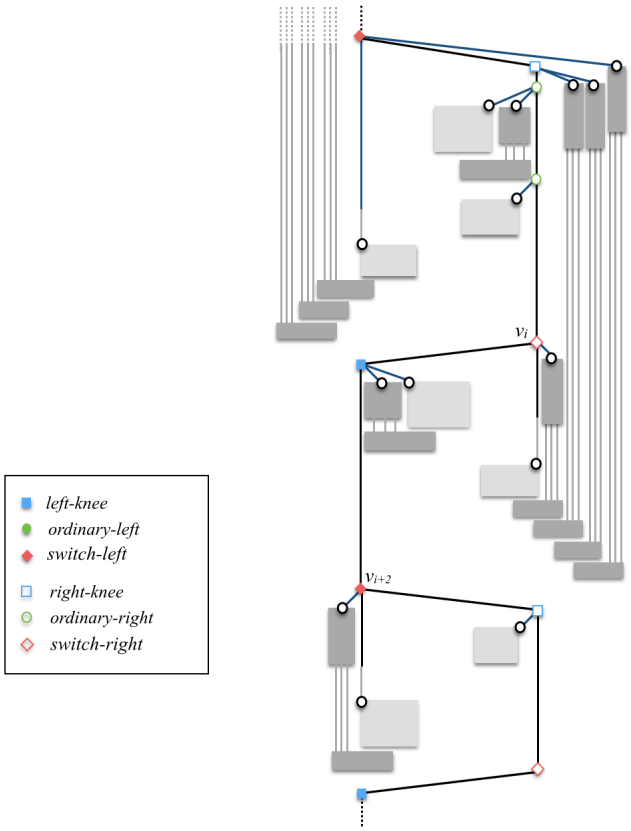


Figure 2.6: An HVA drawing produced by the Overhang Algorithm where the heavy path is drawn using black edges.

(b) If v_1 is a left child, call L-*right*(M_0) and L-*right*(R_0).
If v_1 is a middle child, call *overhang-left*(L_0) and L-*right*(R_0) (this case is illustrated in Figure 2.7).
If v_1 is a right child, call L-*left*(L_0) and L-*left*(M_0).

3. Now consider vertex v_i for some $i > 0$. Omit this step if v_i is the leaf of the heavy path.

(a) v_i is an *ordinary-left* node:
We know that v_i has no child to the left of its heavy child.

i. Construct L-*left*(M_i) and *overhang-left*(R_i).

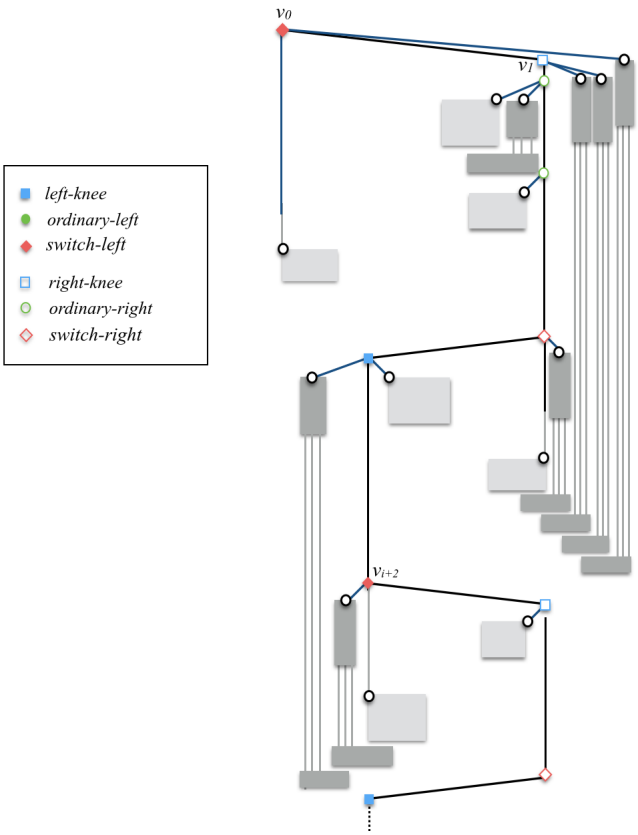


Figure 2.7: An HVA drawing produced by the Overhang Algorithm where the root v_0 is drawn in the top-left corner.

ii. If v_{i+2} does not exist or has no left sibling, then assign v_{i+1} as an *ordinary-left* node, else assign v_{i+1} as a *switch-left* node (note that this obeys Invariant (II)).

(b) v_i is a *switch-left* node:

i. Assign v_{i+1} as a *right-knee* node (note that this obeys Invariant (III)).

ii. If v_{i+1} is a right child, construct *overhang-left*(M_i) and L-*left*(L_i).
If v_{i+1} is a middle child, construct *overhang-left*(L_i) and L-*right*(R_i).
Note that v_i cannot have v_{i+1} as a left child, since this will violate Invariant (II).

(c) v_i is a *left-knee* node:

i. If v_{i+1} is a left child, construct L-*left*(M_i), and *overhang-left*(R_i).

If v_{i+1} is a middle child, construct L-*left*(L_i), and *overhang-left*(R_i).
If v_{i+1} is a right child, construct L-*left*(L_i), and L-*left*(M_i).

ii. If v_{i+2} has no left sibling, then assign v_{i+1} as an *ordinary-left* node (note that this obeys Invariant (I)), else assign v_{i+1} as a *switch-left* node.

(d) v_i is an *ordinary-right*, *right-knee* node, *switch-right* node: Do the same as in the cases, where v_i is an *ordinary-left*, *left-knee* node, or *switch-left* node, respectively with "left" exchanged with "right".

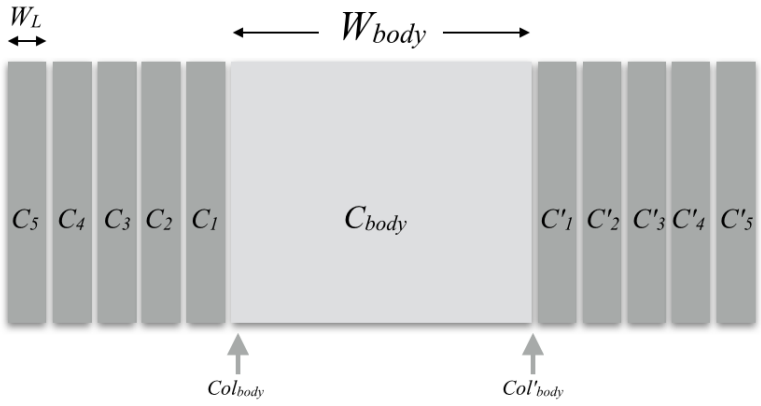


Figure 2.8: Grid structure used in the Overhang Algorithm for constructing HVA drawings.

4. Let W_{overhang} be the maximum width among the Overhang drawings constructed in Step 3. Let W_{L} be the maximum neck width among the L-drawings constructed in Step 3 and let W_{f} be the maximum foot width among the L-drawings constructed in Step 3. Set $W_{\text{body}} = \max\{W_{\text{overhang}} + W_{\text{L}} + 2,\ W_{\text{f}} + 2\}$, then imagine the following grid structure: Five channels of width W_{L}, followed by a single channel of width W_{body}, followed by five channels of width W_{L} (see Figure 2.8). Denote Col_{body} and $\text{Col}'_{\text{body}}$ to be the leftmost column and rightmost column within the channel C_{body}. The idea is to draw the heavy path within the channel C_{body}, and place the neck of L-drawings in the channels C_i and C'_i where $i = 1, \ldots, 5$.

5. Initialize two stacks which can hold up to six drawings each. We will use each stack to store pending-left drawings, and pending-right drawings. Pending-left drawings and pending-right drawings are either the foot of an L-drawing or an Overhang drawing and for each drawing, the x-coordinate (of some reference point of the bounding box)

is known at the time of push, however the y-coordinate (of the root of the drawing) is determined at the time of pop.

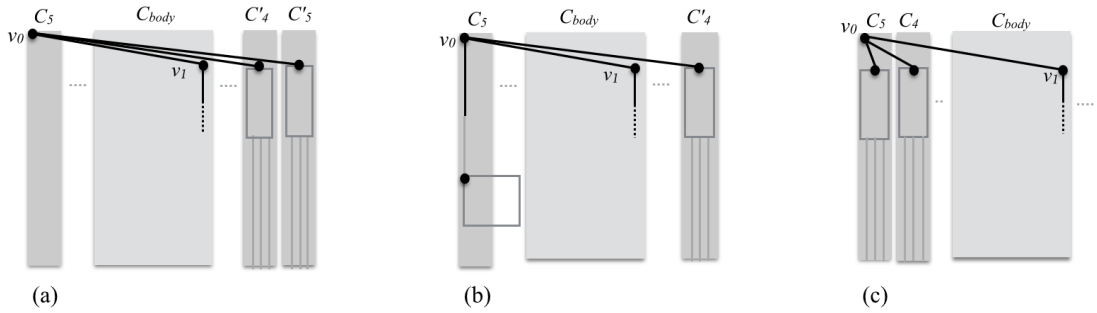


Figure 2.9: Placement of non-heavy subtrees of v_0 when (a) v_1 is the left child, (b) v_1 is the middle child, (c) v_1 is the right child.

In the following steps, we construct the Overhang drawing of T by drawing the heavy path and appropriately placing the subtree drawings around the heavy path. For each v_i in the heavy path, let $D(L_i)$, $D(M_i)$, and $D(R_i)$ denote the drawings of subtree L_i, M_i, and R_i respectively.

6. Place the root v_0 at the origin (since we are constructing a left-corner overhang drawing, this would be the top-left corner of channel C_5). Place v_1 one unit below v_0 and in $\text{Col}'_{\text{body}}$. Then we distinguish the following cases:

   (a) v_1 is a left child (see Figure 2.9a):
   Since $D(M_0)$ and $D(R_0)$ are both L-drawings, they are composed of a neck and a foot. Place the neck of $D(M_0)$ one unit below v_0 and within channel C'_4. Place the neck of $D(R_0)$ one unit below v_0 and within channel C'_5. Push the foot of $D(R_0)$, then the foot of $D(M_0)$ onto the stack of pending-right drawings.

   (b) v_1 is a middle child (see Figure 2.9b):
   Fix the x-position of the root of $D(L_0)$ to be the same as v_0 and push $D(L_0)$ onto the stack of pending-left drawings. Since $D(R_0)$ is an L-drawing, it is composed of a neck and a foot. Place the neck of $D(R_0)$ one unit below v_0 and within channel C'_4. Push the foot of $D(R_0)$ onto the stack of pending-right drawings.

   (c) v_1 is a right child (see Figure 2.9c):
   Place the neck of $D(L_0)$ one unit below v_0 and within channel C_5. Place the

neck of $D(M_0)$ one unit below v_0 and within channel C_4. Push the foot of $D(L_0)$, then the foot of $D(M_0)$ onto the stack of pending-left drawings.

7. Let H_i be the horizontal row corresponding to the node placed lowest in the drawing of T constructed so far. For $i = 1, \ldots, m$, depending on the label of v_i, do:

   (a) v_i is an *ordinary-left* node (see Figure 2.10a):
   Place v_i one unit below H_i and vertically aligned with v_{i-1}. Then place the neck of $D(M_i)$ one unit below and to the right of v_i and place $D(R_i)$ one unit to the right of the neck of $D(M_i)$. Place the foot of $D(M_i)$ in the same vertical channel as the neck of $D(M_i)$ and immediately below the bounding box of $D(R_i)$ and the neck of $D(M_i)$.

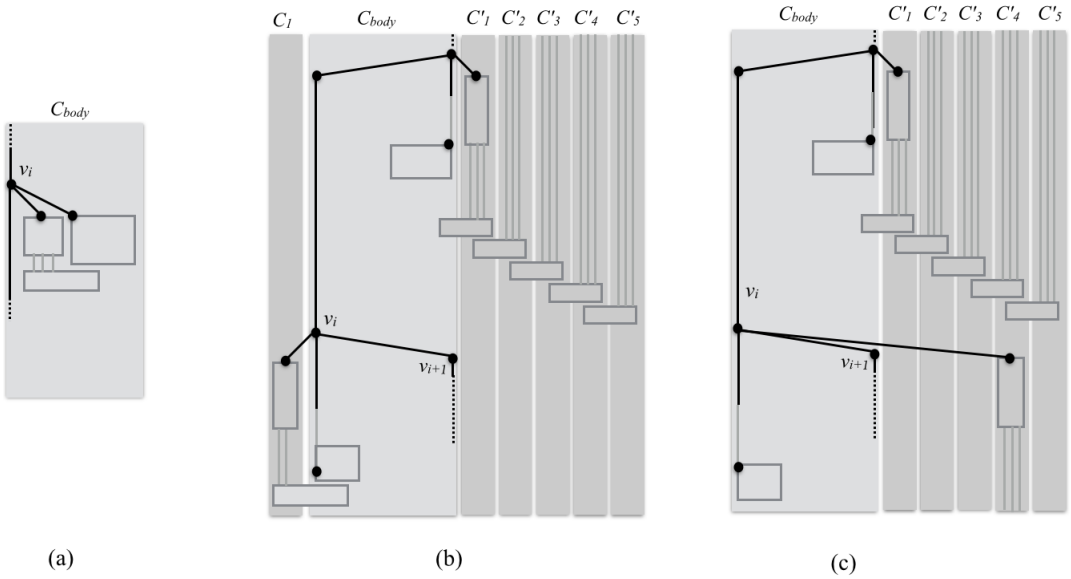


Figure 2.10: (a) v_i is an *ordinary-left* node. (b) v_i is a *switch-left* node and v_{i+1} is the right child. (c) v_i is a *switch-left* node and v_{i+1} is the middle child.

   (b) v_i is a *switch-left* node:
   While the stack of pending-right drawings is not empty, pop one drawing and place it one unit below H_i. Note that the x-position of the drawing is known at the time of push and only the y-position need be decided at the time of pop. Once the stack of pending-right drawings is empty, place v_i in the same column

as v_{i-1} and one unit below the new H_i. Place v_{i+1} one unit below v_i and in $\text{Col}'_{\text{body}}$. Then

i. if v_{i+1} is a right child (see Figure 2.10b):
Place the neck of $D(L_i)$ one unit below v_i and within channel C_1. Push the foot of $D(L_i)$ onto the stack of pending-left drawings. Fix the x-position of the root of $D(M_i)$ to be the same as v_i and push $D(M_i)$ onto the stack of pending-left drawings.

ii. Else by the invariant of *switch-left* nodes, v_{i+1} is a middle child (see Figure 2.10c):
Fix the x-position of the root of $D(L_i)$ to be the same as v_i and push $D(L_i)$ onto the stack of pending-left drawings. Place the neck of $D(R_i)$ one unit below v_i and within channel C'_4. Push the foot of $D(R_i)$ onto the stack of pending-right drawings.

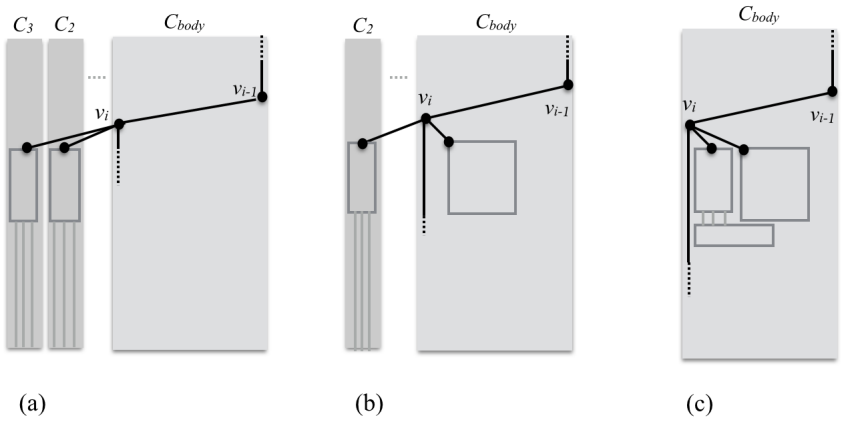


Figure 2.11: (a) v_i is a *left-knee* node and v_{i+1} is the right child. (b) v_i is a *left-knee* node and v_{i+1} is the middle child. (c) v_i is a *left-knee* node and v_{i+1} is the left child.

(c) v_i is a *left-knee* node:
Depending on whether v_{i+1} is the right, middle, or left child, do:

i. v_{i+1} is a right child (see Figure 2.11a):
Place the neck of $D(M_i)$ one unit below v_i and within channel C_2. Place the neck of $D(L_i)$ one unit below v_i and within channel C_3. Push the foot of $D(L_i)$, then push the foot of $D(M_i)$ onto the stack of pending-left drawings.

ii. v_{i+1} is a middle child (See Figure 2.11b):
Place the neck of $D(L_i)$ one unit below v_i and within channel C_2. Then

push the foot of $D(L_i)$ onto the stack of pending-left drawings. Place $D(R_i)$ one unit below and to the right of v_i.

iii. v_{i+1} is a left child (see Figure 2.11c):
Place the neck of $D(M_i)$ one unit below and to the right of v_i. Place $D(R_i)$ one unit below v_i and immediately to the right of the neck of $D(M_i)$. Place the foot of $D(M_i)$ in the same vertical channel as the neck of $D(M_i)$ and immediately below the bounding box of $D(R_i)$ and the neck of $D(M_i)$.

(d) v_i is an *ordinary-right*, *right-knee* node, *switch-right* node:
These cases are the same as the cases where v_i is an *ordinary-left*, *left-knee* node, or *switch-left* node respectively, with "left" exchanged with "right" and C_i exchanged with C'_i for $i = 1, \ldots, 5$.

Note that the stack of pending-right drawings empty every time we process a *switch-left* node (similarly, the stack of pending-left drawings empty every time we process a *switch-right* node). We argue that we push at most six drawings onto the stack of pending-right drawings while processing a sequence of labels beginning from a *switch-left* node and ending in the next *switch-left* node. In this sequence of labels, the labels that could push drawings on to the pending-right stack include, in order along the heavy path, the *switch-left*, the *right-knee*, and the *switch-right* node. The *switch-left* node may use channels C_1, C'_4 and column $\mathrm{Col}_{\mathrm{body}}$. The *right-knee* node may use channels C'_3, C'_2. The *switch-right* node uses channel C'_1, C_4 and column Col'_{body}. All in all, we push at most six drawings onto the stack of pending-right and each corresponds to a channel/column in order from right to left. (Note that channel C'_5 is only used by the root node in a special case (Case 6.a). See Figure 2.9a.) Hence these drawings can be extracted and placed when handling the next *switch-left* node.

Theorem 1. *Any ordered ternary tree of size n admits an ideal HVA drawing of width $O(n^{\log_3 2})$.*

Proof. As seen from the grid structure of the Overhang Algorithm, the width of the drawing follows a sequence where $W_{\mathrm{overhang}}(1) = 1$ and

$$\begin{aligned} W_{\mathrm{overhang}}(n) &\leq 10W_{\mathrm{L}} + W_{\mathrm{body}} \\ &\leq 10W_{\mathrm{neck}}(n_1) + \max\left\{W_{\mathrm{overhang}}(n_2) + W_{\mathrm{neck}}(n_1) + 2, \quad W_{\mathrm{foot}}(n_3) + 2\right\} \\ &\leq \max\left\{11W_{\mathrm{neck}}(n_1) + W_{\mathrm{overhang}}(n_2), \quad 10W_{\mathrm{neck}}(n_1) + W_{\mathrm{foot}}(n_3)\right\} + 2 \end{aligned}$$

for some n_1, n_2, n_3 where $n_1, n_2, n_3 \leq \lfloor \frac{n}{2} \rfloor$.
We will prove by induction that for all $n \geq 1$,

$$W_{\mathrm{overhang}}(n) \leq 40n^{\log_3 2} \tag{2.2}$$

and

$$W_{\text{foot}}(n) \leq 42n^{\log_3 2}. \tag{2.3}$$

Base case: When $n = 1$, the left side of (2.2) is $W_{\text{overhang}}(1) = 1$ and the right side is $40(1^{\log_3 2}) = 40$.
When $n = 1$, the left side of (2.3) is $W_{\text{foot}}(1) = 1$ and the right side is $42(1^{\log_3 2}) = 42$.

Inductive step: Let $N > 1$ and assume (2.2) and (2.3) are true for $1 \leq n < N$.
We refer to Observation 1 (on page 14) and solve the following recurrence where $n_2 \leq n_1$ and $n_3 \leq n_1$ and $n_1 + n_2 + n_3 \leq N$.

$$\begin{aligned}
W_{\text{foot}}(N) &\leq W_{\text{neck}}(n_2) + W_{\text{neck}}(n_3) + \max\{W_{\text{overhang}}(n_1), W_{\text{foot}}(n_2), W_{\text{foot}}(n_3)\} \\
&\quad \text{(by Equation 2.1)} \\
&\leq (2n_2^{\log_3 2} - 1) + (2n_3^{\log_3 2} - 1) + \max\{40n_1^{\log_3 2},\ 42n_2^{\log_3 2},\ 42n_3^{\log_3 2}\} \\
&\leq 2N^{\log_3 2} - 2 + \max\{40N^{\log_3 2},\ 42\Big(\frac{N}{2}\Big)^{\log_3 2}\} \\
&\quad \text{(using Hölder's inequality since } n_2, n_3 \leq n_1 \text{ and therefore } n_2 + n_3 \leq \frac{2N}{3}) \\
&\leq \max\{42N^{\log_3 2},\ (2 + 42\Big(\frac{1}{2}\Big)^{\log_3 2})N^{\log_3 2}\} \\
&\leq 42N^{\log_3 2}.
\end{aligned}$$

By induction, $W_{\text{foot}}(n) \leq 42n^{\log_3 2}$.
Now we solve the recurrence for $W_{\text{overhang}}(N)$ where $n_1, n_2, n_3 \leq \lfloor \frac{N}{2} \rfloor$.

$$\begin{aligned}
W_{\text{overhang}}(N) &\leq \max\left\{11W_{\text{neck}}(n_1) + W_{\text{overhang}}(n_2),\ \ 10W_{\text{neck}}(n_1) + W_{\text{foot}}(n_3)\right\} + 2 \\
&\leq \max\Big\{22\Big(\frac{N}{2}\Big)^{\log_3 2} - 11 + 40\Big(\frac{N}{2}\Big)^{\log_3 2}, \\
&\qquad 20\Big(\frac{N}{2}\Big)^{\log_3 2} - 10 + 42\Big(\frac{N}{2}\Big)^{\log_3 2}\Big\} + 2 \\
&\leq 62\Big(\frac{N}{2}\Big)^{\log_3 2} \\
&\leq 40N^{\log_3 2}.
\end{aligned}$$

Therefore by induction, $W_{\text{overhang}}(n) \leq 40n^{\log_3 2} \in O(n^{\log_3 2})$.

Each part of our construction has respected the order of children and drawn children below their parent. In fact, one immediately verifies that the drawing is strictly upward (there are no horizontal edges) and strictly order-preserving. □

Since the algorithm uses a constant number of operations per node, the running time for the Overhang Algorithm is $O(n)$.

Generalizing the Overhang Algorithm for k-ary trees

We can extend the Overhang Algorithm to construct HVA drawings of general k-ary trees (Figure 2.12 illustrates a drawing produced by the Overhang Algorithm for $k = 5$). For HVA drawings, this is easy to do because there are no natural grid restrictions. We note here that our Overhang algorithm is different from [10] in that the strictly-upward property (i.e. for every edge, the parent node has a y-coordinate greater than the child node) holds even when we generalize the algorithm from drawing ternary trees to k-ary trees. Given an L-drawing of a k-ary tree with n nodes, let $W^k_{\text{neck}}(n)$ denote the width of the neck and let $W^k_{\text{foot}}(n)$ denote the width of the foot.

Lemma 2. *Any ordered k-ary tree of size n admits an ideal HVA L-drawing of neck width $O(n^{\log_k (k-1)})$.*

Proof. The L-algorithm for k-ary trees follows rules similar to the ones seen in the ternary tree case, except there will be k rules instead of three. In each rule, the subtree rooted at the heavy child will be drawn recursively using the Overhang algorithm and the subtrees rooted at the non-heavy children will be drawn recursively using the L-algorithm. Hence, the neck width of k-ary L-drawings follows the recurrence

$$W^k_{\text{neck}}(n) \leq \sum_{i=1}^{k-1} W^k_{\text{neck}}(n_i) + 1 \text{ where } \sum_{i=1}^{k-1} n_i \leq \frac{(k-1)n}{k}.$$

Using Hölder's inequality, one can show that $\sum_{i=1}^{k-1} n_i \leq \frac{(k-1)n}{k}$ implies $\sum_{i=1}^{k-1} n_i^{\log_k (k-1)} \leq n^{\log_k k-1}$. By induction, then $W^k_{\text{neck}}(n) \leq cn^{\log_k k-1} - b$ for some constants c and b. □

Observation 2. *The foot width of k-ary HVA L-drawings follows the recurrence*

$$W^k_{foot}(n) \leq \sum_{i=2}^{k} W^k_{neck}(n_i) + \max_{i\in\{2,\ldots,k\}} \left\{W^k_{overhang}(n_1), W^k_{foot}(n_i)\right\}$$

where $n_1 \geq n_i$ for $i \in \{2, \ldots, k\}$ and $W^k_{overhang}(n)$ denotes the width of the drawing created by the Overhang algorithm for a tree with n_1 nodes.

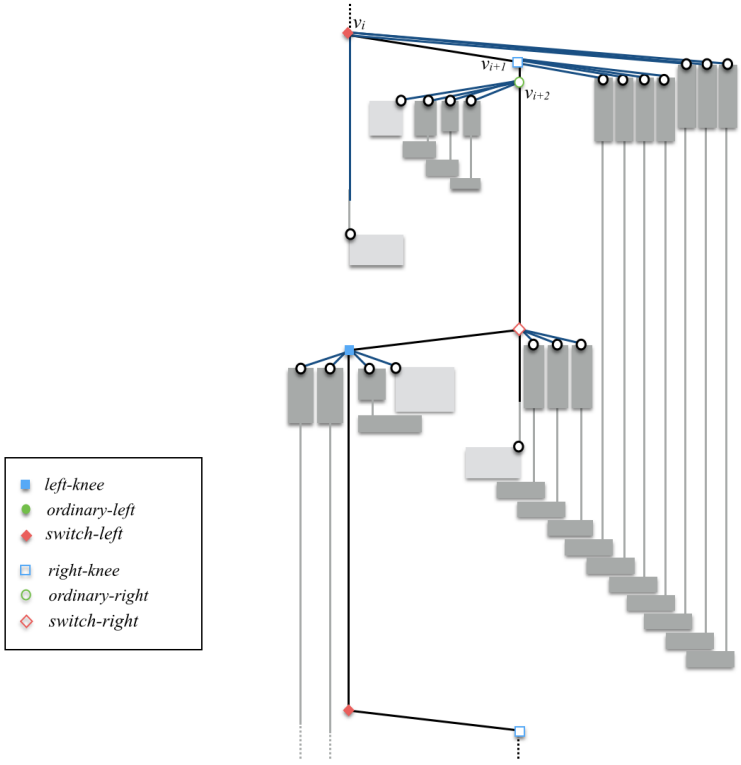


Figure 2.12: An HVA drawing of a k-ary tree produced by the Overhang Algorithm where $k = 5$.

Theorem 2. *Any ordered k-ary tree of size n admits an ideal HVA drawing of width $O(kn^{\log_k(k-1)})$.*

Proof. The grid structure of the Overhang Algorithm can be extended with $6k - 8 \in O(k)$ channels to accommodate k-ary trees, resulting in the recurrence

$$W^k_{\text{overhang}}(n) \leq \max\left\{W^k_{\text{overhang}}(n_1) + O(k)W^k_{\text{neck}}(n_2),\ W^k_{\text{foot}}(n_3) + O(k)W^k_{\text{neck}}(n_2)\right\} + 2$$

where $n_1, n_2, n_3 \leq \frac{n}{2}$. Using Lemma 2 and Observation 2, we can simplify the recurrence as

$$W^k_{\text{overhang}}(n) \leq W^k_{\text{overhang}}\left(\frac{n}{2}\right) + O(k)\left(\frac{n}{2}\right)^{\log_k(k-1)} + 2$$

which solves to $W^k_{\text{overhang}}(n) \in O(kn^{\log_k(k-1)})$. □

2.2 Overhang Algorithm for Octagonal Drawings

In this section, the Overhang Algorithm and L-Algorithm refer to the versions that produce an octagonal drawing.

The L-Algorithm

As opposed to HVA drawings, octagonal drawings are restricted to horizontal, vertical, or diagonal edges. The old left-rule draws an edge neither horizontal nor diagonal and the same split seen in the old left-rule is not possible for octagonal drawings since we cannot make the edge horizontal unless the L-drawing has the root in the right place. Therefore we need a new set of rules for the L-Algorithm to produce an octagonal drawing.

The L-right algorithm (for octagonal drawings) uses the *left rule*, *more-left rule*, *middle rule*, *right rule*, and *more-right rule*, to construct ideal drawings of its subtrees and to combine into an ideal drawing of T. (See Figure 2.13.)

First, we describe which rule to apply at the current root, and which algorithm to use to recursively draw the various subtrees. Let L-*right*(T) denote the drawing of ternary tree T obtained by using the L-right algorithm and let *overhang-right*(T) denote the right-corner drawing of T obtained by using the Overhang Algorithm.

The algorithms depend on two constants α and c_0; we will see later that $c_0 = 0.2783297$ and $\alpha = \log_{\frac{2}{1-c_0}} 2 \leq 0.68$ are suitable values for them (this choice of c_0 will be motivated by Lemma 4 which we will state later).

Let T be a given ternary tree with root vertex v, left subtree L, middle subtree M, and right subtree R. Let v_ℓ denote the left child of v and v_r denote the right child of v. Let L_ℓ, L_m, and L_r denote the left, middle, and right subtree of v_ℓ. Similarly denote the left, middle, and right subtree of v_r by R_ℓ, R_m, and R_r. Let $v_{\ell'}$ denote the left child of v_ℓ and $L_{\ell\ell}$, $L_{\ell m}$, and $L_{\ell r}$ denote the left, middle, and right subtrees of $v_{\ell'}$. Similarly, let $v_{r'}$ denote the right child of v_r and let $R_{r\ell}$, R_{rm}, and R_{rr} denote the left, middle, and right subtree of v'_r.

- If $|T| \leq 1$, return the trivial drawing. Otherwise, compare the size of subtrees L, M, and R.
- If $|M| \geq c_0 n$, recursively compute L-*right*(L), L-*right*(R), and *overhang-right*(M). Then combine the drawings using the middle rule (we define these rules below). (See Figure 2.13)

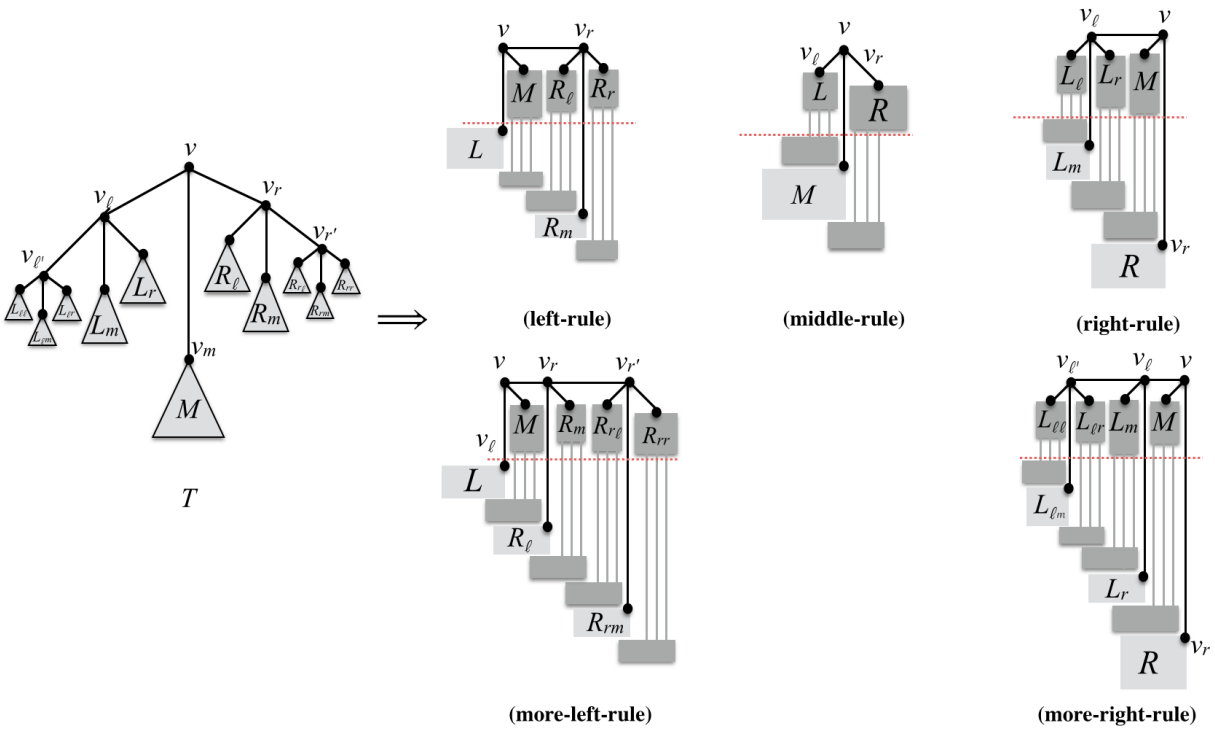


Figure 2.13: The L-right algorithm uses the left rule, more-left rule, middle rule, right rule, and more-right rule. Dark grey denotes subtrees drawn using L-right and light grey denotes subtrees drawn using the Overhang Algorithm.

- Else, assume $|R| \geq |L|$ (which implies $|R| \geq c_0 n$ since $c_0 \leq \frac{1}{3}$). Define $a_M := \frac{|M|}{n}$, $a_{L_m} := \frac{|L_m|}{n}$, $a_{L_{\ell\ell}} := \frac{|L_{\ell\ell}|}{n}$, $a_{L_{\ell r}} := \frac{|L_{\ell r}|}{n}$, and test whether

$$a_{L_m}^{\alpha} + a_{L_{\ell\ell}}^{\alpha} + a_{L_{\ell r}}^{\alpha} + a_M^{\alpha} > 1. \qquad (*)$$

  i. If $(*)$ is satisfied, recursively compute the drawings L-*right*(L_ℓ), L-*right*(L_r), L-*right*(M), *overhang-right*(L_m), and *overhang-right*(R). Then combine the drawings using the right rule.

  ii. Else ($(*)$ is not satisfied), recursively compute the drawings L-*right*$(L_{\ell\ell})$, L-*right*$(L_{\ell r})$, L-*right*(L_m), L-*right*(M), *overhang-right*$(L_{\ell m})$, *overhang-right*(L_r), and *overhang-right*(R). Then combine the drawings using the more-right rule.

- Else, assume $|L| \geq |R|$ (which implies $|L| \geq c_0 n$). Define $b_M := \frac{|M|}{n}$, $b_{R_m} := \frac{|R_m|}{n}$,

$b_{R_{r\ell}} := \frac{|R_{r\ell}|}{n}$, $b_{R_{rr}} := \frac{|R_{rr}|}{n}$, and test whether

$$b_{R_m}^{\alpha} + b_{R_{r\ell}}^{\alpha} + b_{R_{rr}}^{\alpha} + b_M^{\alpha} > 1. \quad (**)$$

i. If $(**)$ is satisfied, then recursively compute L-*right*(M), L-*right*(R_ℓ), L-*right*(R_r), *overhang-right*(L), and *overhang-right*(R_m). Then combine the drawings using the left rule.

ii. Else ($(**)$ is not satisfied), recursively compute L-*right*(M), L-*right*(R_{rm}), L-*right*$(R_{r\ell})$, L-*right*(R_{rr}), *overhang-right*(L), *overhang-right*(R_ℓ), and *overhang-right*(R_{rm}). Then combine the drawings using the more-left rule.

Now we describe how to combine the drawings for the left rule, more-left rule, middle rule, right rule, and more-right rule. Note that since we are dealing with octagonal drawings, when describing where to place a subtree drawing which is connected to its parent node using a diagonal edge, it suffices to describe only the x-position of the bounding box of the subtree drawing in relation to its parent node. Further, due to the vertically stretchable nature of L-drawings, the x-position of the neck fixes the x-position of the foot, while we have freedom in choosing the y-position of the foot. For a subtree T, let $D(T)$ denote the drawing of subtree T.

- In the left rule, we place the neck of $D(M)$ immediately to the right of v using a diagonal edge and place the neck of $D(R_\ell)$ immediately to the right of the neck of $D(M)$ using a diagonal edge such that v_r is placed immediately to the right of the neck of $D(R_\ell)$ and horizontally aligned with v. Place the neck of $D(R_r)$ one unit to the right of v_r using a diagonal edge. Place the root of L immediately below the necks of $D(M)$, $D(R_\ell)$, and $D(R_r)$ and vertically aligned with v. Place the foot of $D(M)$ immediately below the bounding box of $D(L)$ and place the foot of $D(R_\ell)$ immediately below the foot of $D(M)$. Place the root of R_m immediately below the foot of $D(R_\ell)$ and vertically aligned with v_r. Place the foot of $D(R_r)$ immediately below the bounding box of $D(R_m)$.

- In the more-left rule, we place the neck of $D(M)$ one unit to the right of v using a diagonal edge, place v_r immediately to the right of the neck of $D(M)$ and horizontally aligned with v, and place the neck of $D(R_m)$ immediately to the right of v_r using a diagonal edge. Place the neck of $D(R_{r\ell})$ immediately to the right of the neck of $D(R_m)$ using a diagonal edge such that $v_{r'}$ is placed immediately to the right of the neck of $D(R_{r\ell})$ and horizontally aligned with v_r. Place the neck of $D(R_{rr})$ one unit to the right of $v_{r'}$ using a diagonal edge. Place the root of L immediately below the

necks of $D(M)$, $D(R_m)$, $D(R_{r\ell})$, and $D(R_{rr})$ and vertically aligned with v. Place the foot of $D(M)$ immediately below the bounding box of $D(L)$, place the root of R_ℓ immediately below the foot of $D(M)$ and vertically aligned with v_r. Place the foot of $D(R_m)$ immediately below the bounding box of $D(R_\ell)$ and place the foot of $D(R_{r\ell})$ immediately below the foot of $D(R_m)$. Place the root of R_{rm} immediately below the foot of $D(R_{r\ell})$ and vertically aligned with $v_{r'}$ and place the foot of drawing $D(R_{rr})$ immediately below the bounding box of $D(R_{rm})$.

- The middle rule is almost exactly as for HVA drawings except that we use diagonal edges. Thus, place the neck of $D(L)$ one unit to the left of v using a diagonal edge and place the neck of $D(R)$ one unit to the right of v using a diagonal edge. Place the foot of $D(L)$ below the necks of $D(L)$ and $D(R)$. We place the root of M immediately below the foot of $D(L)$ and vertically aligned with v. Then place the foot of $D(R)$ immediately below $D(M)$.

- The right rule and more-right rule is symmetric to the left rule and more-left rule.

Let the neck width, denoted $W_{\text{neck}}(n)$, be the maximum possible width of the neck and let the foot width, denoted $W_{\text{foot}}(n)$, be the maximum possible width of the foot that could be created with this L-drawing algorithm on a ternary tree with n nodes.

Lemma 3. *Any ordered ternary tree of size n admits an octagonal L-drawing of neck width $O(n^{0.68})$.*

Proof. Since $\alpha \leq 0.68$, it suffice to prove by induction that for all $n \geq 1$,

$$W_{\text{neck}}(n) \leq cn^{\alpha} - 1 \text{ for a suitable constant } c. \tag{2.4}$$

This holds in the base case for a suitable choice of c. Since different rules give rise to different neck width recurrences, we break up the proof into the middle rule, right rule, and more-right rule. (We omit the left rule and more-left rule since they are symmetric to right rule and more-right rule respectively.)

i. The middle rule gives rise to the following recurrence:

$$W_{\text{neck}}(n) \leq W_{\text{neck}}(|L|) + W_{\text{neck}}(|R|) + 1$$

where, since $|M| \geq c_0 n$ and $|L| + |M| + |R| \leq n$, we can deduce $|L| + |R| \leq (1 - c_0)n$. Using Hölder's Inequality, $|L| + |R| \leq (1 - c_0)n$ implies $|L|^{\log_{\frac{2}{1-c_0}} 2} + |R|^{\log_{\frac{2}{1-c_0}} 2} \leq n^{\log_{\frac{2}{1-c_0}} 2}$ and by induction, $W_{\text{neck}}(n) \leq cn^{\log_{\frac{2}{1-c_0}} 2} - 1$. The claim now holds since we chose $\alpha = \log_{\frac{2}{1-c_0}} 2$.

ii. The more-right rule gives rise to the recurrence:

$$\begin{aligned} W_{\text{neck}}(n) &\leq W_{\text{neck}}(|L_{\ell\ell}|) + W_{\text{neck}}(|L_{\ell r}|) + W_{\text{neck}}(|L_m|) + W_{\text{neck}}(|M|) + 3 \\ &\leq c(|L_{\ell\ell}|^\alpha + |L_{\ell r}|^\alpha + |L_m|^\alpha + |M|^\alpha) - 1 \end{aligned}$$

Recall that we defined $a_M, a_{L_m}, a_{L_{\ell\ell}}, a_{L_{\ell r}}$ such that $|M| = a_M n$, $|L_m| = a_{L_m} n$, $|L_{\ell\ell}| = a_{L_{\ell\ell}} n$, and $|L_{\ell r}| = a_{L_{\ell r}} n$. Therefore

$$W_{\text{neck}}(n) \leq c(a_{L_{\ell\ell}}^\alpha + a_{L_{\ell r}}^\alpha + a_{L_m}^\alpha + a_M^\alpha) n^\alpha - 1.$$

The algorithm chooses the more-right rule when the following is satisfied

$$a_{L_m}^\alpha + a_{L_{\ell\ell}}^\alpha + a_{L_{\ell r}}^\alpha + a_M^\alpha \leq 1,$$

which implies $W_{\text{neck}}(n) \leq cn^\alpha - 1$ whenever the algorithm chooses the more-right rule.

iii. The right rule gives rise to the recurrence:

$$W_{\text{neck}}(n) \leq W_{\text{neck}}(|L_\ell|) + W_{\text{neck}}(|L_r|) + W_{\text{neck}}(|M|) + 2$$

Define $a_{L_r} := \frac{|L_r|}{n}$, $a_{L_\ell} := \frac{|L_\ell|}{n}$ and recall that we defined a_M, a_{L_m} such that $|M| = a_M n$, $|L_m| = a_{L_m} n$. Therefore

$$W_{\text{neck}}(n) \leq W_{\text{neck}}(a_{L_\ell} n) + W_{\text{neck}}(a_{L_r} n) + W_{\text{neck}}(a_M n) + 2.$$

Since $|L| + |R| \leq (1 - a_M)n$ and $|L| \leq |R|$, we can deduce that $|L| \leq (\frac{1-a_M}{2})n$. Furthermore, we have $|L_\ell| = a_{L_\ell} n \leq (\frac{1-a_M}{2} - a_{L_r} - a_{L_m})n$. Therefore

$$W_{\text{neck}}(n) \leq W_{\text{neck}}(a_{L_\ell} n) + W_{\text{neck}}\Big(\Big(\frac{1 - a_M}{2} - a_{L_\ell} - a_{L_m}\Big)n\Big) + W_{\text{neck}}(a_M n) + 2.$$

Applying the induction hypothesis, we get

$$W_{\text{neck}}(n) \leq c\Big[a_M^\alpha + a_{L_\ell}^\alpha + \Big(\frac{1 - a_M}{2} - a_{L_\ell} - a_{L_m}\Big)^\alpha\Big] n^\alpha - 1. \qquad (\dagger)$$

Recall that the algorithm chooses the right rule when the following is satisfied

$$a_{L_m}^\alpha + a_{L_{\ell\ell}}^\alpha + a_{L_{\ell r}}^\alpha + a_M^\alpha > 1.$$

Using Hölder's inequality with values $m = 2$, $(x_1, x_2) = (a_{L_{\ell\ell}}^\alpha, a_{L_{\ell r}}^\alpha)$, $(y_1, y_2) = (1, 1)$, $p = \frac{1}{\alpha}$, $q = \frac{1}{1-\alpha}$, we can show that $a_{L_{\ell\ell}} + a_{L_{\ell r}} \leq a_{L_\ell}$ implies $a_{L_{\ell\ell}}^\alpha + a_{L_{\ell r}}^\alpha \leq 2^{1-\alpha} a_{L_\ell}^\alpha$. This implies that the algorithm chooses the right rule when

$$a_{L_m}^\alpha > 1 - 2^{1-\alpha} a_{L_\ell}^\alpha - a_M^\alpha.$$

Now we can rewrite (†) as

$$W_{\text{neck}}(n) \le c\Big[a_M^\alpha + a_{L_\ell}^\alpha + \Big(\frac{1-a_M}{2} - a_{L_\ell} - \max\{1 - 2^{1-\alpha}a_{L_\ell}^\alpha - a_M^\alpha,\ 0\}^{\frac{1}{\alpha}}\Big)^\alpha\Big]n^\alpha - 1.$$

Using Lemma 4 (see below), the following holds

$$a_M^\alpha + a_{L_\ell}^\alpha + \Big(\frac{1-a_M}{2} - a_{L_\ell} - \max\{1 - 2^{1-\alpha}a_{L_\ell}^\alpha - a_M^\alpha,\ 0\}^{\frac{1}{\alpha}}\Big)^\alpha \le 1$$

for $0 \le a_{L_\ell} \le \frac{1-a_M}{2}$ and $0 \le a_M \le c_0$. Therefore we can conclude $W_{\text{neck}}(n) \le cn^\alpha - 1$.

□

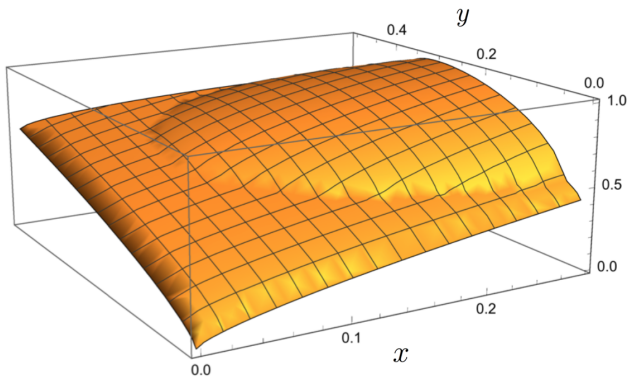


Figure 2.14: Surface plot of $F(x, y)$

Lemma 4. *Let* $c_0 = 0.2783297$, $\alpha = \log_{\frac{2}{1-c_0}} 2 < 0.68$, *and*

$$F(x,y) = x^\alpha + y^\alpha + \max\left\{\frac{1-x}{2} - y - \max\{1 - 2^{1-\alpha}y^\alpha - x^\alpha,\ 0\}^{\frac{1}{\alpha}},\ 0\right\}^\alpha$$

Then $\max F(x, y) \le 1$ *over all* x, y *where* $0 \le x \le c_0$ *and* $0 \le y \le \frac{1-x}{2}$.

Proof. The statement has been verified by writing a Mathematica program using interval arithmetic. See Figure 2.14 for a surface plot of $F(x, y)$. □

We cannot yet analyze the foot width of L-drawings since it depends on the width obtained with the Overhang Algorithm. However we state the recurrence here for future reference.

Observation 3. *We can observe that the left rule, more-left rule, middle rule, right rule, and more-right rule give rise to the following recurrence.*

$$W_{foot}(n) \leq \left(\sum_{i=1}^{4} W_{neck}(n_i) \right) + \max \left\{ W_{foot}(n_5), W_{overhang}(n_6) \right\}$$

for some n_i where $\sum_{i=1}^{6} n_i \leq n$, $n_5 \leq (1 - c_0)n$, and $W_{overhang}(n_6)$ denotes the width of the Overhang drawing with n_6 nodes.

The Overhang Algorithm

We now describe the Overhang Algorithm. Similar to the one for HVA drawings, some subtrees are recursively drawn using the Overhang Algorithm and some subtrees are recursively drawn using the L-Algorithm. As before the Overhang Algorithm can either produce a left-corner drawing or a right-corner drawing depending on whether we want the root vertex to be on the top-left corner or the top-right corner of the bounding box of the drawing. We describe here only the algorithm to produce *overhang-left*(T), the left-corner drawing of T obtained by using the Overhang Algorithm; the other algorithm is symmetric.

The algorithm proceeds again by labeling each vertex in the heavy path as one of *left-knee*, *right-knee*, *ordinary-left*, *ordinary-right*, *switch-left*, or *switch-right*. In fact, the labelling of nodes in the heavy path is identical to the HVA drawing model except at the root node, and the same invariants as in Section 2.1 hold. However, deciding between the L-Algorithm and the Overhang Algorithm to recursively draw the non-heavy subtrees and deciding where to place the already constructed drawings of non-heavy subtrees is different in this version of the Overhang Algorithm.

1. If T is a single node, return the trivial drawing. Else, let $P = \langle v_0, v_1, v_2, \ldots, v_m \rangle$ be the *heavy path* of the ternary tree T. We will need names for the subtrees for each v_i, $i \geq 0$. To avoid index-overload, we will not indicate the index i for these names. Thus let L, M, and R denote the left, middle, and right subtree of v_i. Further, let R_ℓ, R_m, and R_r denote the left, middle, and right subtrees of R and let L_ℓ, L_m, and L_r denote the left middle and right subtrees of L respectively. For ease of notation, we assume that all nodes on the heavy path have exactly three children where the non-heavy children may be present or be an empty subtree.

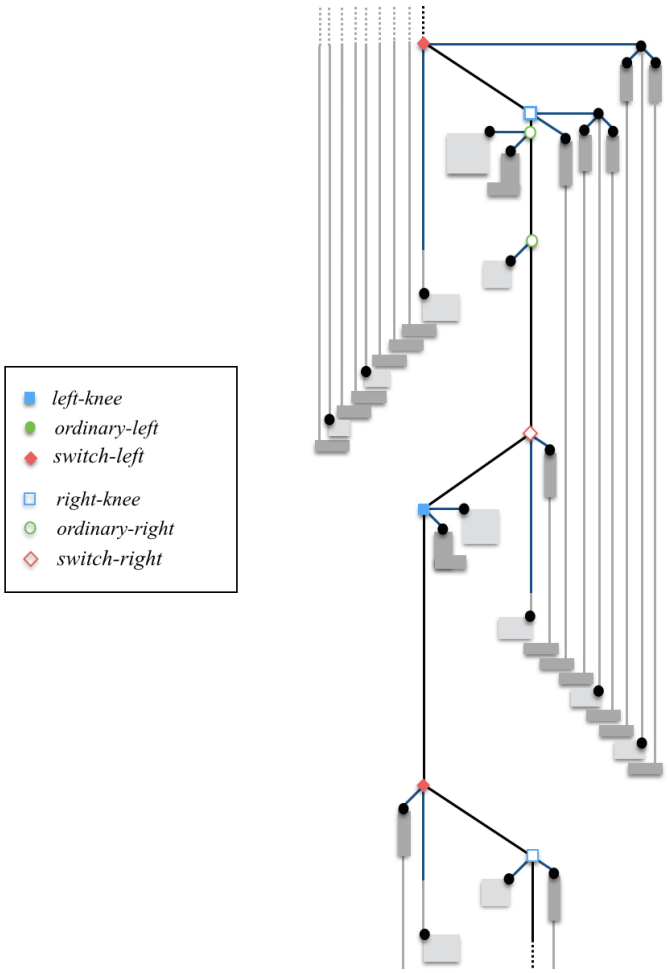


Figure 2.15: An octagonal drawing produced by the Overhang Algorithm where the heavy path is drawn using black edges.

2. The root case. Since we are constructing a drawing which has the root at the top-left corner, treatment of the root node (and possibly some of its descendants) requires a little more care. Let v_i be the root (initially $i = 0$).

   (a) If v_{i+1} is a left child, construct *overhang-left*(M), *overhang-left*(R_m), L-*left*(R_ℓ), and L-*left*(R_r). Repeat Step 2 with v_{i+1} as new root.

   (b) If v_{i+1} is a middle child, construct *overhang-left*(L), *overhang-right*(R_m), L-*right*(R_ℓ), and L-*right*(R_r). Assign v_{i+1} as a *right-knee* node. Move to Step 3.

(c) If v_{i+1} is a right child, construct *overhang-left*(L) and L-*left*(M). Then, if v_{i+1} has the leftmost child as a heavy child, assign v_{i+1} as an *ordinary-left* node. Else, assign v_{i+1} has a *switch-left* node. Move to Step 3.

3. Now consider vertex v_i for some $i > 0$ that is not treated by the root case. Omit this step if v_i is the leaf of the heavy path. Consider the labelling of v_i:

   (a) v_i is an *ordinary-left* node:
   We know that all non-heavy children are to the right of the heavy child.

      i. Construct *overhang-left*(R), and L-*left*(M).

      ii. If v_{i+2} does not exist or has no left sibling, then assign v_{i+1} as an *ordinary-left* node, else assign v_{i+1} as a *switch-left* node.

   (b) v_i is a *switch-left* node:
   We know that the heavy child v_{i+1} has a left sibling.

      i. Assign v_{i+1} as a *right-knee* node.

      ii. If v_{i+1} is a right child, construct *overhang-left*(M) and L-*left*(L).
      If v_{i+1} is a middle child, construct *overhang-left*(L), *overhang-right*(R_m), L-*right*(R_ℓ), and L-*right*(R_r).

   (c) v_i is a *left-knee* node:

      i. If v_{i+1} is a left child, construct *overhang-left*(R), and L-*left*(M).
      If v_{i+1} is a middle child, construct L-*left*(L), and *overhang-left*(R).
      If v_{i+1} is a right child, construct L-*left*(M), *overhang-left*(L_m), L-*left*(L_ℓ), and L-*left*(L_r).

      ii. If v_{i+2} does not exist or has no left sibling, then assign v_{i+1} as an *ordinary-left* node, else assign v_{i+1} as a *switch-left* node.

   (d) v_i is an *ordinary-right*, *right-knee* node, *switch-right* node: Do the same as in the cases, where v_i is an *ordinary-left*, *left-knee* node, or *switch-left* node, respectively with "left" exchanged with "right" (also in the naming of subtrees).

4. Let W_{overhang} be the maximum width among the Overhang drawings constructed in Step 3. Let W_{L} be the maximum neck width among the L-drawings constructed in Step 3 and let W_{f} be the maximum foot width among the L-drawings constructed in Step 3. Set $W_{\text{body}} = \max\{W_{\text{overhang}} + W_{\text{L}} + 2,\ W_{\text{f}} + 2\}$, then imagine the following grid structure: Six channels $C_1, \ldots, C_6$ of width W_{L} (with an extra column $\text{Col}_{4,3}$ to the left of C_3 and another column $\text{Col}_{6,5}$ to the left of C_5), followed by a single channel C_{body} of width W_{body}, followed by six channels $C'_1, \ldots, C'_6$ of width W_{L} (with an extra

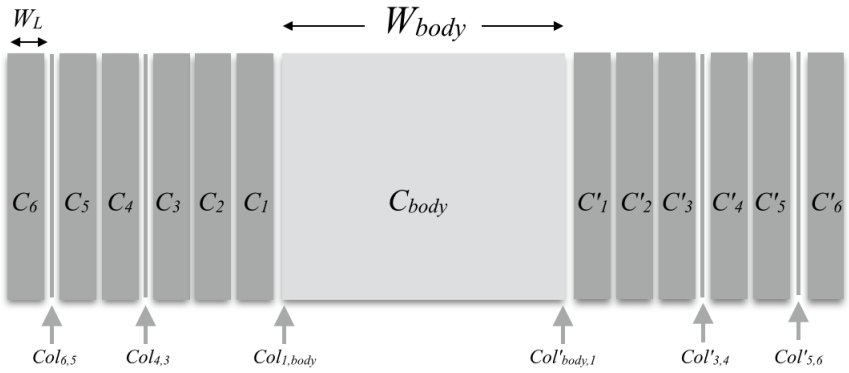


Figure 2.16: Grid structure used in the Overhang Algorithm for constructing octagonal drawings.

column $\text{Col}'_{3,4}$ to the right of C'_3 and another column $\text{Col}'_{5,6}$ to the right of C'_5.) See Figure 2.16. Let $\text{Col}_{1,\text{body}}$ and $\text{Col}'_{\text{body},1}$ be the leftmost column and rightmost column within the channel C_{body}. The idea is to draw the heavy path within the channel C_{body}, and place the neck of L-drawings in the channels C_i, C'_i where $i = 1, \ldots, 6$.

5. Initialize two stacks which can hold up to nine drawings each. We will use each stack to store *pending-left* drawings, and *pending-right* drawings. Each drawing in the stack will either be the foot of an L-drawing or an Overhang drawing and for each drawing in the stack, the x-coordinate (some reference point of the bounding box) is known at the time of push, however the y-coordinate (of the top row of the drawing) is determined at the time of pop.

In the following steps, we construct the Overhang drawing of T by drawing the heavy path and appropriately placing the subtree drawings around the heavy path. For a subtree T, let $D(T)$ denote the drawing of subtree T. Let H_i be the horizontal row corresponding to the node placed lowest in the drawing of T constructed so far.

6. The root phase. Let v_i be the root (initially $i = 0$). Place v_i one unit below H_i and in the leftmost column within channel C_6, and distinguish cases:

   (a) If v_{i+1} is a left child (see Figure 2.17a):
   Place the right child of v_i in $\text{Col}_{4,3}$ and horizontally aligned with v_i. Place the neck of $D(R_r)$ within channel C_3 using a diagonal edge and place the neck of

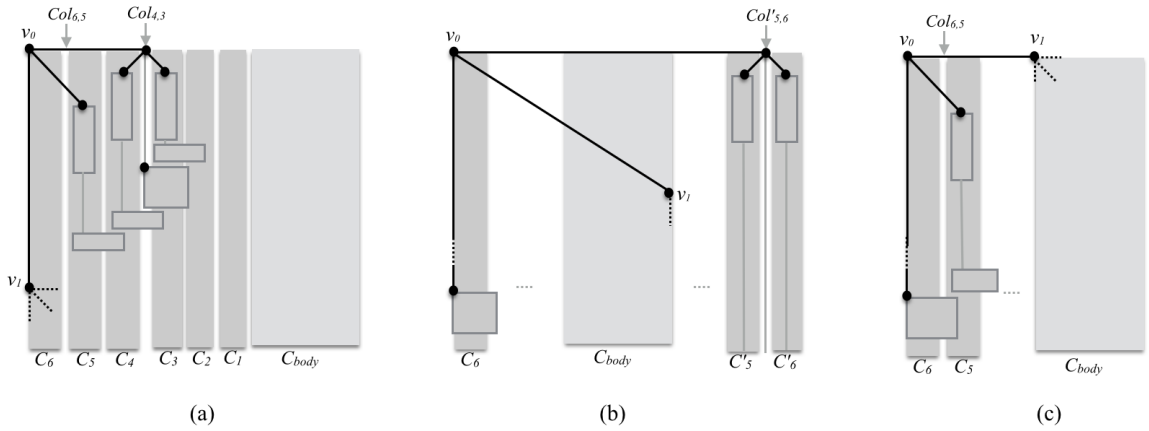


Figure 2.17: (a) v_1 is the left child. (b) v_1 is the middle child. (c) v_1 is the right child.

$D(R_\ell)$ within channel C_4 using a diagonal edge. Place the neck of $D(M)$ within channel C_5 using a diagonal edge. Place the foot of $D(R_r)$ immediately below the necks of $D(R_r)$ and $D(R_\ell)$. Place the root of R_m immediately below the foot of $D(R_r)$. Place the foot of $D(R_\ell)$ immediately below the bounding box of $D(R_m)$ and place the foot of $D(M)$ immediately below the neck of $D(M)$ and foot of $D(R_\ell)$. Repeat Step 7 with v_{i+1} as the new root case.

(b) If v_{i+1} is a middle child (see Figure 2.17b):
Place the right child of v_i in $\text{Col}'_{5,6}$ and horizontally aligned with v_i. Place the neck of $D(R_r)$ within channel C'_6 using a diagonal edge and place the neck of $D(R_\ell)$ within channel C'_5 using a diagonal edge. Push the foot of $D(R_r)$, then the foot of $D(R_\ell)$ onto the stack of pending-right drawings. Fix the x-position of the root of $D(L)$ to be the same as v_i and push $D(L)$ onto the stack of pending-left drawings. Move to Step 7.

(c) If v_{i+1} is a right child (see Figure 2.17c):
Place the neck of $D(M)$ within channel C_5 using a diagonal edge. Fix the x-position of the root of $D(L)$ to be the same as v_i. Push $D(L)$, then the foot of $D(M)$ onto the stack of pending-left drawings. Place v_{i+1} in column $C_{1,\text{body}}$ and horizontally aligned with v_i. Move to Step 7.

7. Now consider vertex v_i for some $i > 0$ that is not treated by the root phase. Depending on the label of v_i, do:

(7a) v_i is an *ordinary-left* node (see Figure 2.18a):

Place v_i one unit below H_i and vertically aligned with v_{i-1}. Place the neck of $D(M)$ one unit to the right of v_i using a diagonal edge and place $D(R)$ immediately to the right of the neck of $D(M)$ horizontally aligned with v_i. Place the foot of $D(M)$ immediately below the neck of $D(M)$ and $D(R)$.

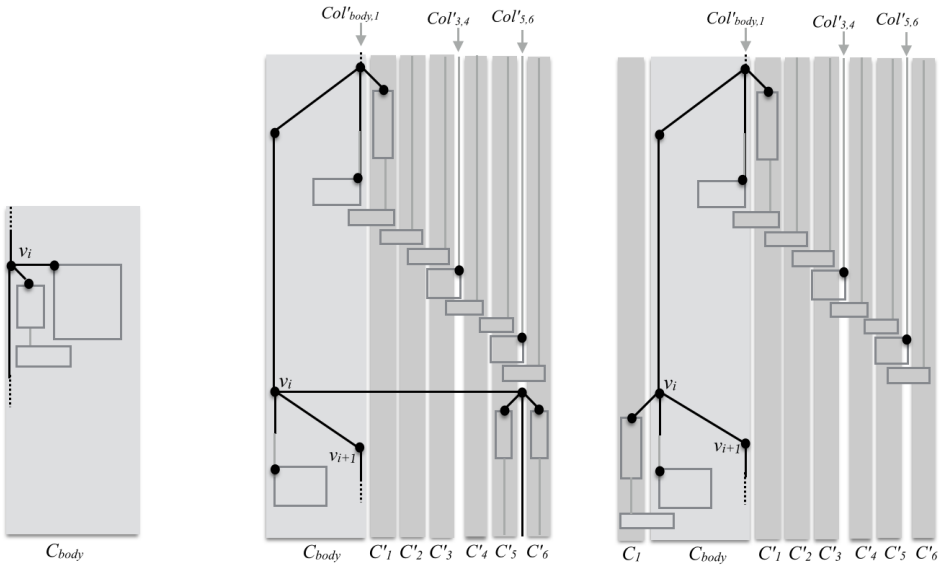


Figure 2.18: (a) v_i is an *ordinary-left* node and v_{i+1} is the left child. (b) v_i is a *switch-left* node and v_{i+1} is the middle child. (c) v_i is a *switch-left* node and v_{i+1} is the right child.

(7b) v_i is a *switch-left* node:
While the stack of pending-right drawings is not empty, pop one drawing and place it one unit below H_i. Note that the x-position of the drawing is known at the time of push and only the y-position need be decided at the time of pop. Once the stack of pending-right drawings is empty, place v_i in the same vertical channel as v_{i-1} and one unit below the new H_i. Place v_{i+1} in the column $\text{Col}'_{\text{body},1}$ using a diagonal edge from v_i.

i. v_{i+1} is a middle child (see Figure 2.18c):
Fix the x-position of the root of $D(L)$ to be the same as v_i and push $D(L)$ onto the stack of pending-left drawings. Place the right child of v_i in $\text{Col}'_{5,6}$ and horizontally aligned with v_i. Place the neck of $D(R_\ell)$ within channel C'_5 using a diagonal edge. Place the neck of $D(R_r)$ within channel C'_6 using

a diagonal edge. Push the foot of $D(R_r)$ onto the stack of pending-right drawings. Fix the x-position of the root of $D(R_m)$ to be the same as the right child of v_i and push $D(R_m)$ onto the stack of pending-right drawings. Push the foot of $D(R_\ell)$ onto the stack of pending-right drawings.

ii. v_{i+1} is a right child (see Figure 2.18b):
Place the neck of $D(L)$ within channel C_1 using a diagonal edge. Push the foot of $D(L)$ onto the stack of pending-left drawings. Fix the x-position of the root of $D(M)$ to be the same as v_i and push $D(M)$ onto the stack of pending-left drawings.

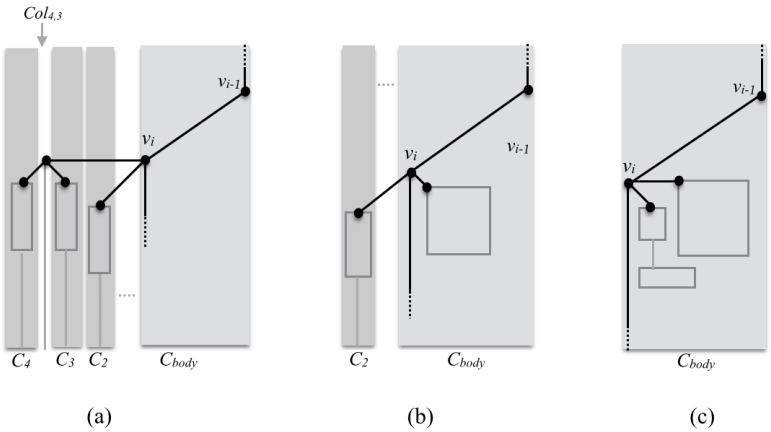


Figure 2.19: (a) v_i is a *left-knee* node, v_{i+1} is the right child (b) v_i is a *left-knee* node, v_{i+1} is the middle child. (c) v_i is a *left-knee* node, v_{i+1} is the left child.

(7c) v_i is a *left-knee* node:
Depending on whether v_{i+1} is the right, middle, or left child, do:

i. v_{i+1} is a right child (see Figure 2.19a):
Place the left child of v_i in $\text{Col}_{4,3}$ and horizontally aligned with v_i. Place the neck of $D(L_\ell)$ within channel C_4 using a diagonal edge and place the neck of $D(L_r)$ within channel C_3 using a diagonal edge. Place the neck of $D(M)$ within the channel C_2 using a diagonal edge. Fix the x-position of the root of $D(L_m)$ to be $\text{Col}_{4,3}$. Push the foot of $D(L_\ell)$, then $D(L_m)$, then the foot of $D(L_r)$, then the foot of $D(M)$ onto the stack of pending-left drawings.

ii. v_{i+1} is a middle child (see Figure 2.19b):

Place the neck of $D(L)$ within channel C_2 using a diagonal edge. Then push the foot of $D(L)$ onto the stack of pending-left drawings. Place $D(R)$ one unit below and to the right of v_i.

iii. v_{i+1} is a left child (see Figure 2.19c):
Place $D(M)$ and $D(R)$ in the same fashion as the case where v_i is an *ordinary-left* node.

(7d) v_i is an *ordinary-right*, *right-knee* node, *switch-right* node:
These cases are the same as the cases where v_i is an *ordinary-left*, *left-knee* node, or *switch-left* node respectively, with "left" exchanged with "right".

Note that we empty the stack of pending-right drawings every time we process a *switch-left* node (similarly, we empty the stack of pending-left drawings every time we process a *switch-right* node). We argue that we push at most nine drawings onto the stack of pending-right drawings while processing a sequence of labels beginning from a *switch-left* node and ending in the next *switch-left* node. In this sequence of labels, the labels that could push drawings on to the pending-right stack include, in order along the heavy path, the *switch-left*, the *right-knee*, and the *switch-right* node. The *switch-left* node may use channels C_1, C'_5, C'_6, and columns $\mathrm{Col}_{1,\mathrm{body}}$, $\mathrm{Col}'_{5,6}$. The *right-knee* node may use channels C'_2, C'_3, C'_4 and column $\mathrm{Col}'_{3,4}$. The *switch-right* node uses channel C'_1 and column $\mathrm{Col}'_{body,1}$. All in all, we push at most nine drawings onto the pending-right and each corresponds to a channel/column in order from right to left. Hence these drawings can be extracted and placed when handling the next *switch-left* node.

Theorem 3. *Any ordered ternary tree of size n admits an ideal octagonal drawing of width* $O(n^{0.68})$.

Proof. As seen from the grid structure of the Overhang Algorithm, the width of the drawing follows a sequence where $W_{\mathrm{overhang}}(1) = 1$ and

$$\begin{aligned} W_{\mathrm{overhang}}(n) &\leq 12W_{\mathrm{L}} + W_{\mathrm{body}} + 4 \\ &\leq 12W_{\mathrm{neck}}(n_1) + \max\left\{ \begin{matrix} W_{\mathrm{overhang}}(n_2) + W_{\mathrm{neck}}(n_1) + 2, \\ W_{\mathrm{foot}}(n_3) + 2 \end{matrix} \right\} + 4 \\ &\leq \max\left\{13W_{\mathrm{neck}}(n_1) + W_{\mathrm{overhang}}(n_2), \quad 12W_{\mathrm{neck}}(n_1) + W_{\mathrm{foot}}(n_3)\right\} + 6 \end{aligned}$$

for some n_1, n_2, n_3 where $n_1, n_2, n_3 \leq \lfloor \frac{n}{2} \rfloor$ (this is because n_1, n_2, and n_3 are sizes of subtrees not rooted at a node in the heavy path).

By Lemma 3, we know that $W_{\text{neck}}(n) \le cn^{0.68} - 1$ for some constant c. Now we will prove by induction that for all $n \ge 1$,

$$W_{\text{overhang}}(n) \le c_1 n^{0.68} \quad \text{where } c_1 = 29c \tag{2.5}$$

and

$$W_{\text{foot}}(n) \le c_2 n^{0.68} \quad \text{where } c_2 = 33c. \tag{2.6}$$

Base case: When $n = 1$, the left side of (2.5) is $W_{\text{overhang}}(1) = 1$ and the right side is $c_1(1^{0.68}) = c_1$.
When $n = 1$, the left side of (2.6) is $W_{\text{foot}}(1) = 1$ and the right side is $c_2(1^{0.68}) = c_2$.

Inductive step: Let $N > 1$ and assume (2.5) and (2.6) is true for $1 \le n < N$.
We refer to Observation 3 (page 32) and solve the following recurrence where $\sum_{i=1}^{6} n_i \le N$, and $n_5 \le (1 - c_0)N$.

$$\begin{aligned}
W_{\text{foot}}(N) &\le \Big(\sum_{i=1}^{4} W_{\text{neck}}(n_i)\Big) + \max\big\{W_{\text{foot}}(n_5),\ W_{\text{overhang}}(n_6)\big\} \\
&\le \Big(\sum_{i=1}^{4} (cn_i^{0.68} - 1)\Big) + \max\big\{c_2 n_5^{0.68},\ c_1 n_6^{0.68}\big\} \\
&\le \Big(\sum_{i=1}^{4} (cN^{0.68} - 1)\Big) + \max\big\{c_2((1-c_0)N)^{0.68},\ c_1 N^{0.68}\big\} \\
&= (4cN^{0.68} - 4) + \max\big\{c_2((1-c_0)N)^{0.68},\ c_1 N^{0.68}\big\} \\
&\le \max\big\{(4c + c_2(1-c_0)^{0.68})N^{0.68},\ (4c + c_1)N^{0.68}\big\} \\
&= \max\big\{(4c + 33c(1-c_0)^{0.68})N^{0.68},\ (4c + 29c)N^{0.68}\big\} \\
&\text{(Note that } \ 33c(1-c_0)^{0.68} \le 29c \ \text{ since } c_0 = 0.2783297.) \\
&\le 33cN^{0.68} \\
&\le c_2 N^{0.68}.
\end{aligned}$$

So by induction, $W_{\text{foot}}(n) \leq c_2 n^{0.68}$. Now we solve the recurrence for $W_{\text{overhang}}(N)$ where $n_1, n_2, n_3 \leq \lfloor \frac{N}{2} \rfloor$.

$$\begin{aligned}
W_{\text{overhang}}(N) &\leq \max\left\{13W_{\text{neck}}(n_1) + W_{\text{overhang}}(n_2), \quad 12W_{\text{neck}}(n_1) + W_{\text{foot}}(n_3)\right\} + 6 \\
&\leq \max\left\{ \begin{array}{l} 13c\Big(\frac{N}{2}\Big)^{0.68} - 13 + c_1\Big(\frac{N}{2}\Big)^{0.68}, \\ 12c\Big(\frac{N}{2}\Big)^{0.68} - 12 + c_2\Big(\frac{N}{2}\Big)^{0.68} \end{array} \right\} + 6 \\
&\leq \max\left\{(13c + c_1)\Big(\frac{N}{2}\Big)^{0.68}, \quad (12c + c_2)\Big(\frac{N}{2}\Big)^{0.68}\right\} \\
&= \max\left\{(13c + 29c)\Big(\frac{N}{2}\Big)^{0.68}, \quad (12c + 33c)\Big(\frac{N}{2}\Big)^{0.68}\right\} \\
&= \max\left\{\Big(\frac{42c}{2^{0.68}}\Big)N^{0.68}, \quad \Big(\frac{45c}{2^{0.68}}\Big)N^{0.68}\right\} \\
&\leq 29cN^{0.68} \\
&= c_1 N^{0.68}.
\end{aligned}$$

Therefore by induction, $W_{\text{overhang}}(n) \leq c_1 n^{0.68} \in O(n^{0.68})$. □

Since the algorithm uses a constant number of operations per node, the running time for the Overhang Algorithm is $O(n)$.

Chapter 3

Width Lower Bounds

In this chapter, we present asymptotic lower bounds on the width of octagonal drawings of complete ternary trees. In our approach, we consider drawings of complete ternary trees and show the width lower bound as a function of the number of leaves in the tree. (In this chapter, n will denote the number of leaves, rather than the number of nodes, because some formulas then simplify, e.g., the height is $\log_3 n$. Since the number of leaves is proportional to the number of nodes in a complete ternary tree, this assumption does not affect asymptotic bounds.) The results are derived through a series of improvements where we define a rectangle R with respect to the given drawing of the tree, obtain a lower bound on the minimum number of nodes that must be drawn inside R, then use the Pigeonhole principle to derive a lower bound on the minimum width of R which in turn is a lower bound on the minimum width of the drawing. In each iteration of the proof, we improve the lower bound by redefining the rectangle R and obtaining asymptotically stronger bounds on the number of nodes that must lie inside R. This comes at the cost of smaller constant factors, making the latter bounds to be of mostly theoretical interest.

Throughout this chapter, let $D(T)$ be an ideal octagonal drawing of a complete ternary tree T with n leaves and height $\ell = \log_3 n$.

3.1 Method 0

Since we measure the area of a drawing by the number of grid points occupied by the bounding box of the drawing, we define w to be the maximum integer such that some two nodes are drawn $w - 1$ columns apart (say, after translation all nodes of $D(T)$ have x-coordinate in $\{1, \ldots, w\}$).

Definition 6. *For $i = 1, \ldots, \ell$, let R_i be the rectangle consisting of the topmost $w \cdot i$ rows of the bounding box of $D(T)$.*

Definition 7. *Let $n_0 = 1$ and for $i = 1, \ldots, \ell$, let n_i denote the number nodes at tree depth i that appear in R_i (the root has tree depth 0).*

Lemma 5. $n_i \geq 2n_{i-1}$ *for $i = 1, \ldots, \ell$.*

Proof. For $i \geq 2$, consider some node v_{i-1} that has tree depth $i-1$ and is drawn inside R_{i-1}. Since $D(T)$ is an ideal octagonal drawing, at least two children of v_{i-1} are drawn with a horizontal or diagonal edge. Note that any horizontal or diagonal edge spans a vertical distance of at most $w-1$. So node v_{i-1} can have at most one child (namely the vertical child) which has vertical distance greater than $w-1$ from v_{i-1}. Since the height of R_i is w units more than the height of R_{i-1} [1], at least two children of v_{i-1} are in R_i, and the bound follows. For $i = 1$, we argue similarly that $n_1 \geq 2$ since the root is on the topmost row in R_1 and the root must have at least two children in R_1. □

Theorem 4. *A complete ternary tree T with n leaves requires $\sqrt{\frac{n^{\log_3 2}}{\log_3 n}}$ width in any ideal octagonal drawing.*

Proof. Since there are $n_\ell \geq 2^\ell$ nodes in R_ℓ, the number of grid points in R_ℓ must be at least 2^ℓ. Since R_ℓ has $w(w\ell)$ grid points and $\ell = \log_3 n$, the following is true:

$$w^2 \log_3 n \geq 2^{\log_3 n} = n^{\log_3 2}.$$

We solve for w and achieve a lower bound on the width:

$$w \geq \sqrt{\frac{n^{\log_3 2}}{\log_3 n}} \in \Omega(n^{0.315}).$$

□

3.2 Method 1

Define again w to be the maximum integer such that some two nodes are drawn $w-1$ columns apart. We also use the same rectangle definition as before, i.e., R_i has width w and height $w \cdot i$. The main difference from Method 0 is that we find a second lower bound on n_i, and use it to obtain better bounds. We need a definition.

[1] It would have been enough to use height $i \cdot (w-1) + 1$ for rectangle R_i for this bound to hold, but for ease of notation we use height $i \cdot w$.

Definition 8. *Fix $i \in \{1, \ldots, \ell\}$ and a node v_{i-1} at tree depth $i-1$. We say that v_{i-1} is* ***good*** *if*

i. all three children of v_{i-1} are in R_i and

ii. either $i = 1$ or v_{i-1} is in R_{i-1}.

Else v_{i-1} is ***bad****.*

Let v_{i-1} be a node at depth $i-1$ that is either the root or inside R_{i-1}. Assume that v_{i-1} is bad. Then at least one child of v_{i-1} is outside R_i. Since R_i is w rows taller than R_{i-1}, this is possible only if this child is a vertical child of v_{i-1}, and the edge from v_{i-1} to this child intersects the bottom of R_{i-1}. This allows us to develop a better bound on n_i as follows:

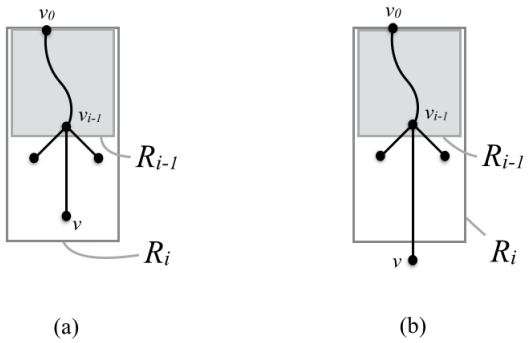


Figure 3.1: (a) Node v_{i-1} is good since all three children are drawn inside R_i and v_{i-1} is drawn inside R_{i-1}. (b) Node v_{i-1} is bad since it has a vertical child v drawn outside R_i.

Lemma 6. *$n_i \geq 3n_{i-1} - w$ for $i = 1, \ldots, \ell$.*

Proof. For $i \geq 2$, consider a node v_{i-1} at depth $i-1$ and drawn in R_{i-1}. If v_{i-1} is a good node then all three of its children are in R_i. If v_{i-1} is a bad node, then (as in the proof of Lemma 5) at least two of its children are in R_i. Hence, $n_i \geq 3n_{i-1} - n_b$ where n_b is the number of bad nodes at tree depth $i-1$ drawn inside R_{i-1}. Every such bad node uses up one grid point on the bottom side of R_{i-1} for the edge to its vertical child. Since there are w such points, we have $n_b \leq w$ and so $n_i \geq 3n_{i-1} - w$. For $i = 1$, we argue similarly that $n_1 \geq 3 - w$ since the root is on the topmost row in R_1 and the root can either be a good node and have three children in R_1, or be a bad node and have only two children in R_1, hence $n_1 \geq 2 = 3n_0 - 1 \geq 3 - w$. □

Lemma 7. *Let n_i be a sequence that satisfies the following recurrence:*

i. $n_0 = 1$

ii. $n_i \geq 2n_{i-1}$ *for* $i = 1, \ldots, \ell$

iii. $n_i \geq 3n_{i-1} - w$ *for* $i = 1, \ldots, \ell$.

Then $n_\ell \geq \frac{1}{6} n w^{1-\log_2 3}$.

Proof. Define $k = \lceil \log_2 w \rceil$. Using i. and ii., we have $n_k \geq 2^k \geq 2^{\log_2 w} = w$. Therefore, (with iii.) $n_{k+1} \geq 3w - w, n_{k+2} \geq 3^2 w - 3w - w$ and generally $n_{k+i} \geq 3^i w - (\sum_{m=0}^{i-1} 3^m)w = 3^i w - \frac{3^i - 1}{2} w \geq \frac{3^i}{2} w$. Using $i = \ell - k$, we get $n_\ell \geq \frac{w}{2} 3^{\ell-k} = \frac{w}{2} \frac{3^{\log_3 n}}{3^{\lceil \log_2 w \rceil}} \geq \frac{w}{2} \frac{n}{3^{\log_2 w + 1}} = \frac{n}{6} \frac{w}{w^{\log_2 3}}$ as desired. □

Theorem 5. *A complete ternary tree T with n leaves requires $\Omega\Big(\Big(\frac{n}{\log_3 n}\Big)^{\frac{\log_3 2}{\log_3 2 + 1}}\Big)$ width in any ideal octagonal drawing.*

Proof. Since the number of grid points in R_ℓ must be no smaller than n_ℓ, the following is true:

$$w(w \log_3 n) \geq \frac{1}{6} n w^{1-\log_2 3}, \text{ hence } \; w^{1+\log_2 3} \geq \frac{1}{6} \frac{n}{\log_3 n}.$$

Since $1 + \log_2 3 = \frac{\log_3 2 + 1}{\log_3 2}$, this gives

$$w \geq \left(\frac{1}{6} \frac{n}{\log_3 n}\right)^{\frac{\log_3 2}{\log_3 2 + 1}} \in \Omega\Big(\Big(\frac{n}{\log_3 n}\Big)^{\frac{\log_3 2}{\log_3 2 + 1}}\Big) \subseteq \Omega\Big(n^{0.386}\Big).$$

□

The constant hidden in the asymptotic notation is $\frac{1}{6}$ which makes this bound better than the one of Theorem 4 when $n \geq 22$.

3.3 Method 2

For our next bound, we need a much more complicated definition of rectangles, and the idea of "trapped subtrees". We state the main result first.

Theorem 6. *A complete ternary tree T with n leaves requires* $\Omega\left(\left(\frac{n}{\log_3 n}\right)^{\frac{1}{1+3\log_{10} 3}}\right) \subseteq \Omega\left(n^{0.411}\right)$ *width in any ideal octagonal drawing.*

The rest of this section gives the proof of Theorem 6, which is by induction on n. Let $w = c_0(\frac{n}{\log_3 n})^\alpha$ where $\alpha = \frac{1}{1+3\log_{10} 3} > 0.411$ and $c_0 < 0.000279$. So in contrast to the previous proofs, w is not necessarily exactly the width of $D(T)$. Instead we show a stronger statement: there exist two grid points within rows $1, \ldots, \lceil c_1 w\ell \rceil$ and with horizontal distance at least $w - 1$ that are both occupied by (a vertex or an edge of) $D(T)$. Here, c_1 is a suitable constant ($c_1 = 4$ should do).

Base case: We prove the base case for all n with $\ell = \log_3 n < 12$ and all n with $c_1 w\ell = c_1 c_0(\frac{n}{\log_3 n})^\alpha \log_3 n < 1$, since we need these assumptions on n in the induction step. We need to show that within $\lceil c_1 w\ell \rceil$ rows we use width at least w. This holds since the root needs a column and $w \leq 1$ for this range of n.

Inductive step: Assume for contradiction $(**)$ that $D(T)$ has width less than w for the first $\lceil c_1 w\ell \rceil$ grid rows.

We use a different rectangle definition than before where the height is a more complicated function B_i.

Definition 9. *For $i = 1, \ldots, \ell$, let*

$$B_i = \sum_{j=1}^{i} b_j$$

where

$$b_j = \begin{cases} 3w + c_1 w\ell(\frac{1}{3^j})^\alpha & \text{if } j > 11 \\ w & \text{if } j \leq 11. \end{cases}$$

Lemma 8. *Assume that $\ell \geq 12$. Then B_ℓ is at most $\lceil c_1 w\ell \rceil$.*

Proof. In essence, this holds because the b_j-terms form a geometric series that can be upper-bounded. More precisely, we have

$$\begin{aligned}
B_\ell &= \sum_{j=1}^{\ell} b_j \\
&= 11w + 3(\ell - 11)w + \Big[\sum_{j=12}^{\ell} c_1 w\ell(\frac{1}{3^j})^\alpha\Big] \\
&\leq 3w\ell + c_1 w\ell \sum_{j=12}^{\infty} \frac{1}{(3^j)^\alpha} \\
&= 3w\ell + c_1 w\ell\Big(\frac{1}{3^{12\alpha}}\Big)\Big(\frac{1}{1-\frac{1}{3^\alpha}}\Big) \\
&= w\ell\Big[3 + c_1\Big(\frac{1}{3^{12\alpha}}\Big)\Big(\frac{3^\alpha}{3^\alpha - 1}\Big)\Big] \\
&\leq c_1 w\ell \\
&\leq \lceil c_1 w\ell \rceil.
\end{aligned}$$

The next-to-last inequality holds since for $c_1 \geq 3.04$, we have

$$3 + c_1\Big(\frac{1}{3^{12\alpha}}\Big)\Big(\frac{3^\alpha}{3^\alpha - 1}\Big) \approx 3 + c_1(0.00443)(2.752) \leq c_1.$$

□

Definition 10. *For $i = 1, \ldots, \ell$, let R'_i be the rectangle that is the bounding box of the first B_i rows of $D(T)$.*

By assumption $(**)$ and Lemma 8, rectangle R'_i has width less than w.

Definition 11. *Let $n_0 = 1$ and for $i = 1, \ldots, \ell$, let n_i denote the number of nodes at tree depth i that appear in R'_i.*

We use the same definition for good and bad nodes as in the first method (Definition 8), applied for rectangles R'_i in place of R_i.

Definition 12. *For $i \geq 14$, let v_{i-3} be a node at tree depth $i-3$ that is inside R'_{i-3}. Then a subtree rooted at some descendant v of v_{i-3} is **trapped** by v_{i-3} if all of the following are satisfied:*

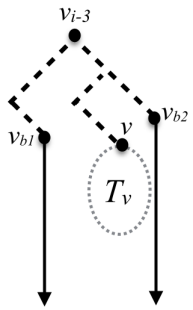


Figure 3.2: By the definition of trapped, the subtree rooted at some descendant v of v_{i-3} is trapped by v_{i-3}.

1. *v has tree depth $i, i-1$, or $i-2$.*
2. *There exists a bad node v_{b1} at tree depth $i-1, i-2$, or $i-3$ with a smaller x-coordinate than v and greater or equal y-coordinate than v.*
3. *There exists a bad node v_{b2} at tree depth $i-1, i-2$, or $i-3$ with a larger x-coordinate than v and greater or equal y-coordinate than v.*
4. *Nodes v_{b1} and v_{b2} are descendants of v_{i-3}.*
5. *The path from v_{i-3} to v does not include any vertical edges.*

For ease of wording, we may say 'node v is trapped by v_{i-3}' as an equivalent statement to 'the subtree rooted at v is trapped by v_{i-3}'.

Figure 3.2 illustrates these concepts. In essence, because v_{b1} and v_{b2} are bad, they have vertical edges that reach "far below". By choice of the coordinates, the drawing of T_v is "between" these vertical edges, at least for some parts and because we know a lower bound on the width of $D(T_v)$, this will allow us to develop a tighter bound on n_i. The following lemmas discuss properties of trapped nodes, and then make the above argument more precise.

Observation 4. *For $i \geq 14$, consider some node v_{i-3} that is at tree depth $i-3$ in T and is drawn inside R'_{i-3}. Let v be a descendant of v_{i-3} that is trapped by v_{i-3}. Then v is inside R'_{i-2} and has vertical distance at most $3w$ from v_{i-3}. Moreover, the vertical edges of the nodes v_{b1} and v_{b2} that trap v intersect the bottom of R'_{i-2}.*

Proof. We know that v_{i-3} is inside R'_{i-3} which by $(**)$ has width at most w. Since the path from v_{i-3} to v does not include any vertical edges by Condition 5 of "trapped", and the path length from v_{i-3} to v is at most 3 by Condition 1, v has vertical distance at most $3w$ from v_{i-3}. By definition of b_{i-2} therefore v is inside R'_{i-2}. By Conditions 2 and 3, vertices v_{b1} and v_{b2} have y-coordinates no smaller than that of v, and so are inside R'_{i-2}. But they are bad and at depth $i-3$ or more, so their vertical children are outside R'_{i-2} by definition of "bad". □

Lemma 9. *For $i \geq 14$, let v_{i-3} be a node at tree depth $i-3$ that is drawn inside R'_{i-3} and let T_v be a subtree rooted at node v that is trapped by v_{i-3}. Then node v is drawn at least $\lceil c_0 c_1 |T_v|^{\alpha} (\log_3 |T_v|)^{1-\alpha} \rceil$ rows above the bottom of R'_{i-2}.*

Proof. By Observation 4, v has vertical distance at most $3w$ from v_{i-3}. Rectangle R'_{i-2} has b_{i-2} units more height than R'_{i-3} where $b_{i-2} = 3w + c_1 w \ell(\frac{1}{3^{i-2}})^{\alpha}$, so v is at least $c_1 w \ell(\frac{1}{3^{i-2}})^{\alpha}$ above the bottom of R'_{i-2}. The largest $|T_v|$ is achieved when v is a child of v_{i-3}, therefore $|T_v| \leq \frac{n}{3^{i-2}}$. Since $c_1 w \ell(\frac{1}{3^{i-2}})^{\alpha} = c_0 c_1 (\frac{n}{3^{i-2}})^{\alpha} (\log_3 n)^{1-\alpha} \geq c_0 c_1 |T_v|^{\alpha} (\log_3 |T_v|)^{1-\alpha}$ and since both v and the bottom of R'_{i-2} are at integer coordinates, v is at least $\lceil c_0 c_1 |T_v|^{\alpha} (\log_3 |T_v|)^{1-\alpha} \rceil$ rows above the bottom of R'_{i-2}. □

Definition 13. *We say node v_j is a* non-vertical child/grandchild *of node v_i if v_j is a child/grandchild of v_i and the path from v_i to v_j does not include any vertically drawn edges.*

Lemma 10. *For $i \geq 14$, let v_{i-3} be a node at tree depth $i-3$ that is drawn inside R'_{i-3}. Let n'_i denote the number of descendants of v_{i-3} at tree depth i that appear in R'_i. If $n'_i < 10$, then there exists a descendant of v_{i-3} that is trapped by v_{i-3}.*

Proof. By Observation 4, the four non-vertical grandchildren of v_{i-3} are inside R'_{i-2}. Since there are at least two non-vertical children for each grandchild and $R'_i \supset R'_{i-2}$, there are at least eight nodes that contribute to n'_i (these nodes are shown as diamond shapes in Figure 3.3). We now present a case analysis on the drawing configurations of the subtree rooted at v_{i-3} to find either two more nodes that contribute to n'_i, or to find a trapped descendant:

Case 1. v_{i-3} is a good node:
Since v_{i-3} is a good node, the vertical child of v_{i-3} is drawn inside R'_{i-2}, which guarantees that the four non-vertical grandchildren of the vertical child of v_{i-3} are in R'_{i-1} by Observation 4 (these nodes are shown as circular shapes in Figure 3.3). Since $R'_i \supset R'_{i-1}$, we can conclude $n'_i \geq 12$.

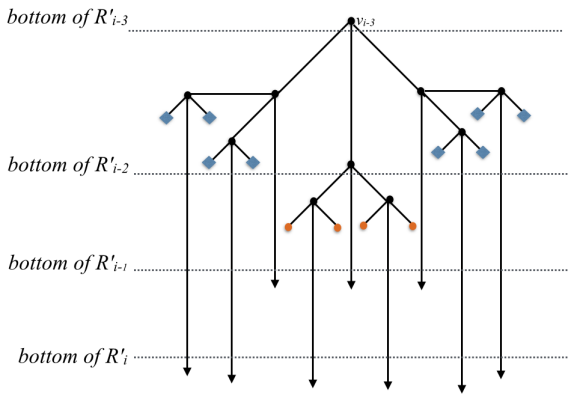


Figure 3.3: Case 1.

Note that Case 1 applies whenever all children of v_{i-3} are non-vertical, so for all following cases we know that v_{i-3} has a vertical child.

Case 2. v_{i-3} is a bad node and at least one of the two non-vertical children of v_{i-3} is a good node:
Let v_{i-2} denote the non-vertical child of v_{i-3} which is a good node and let v'_{i-2} denote another non-vertical child of v_{i-3}. Since v_{i-3} is drawn inside R'_{i-3}, v'_{i-2} has at least four grandchildren in $R'_{i-2} \subset R'_i$ by Observation 4 (these nodes are shown as diamond shapes in Figure 3.4). Moreover since v_{i-2} is a good node, v_{i-2} must have at least six grandchildren in $R'_{i-1} \subset R'_i$ (these nodes are shown as circular shapes in Figure 3.4). Therefore, we can conclude that $n'_i \geq 10$.

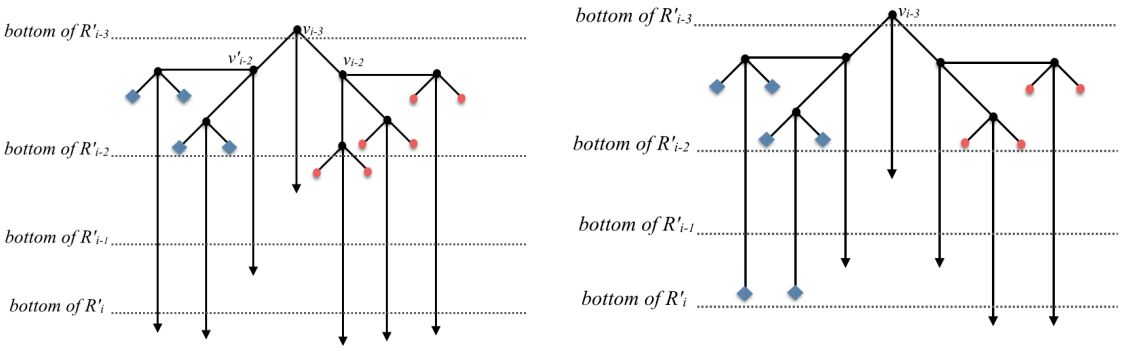
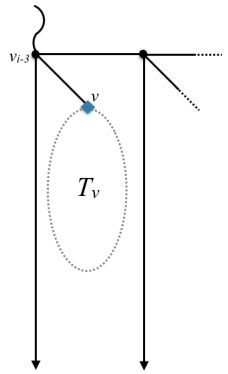


Figure 3.4: (Left) Case 2. (Right) Case 3.

Case 3. v_{i-3} is a bad node, and the non-vertical children of v_{i-3} are bad nodes, and at least two non-vertical grandchildren of v_{i-3} are good nodes:
Since v_{i-3} is drawn inside R'_{i-3}, each non-vertical grandchild of v_{i-3} that is a good node has three children in R'_i and therefore each such grandchild contributes three nodes to n'_i (these nodes are shown in as diamond shapes in Figure 3.4). Moreover, each non-vertical grandchild of v_{i-3} that is a bad node has at least two non-vertical children which are in $R'_{i-2} \subset R'_i$ by Observation 4, so it contributes at least two nodes to n'_i (these nodes are shown as circular shapes in Figure 3.4). Since there are at least two non-vertical grandchildren of v_{i-3} that are good nodes and at least four non-vertical grandchildren, we can conclude that $n'_i \geq 10$.

Case 4. v_{i-3} is a bad node, and all non-vertical children of v_{i-3} are bad nodes, and at most one non-vertical grandchild of v_{i-3} is a good node:
We consider all possible drawing configurations where there is at most one non-vertical grandchild of v_{i-3} that is a good node and show the existence of a node v where the subtree rooted at v is trapped by v_{i-3}. We have the following subcases:



(a)

Figure 3.5: Case 4(a) is shown with the diamond shaped node denoting node v. (We show the existence of node v where the subtree rooted at v is trapped by v_{i-3}.)

a) v_{i-3} is drawn with the left child as the vertical child. See Figure 3.5(a). Define v to be the middle child of v_{i-3}. We claim that v is trapped and verify the conditions as follows:

1. v has tree depth $i-2$.
2. Use $v_{b1} := v_{i-3}$. By case assumption v_{i-3} has smaller x-coordinate than its middle child v. Since we have an upward drawing, v_{i-3} has y-coordinate greater than or equal to v.
3. Set v_{b2} to be the right child of v_{i-3}. By case assumption (v_{i-3}, v_{b2}) is drawn horizontally, thus v_{b2} has y-coordinate greater than or equal to v. Also by case assumption v_{b2} is bad, so its vertical child is outside R'_{i-1}. Thus the vertical edge from v_{b2} must bypass v (which is in R'_{i-2}), so the x-coordinate of v_{b2} is bigger than v.
4. Holds by choice of v_{b1} and v_{b2}.
5. Holds by our assumption.

(The case where v_{i-3} is drawn with the right child as the vertical child is symmetric.)

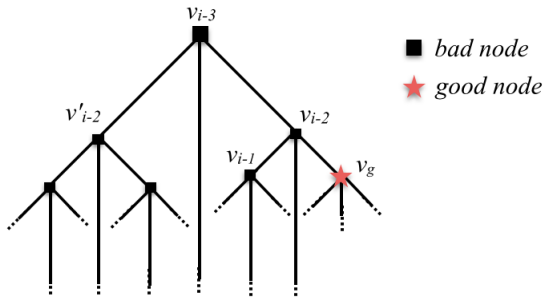


Figure 3.6: Labelling of good and bad nodes for describing subcases of Case 4. We do not necessarily have children appear in the order labelled above, e.g., v_g could be the leftmost child.

b) v_{i-3} is drawn with the middle child as the vertical child. Let v_g be the non-vertical grandchild of v_{i-3} that is good (set v_g to be an arbitrary grandchild if there is none that is good). Let v_{i-2} be the parent of v_g and let v_{i-1} be a sibling of v_g that is a non-vertical child of v_{i-2}. By assumption v_{i-1} is bad. Let v'_{i-2} denote a non-vertical sibling of v_{i-2}. See Figure 3.6. We may assume that v'_{i-2} is the left child and v_{i-2} is the right child of v_{i-3}, the other case is symmetric. In this subcase we must consider drawing configurations of the two non-vertical children of v'_{i-2}. Let v'_{i-1} be the leftmost non-vertical child of v'_{i-2}; by case assumption and choice of v'_{i-2}, node v'_{i-1} is bad.

b1) v'_{i-2} is drawn with the right child as the vertical child. See Figure 3.7(b1). Set v to be the middle child of v'_{i-2} and verify that it is trapped using $v_{b1} := v'_{i-1}$, $v_{b2} := v'_{i-2}$. (Details are left to the reader.)

b2) v'_{i-2} is drawn with the middle child as the vertical child. See Figure 3.7(b2). Set v to be the right child of v'_{i-2} and verify that it is trapped using $v_{b1} := v'_{i-2}$, $v_{b2} := v_{i-3}$.

b3) v'_{i-2} is drawn with the left child as the vertical child. See Figure 3.7(b3). Set v to be the middle child of v'_{i-2} and verify that it is trapped using $v_{b1} := v'_{i-2}$, $v_{b2} := v_{i-3}$.

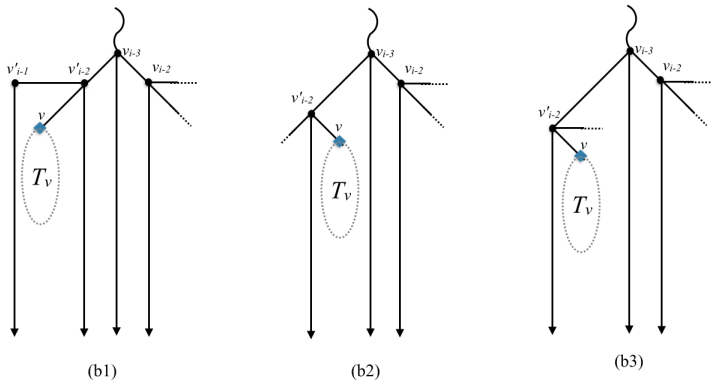


Figure 3.7: Subcases of case 4(b) are shown.

By case analysis, we have shown the existence of a node v in R'_{i-2} where the subtree rooted at v is trapped. Note that in each subcase, we only show a single node v such that subtree rooted at v is trapped. However, in most cases there can be many such vertices.

□

Lemma 11. $n_i \geq 10n_{i-3} - 2(3^{i\alpha})$ *for* $i = 14, \ldots, \ell$.

Proof. For $i = 14, \ldots, \ell$, consider a node v_{i-3} at depth $i-3$ and drawn in R'_{i-3}. By Lemma 10, if the number of descendants of v_{i-3} at tree depth i that are located in R'_i is less than 10, then there exists a descendant of v_{i-3} that is trapped by v_{i-3}. Further, Lemma 5 implies

that any v_{i-3} has at least eight descendants with tree depth i inside R'_i. Therefore we can deduce the following.

$$n_i \geq 10n_{i-3} - 2\Big(\#\text{ of subtrees trapped by a node at level } i-3\Big)$$

Since the vertical lines that separate trapped subtrees all intersect the bottom of R'_{i-2}, trapped subtrees are disjoint and the maximum number of trapped subtrees in R'_{i-2} can be bounded by the width of the R'_{i-2} which is at most w by $(**)$, divided by the minimum width of a trapped subtree. We can bound the width of each trapped subtree by induction. Let v be the root of a trapped subtree; we know that v is at level $i-2, i-1$, or i. Therefore T_v contains at least $\frac{n}{3^i}$ nodes. By Lemma 9, node v is drawn at least $\lceil c_0 c_1 |T_v|^\alpha (\log_3 |T_v|)^{1-\alpha} \rceil$ grid rows above the bottom of R'_{i-2}. This is at least $c_1 w_v \ell_v$, where $w_v := c_0 \Big(\frac{|T_v|}{\log_3 |T_v|}\Big)^\alpha$ and $\ell_v := \log_3(|T_v|)$ are the width requirement and height of subtree T_v. We know by induction that T_v requires width at least $c_0 \Big(\frac{|T_v|}{\log_3 |T_v|}\Big)^\alpha \geq c_0 \Big(\frac{n/3^i}{\log_3(n/3^i)}\Big)^\alpha$ within the first $c_1 w_v \ell_v$ rows therefore it requires this width between the two vertical lines that trap it until the bottom of R'_{i-2}. Thus,

$$\begin{aligned}
n_i &\geq 10n_{i-3} - 2\left(\frac{w}{c_0\Big(\frac{n/3^i}{\log_3(n/3^i)}\Big)^\alpha}\right) \\
&\geq 10n_{i-3} - \frac{2\Big(\frac{n}{\log_3 n}\Big)^\alpha}{\Big(\frac{n/3^i}{\log_3(n/3^i)}\Big)^\alpha} \\
&\geq 10n_{i-3} - 2(3^{i\alpha}) && \text{since } \log_3(n/3^i) \leq \log_3 n.
\end{aligned}$$

□

Lemma 12. *Let n_i be a sequence that satisfies the following recurrence:*

i. $n_0 = 1$

ii. $n_i \geq 2n_{i-1}$ *for* $i = 1, \ldots, \ell$

iii. $n_i \geq 10n_{i-3} - 2(3^{i\alpha})$ *for* $i = 14, \ldots, \ell$

iv. $n_i \geq 3n_{i-1} - w$ *for* $i = 1, \ldots, \ell$

where $\ell = \log_3 n$. Then for n sufficiently large, $n_\ell \geq c_3 n w^{1-3\log_{10} 3}$ for some constant c_3.

Proof. Define $k = 11$. Then by using i. and ii., we have $n_{k-2} \geq 2^{k-2} \geq 2(3^{\alpha(k+1)})$ since $2^9 > 452 = \lceil 2 \cdot 3^{(0.411 \cdot 12)} \rceil \geq 2 \cdot 3^{(0.411 \cdot 12)} \geq 2 \cdot 3^{\alpha(k+1)}$. Therefore, (with iii.)

$$n_{k+1} \geq 10 \cdot 2 \cdot 3^{\alpha(k+1)} - 2 \cdot 3^{\alpha(k+1)}$$
$$n_{k+4} \geq 10^2 \cdot 2 \cdot 3^{\alpha(k+1)} - 10 \cdot 2 \cdot 3^{\alpha(k+1)} - 2 \cdot 3^{\alpha(k+4)}$$

and generally for $i - 1$ a multiple of 3:

$$n_{k+i} \geq 2 \cdot 3^{\alpha(k+1)} \cdot 10^{\frac{i+2}{3}} - 2 \cdot 3^{\alpha(k+i)} \left[\sum_{m=0}^{\frac{i-1}{3}} \left(\frac{10}{3^{3\alpha}} \right)^m \right]$$

We have

$$\sum_{m=0}^{\frac{i-1}{3}} \left(\frac{10}{3^{3\alpha}} \right)^m = \frac{\left(\frac{10}{3^{3\alpha}} \right)^{\frac{i-1}{3}+1} - 1}{\frac{10}{3^{3\alpha}} - 1} \geq \frac{10^{\frac{i+2}{3}} \cdot 3^{-(i+2)\alpha}}{\frac{10}{3^{3\alpha}} - 1}$$

and therefore

$$\begin{aligned} n_{k+i} &\geq 2 \cdot 3^{\alpha(k+1)} \cdot 10^{\frac{i+2}{3}} - 2 \cdot 3^{\alpha(k+i)} \cdot 3^{-(i+2)\alpha} \cdot 10^{\frac{i}{3}} \left(\frac{10^{\frac{2}{3}}}{\frac{10}{3^{3\alpha}} - 1} \right) \\ &\geq 2 \cdot 3^{\alpha(k+1)} \cdot 10^{\frac{i}{3}} \left[10^{\frac{2}{3}} - 3^{-3\alpha} \left(\frac{10^{\frac{2}{3}}}{\frac{10}{3^{3\alpha}} - 1} \right) \right]. \end{aligned}$$

Define $c_2 := 2 \cdot 3^{\alpha(k+1)} \left[10^{\frac{2}{3}} - 3^{-3\alpha} \left(\frac{10^{\frac{2}{3}}}{\frac{10}{3^{3\alpha}} - 1} \right) \right] \approx 1751$, we hence have $n_{k+i} \geq c_2 10^{\frac{i}{3}}$. (†)
Define t to be the smallest integer such that $t \geq 3 \log_{10} w + k - 3 \log_{10} c_2$ and $t - k - 1$ is a multiple of 3. We have

$$t \leq 3 \log_{10} w + k + 3 - 3 \log_{10} c_2.$$

Using $i = t - k$ in (†), we get

$$n_t \geq c_2 10^{\frac{3 \log_{10} w - 3 \log_{10} c_2}{3}} = c_2 \frac{10^{\log_{10} w}}{10^{\log_{10} c_2}} = w.$$

Therefore (with iv), $n_{t+1} \geq 3w - w$, $n_{i+2} \geq 3^2 w - 3w - w$ and generally $n_{t+i} \geq 3^i w - (\sum_{m=0}^{i-1} 3^i) w = 3^i w - \frac{3^i - 1}{2} w \geq \frac{3^i}{2} w$. Using $i = \ell - t$, we get

$$n_\ell \geq \frac{w}{2} 3^{\ell - t} \geq \frac{w}{2} \frac{3^{\log_3 n}}{3^{3 \log_{10} w + k + 3 - 3 \log_{10} c_2}} = c_3 n w^{1 - 3 \log_{10} 3}$$

where $c_3 = \frac{1}{2}\frac{1}{3^{k+3-3\log_{10} c_2}} \approx 0.00459$.

□

We now continue the proof of Theorem 6. Since the number of grid points in R'_ℓ must be larger than n_ℓ and since R'_ℓ has height at most $\lceil c_1 w \ell \rceil$ (by Lemma 8), width at most w by $(**)$, and $\ell = \log_3 n$, the following is true:

$$w(\lceil c_1 w \log_3 n \rceil) \geq c_3 n w^{1-3\log_{10} 3}.$$

Assume that n is large enough that $c_1 w \log_3 n \geq 1$ (and therefore $2c_1 w \log_3 n \geq \lceil c_1 w \log_3 n \rceil$). Then solving for w gives the lower bound

$$w \geq \left(\frac{c_3}{2c_1}\frac{n}{\log_3 n}\right)^{\frac{1}{1+3\log_{10} 3}}.$$

This contradicts our assumption $(**)$ since $w = c_0(\frac{n}{\log_3 n})^{\frac{1}{1+3\log_{10} 3}}$ and $c_0 < \left(\frac{c_3}{2c_1}\right)^{\frac{1}{1+3\log_{10} 3}} \approx 0.0464$. Therefore, the complete ternary tree T requires at least $c_0(\frac{n}{\log_3 n})^\alpha$ width within R'_ℓ. Since R'_ℓ has height $B_\ell \leq \lceil c_1 w \ell \rceil$ by Lemma 8, the induction hypothesis holds and Theorem 6 is proved. □

Note that the constant in Theorem 6 is $c_0 < 0.000279$, so only for very large values of n is this bound bigger than the bound of Theorem 5.

3.4 Beyond Method 2

In this section, we give ideas that may be used to further improve Theorem 6. Recall that we considered a sequence that satisfies the following recurrence:

i) $n_0 = 1$

ii) $n_i \geq 2n_{i-1}$ for $i = 1, \ldots, \ell$

iii) $n_i \geq 10n_{i-3} - 2(3^{i\alpha})$ for $i = 14, \ldots, \ell$

iv) $n_i \geq 3n_{i-1} - w$ for $i = 1, \ldots, \ell$

where $\ell = \log_3 n$ and give a lower bound for n_ℓ.

We believe that we can improve the lower bound on n_ℓ by replacing recurrence iii) with

$$n_i \geq 26n_{i-4} - 10(3^{i\alpha}) \tag{3.1}$$

by considering subtrees of height four instead of subtrees of height three in the context of trapped subtrees. This will require redefining our rectangles and proving lemmas similar to Lemma 10 and 11 from Method 2. In particular, one would have to show that any node at depth $i-4$ inside rectangle R'_{i-4} has either at least 26 descendants at depth i, or at least one trapped descendant within levels $i-3, \ldots, i$. Pushing this further, we can iteratively increase the height of the subtrees used in the context of trapped subtrees (thereby iteratively improving (3.1)) and we believe we can achieve a width lower bound of $\Omega\left(\left(\frac{n}{\log_3 n}\right)^{\frac{1}{1+\log_{2.617}3}}\right) \approx \Omega\left(n^{0.4668}\right)$. Since the details of these ideas require further work, they are not included in this thesis.

Chapter 4

Conclusions and Open Problems

In this thesis, we have studied octagonal drawings of ternary trees. We have shown an upper bound (Theorem 3) concerning the width requirement of ideal octagonal grid drawings of arbitrary ternary trees and some lower bounds (Theorems 4, 5, and 6) concerning the width requirement of ideal octagonal grid drawings of complete ternary trees.

We have not analyzed the height of the octagonal drawing produced by our algorithm in Chapter 2. We believe that a modification of the algorithm can achieve near-linear height, but the details may require further work and are not included in this thesis.

The upper and lower bounds for octagonal drawings are not asymptotically tight, leaving space for improvement. The exponent in the width upper bound was determined numerically and can likely be improved slightly by expanding the five rules of the L-algorithm into more cases. However, it is unclear how to improve the upper bound for octagonal drawings to match the $O(n^{\log_3 2})$ upper bound for HVA drawings.

In terms of the lower bound, while we were able to improve the width lower bound iteratively, there are limitations to this method. Suppose in any version of this approach, we let w denote the width of our rectangle and $w \log_3 n$ be the height of our rectangle. Even if we can show that the entire complete ternary tree is drawn inside our rectangle (which has $w^2 \log_3 n$ grid points), our best lower bound for width w, will still only be $\Omega((\frac{n}{\log_3 n})^{0.5})$, which is not near our best known width upper bound which is $O(n^{0.68})$ for ideal octagonal ternary tree drawings. In order to show matching width upper and lower bounds for ideal octagonal ternary tree drawings, a different approach for lower bounds seems necessary.

For future work, we are interested in related open problems such as

- What are the width upper and lower bounds for k-ary trees using $O(k)$ slopes?

- How small can you draw a complete ternary tree in an octagonal grid while maintaining the upward property?
- What is the computational complexity of finding ideal octagonal drawings of minimum width?

References

[1] Christian Bachmaier, Franz J. Brandenburg, Wolfgang Brunner, Andreas Hofmeier, Marco Matzeder, and Thomas Unfried. Tree drawings on the hexagonal grid. In *International Symposium on Graph Drawing*, volume 5417 of *Lecture Notes in Computer Science*, pages 372–383. Springer, 2009.

[2] Christian Bachmaier and Marco Matzeder. Drawing unordered trees on k-grids. *Journal of Graph Algorithms and Applications*, 17(2):103–128, 2013.

[3] Johannes Batzill and Therese C. Biedl. Order-preserving drawings of trees with approximately optimal height (and small width). *CoRR*, abs/1606.02233, 2016.

[4] Therese C. Biedl. Ideal tree-drawings of approximately optimal width (and small height). *CoRR*, abs/1502.02753, 2015.

[5] Wolfgang Brunner and Marco Matzeder. Drawing ordered (k-1)-ary trees on k-grids. In *International Symposium on Graph Drawing*, volume 6502 of *Lecture Notes in Computer Science*, pages 105–116. Springer, 2010.

[6] Timothy M. Chan. A near-linear area bound for drawing binary trees. *Algorithmica*, 34(1):1–13, 2002.

[7] Timothy M. Chan, Michael T. Goodrich, S. Rao Kosaraju, and Roberto Tamassia. Optimizing area and aspect ratio in straight-line orthogonal tree drawings. *Computational Geometry*, 23(2):153–162, 2002.

[8] Fabrizio Frati. Straight-line orthogonal drawings of binary and ternary trees. In *International Symposium on Graph Drawing*, volume 4875 of *Lecture Notes in Computer Science*, pages 76–87. Springer, 2007.

[9] Ashim Garg, Michael T. Goodrich, and Roberto Tamassia. Planar upward tree drawings with optimal area. *International Journal of Computational Geometry & Applications*, 6(3):333–356, 1996.

[10] Ashim Garg and Adrian Rusu. Area-efficient order-preserving planar straight-line drawings of ordered trees. *International Journal of Computational Geometry & Applications*, 13(6):487–505, 2003.

[11] Donald E. Knuth. The Art of Computer Programming. vol. 1: Fundamental Algorithms. Addision-Wesley. *Reading, Mass*, 1968.

[12] Edward M. Reingold and John S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, (2):223–228, 1981.