AREA-EFFICIENT GRID DRAWINGS OF GRAPHS

By

Adrian Rusu

August, 2003

A DISSERTATION SUBMITTED TO THE

FACULTY OF THE GRADUATE SCHOOL OF STATE

UNIVERSITY OF NEW YORK AT BUFFALO

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

To my son, wife, and mother, Alex, Amalia, and Olimpia, for their endless trust, support, encouragement, and love, and in the everlasting memory of my father, Ioan

Acknowledgments

I would like to express my deep appreciation to my major advisor, Dr. Ashim Garg, for his clear guidance, encouragement, wise counsel, and unwavering support over the past few years. His influence has been invaluable in my development as a researcher. It is his continuous help, motivation, and insightful advice that made this dissertation possible. I also want to show my respects to him for his incredible enthusiasm for research. Many thanks for supporting me as a Research Assistant when I needed it most.

Many thanks to Dr. Xin He and Dr. Jinhui Xu for being in my dissertation committee. My thanks to Dr. Xin He also for presenting several research problems which sparked my interest in algorithms and graph theory. His input into my academic career is greatly appreciated. I would also like to express my deep appreciation to Dr. Giuseppe Liotta for serving as the outside reader of my dissertation and for his invaluable comments.

During my graduate studies at University at Buffalo, I had the pleasure to make many good friends including Dr. Ramazan Aygun, Mr. Guru Prasad, Dr. Cristian Nastase, Mr. Catalin Tomai, Mr. Samik Sengupta, Mr. Huaming Zhang; Dr. Camil Nichita, Dr. Florin Nichita, Dr. Cezar Joita, and Mr. Dragos Bontea, teammates in the intramural soccer team, winner of five championships; Mr. Alex Eisen, Mr. Surya Kompalli, Mr. Dahai Xu, Ms. Fran Johnson, Mr. John Santore, and Dr. Vishal Anand, colleagues in the CSEGSA.

I would like to thank the Department of Computer Science and Engineering in the State University of New York at Buffalo for first offering me a Teaching Assistantship and later a position as Instructor.

Next, I would like to offer my thanks to many people at the University at Buffalo: the faculty, for providing a propitious teaching and research environment; the office staff, for providing a friendly face; my students, for being very active and challenging; and to Ms. Helene Kershner for her advice, support, and trust.

Special thanks to my wife, Amalia, for her great support from every aspect of everyday life and research, and to my son, Alex, for being such a good boy. I hope I will be a person that they will be proud of.

For their love and inspiration, which has guided me through every crucial stage in my life, I would like to thank my mother, Olimpia, and my father, Ioan, who is gone but never forgotten. I know you are watching over me.

The research in this thesis has been supported by NSF CAREER Award No. IIS-9985136, NSF CISE Research Infrastructure Award No. 0101244, and by The Mark Diamond Research Fund Grant No. 13-Summer-2003 from the Graduate Student Association of the State University of New York.

Abstract

The visualization of relational information is concerned with the presentation of abstract information about relationships between various entities. It has many applications in diverse domains such as software engineering, biology, civil engineering, and cartography. Relational information is typically modeled by an abstract graph, where vertices are entities and edges represent relationships between entities. The aim of graph drawing is to automatically produce drawings of graphs which clearly reflect the inherent relational information.

This thesis is primarily concerned with problems related to the automatic generation of area-efficient grid drawings of trees and outerplanar graphs, which are important categories of graphs.

The main achievements of this thesis include:

1. An algorithm for producing planar straight-line grid drawings of binary trees with optimal linear area and with user-defined arbitrary aspect ratio,

2. An algorithm for producing planar straight-line grid drawings of degree-d trees with n nodes, where $d = O(n^{\delta})$ and $0 \leq \delta < 1/2$ is a constant, with optimal linear area and with user-defined arbitrary aspect ratio,

3. An algorithm which establishes the currently best known upper bound, namely $O(n \log n)$, on the area of order-preserving planar straight-line grid drawings of ordered trees,

4. An algorithm which establishes the currently best known upper bound, namely $O(n \log \log n)$, on the area of order-preserving planar straight-line grid drawings of ordered binary trees,

5. An algorithm for producing order-preserving upward planar straight-line grid drawings of ordered binary trees with optimal $O(n \log n)$ area,

6. An algorithm which establishes the trade-off between the area and aspect ratio of order-preserving planar straight-line grid drawings of ordered binary trees, in the case when the aspect ratio is arbitrarily defined by the user, and

7. An algorithm for producing planar straight-line grid drawings of outerplanar graphs with n vertices and degree d in $O(dn^{1.48})$ area. This result shows for the first time that a large category of outerplanar graphs, namely those with degree $d = O(n^{\delta})$, where $0 \leq \delta < 0.52$ is a constant, can be drawn in sub-quadratic area.

All our algorithms are time-efficient. More specifically, algorithms 1 and 2 run in $O(n \log n)$ time each, and algorithms 3, 4, 5, 6, and 7 run in $O(n)$ time each.

List of Figures

List of Tables

Contents

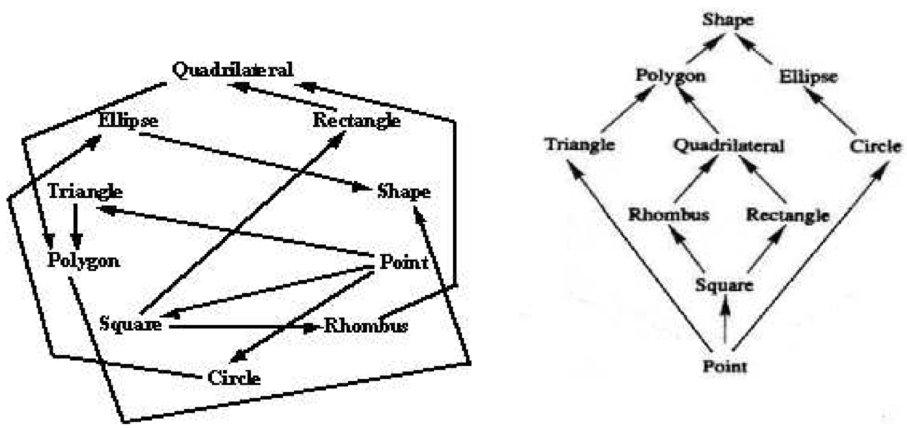
Chapter 1

Introduction

1.1 Graph Drawing

Graph drawing is concerned with the automatic generation of geometric representations of relational information, often for visualization purposes. The typical data structure for modeling relational information is a graph whose vertices represent entities and whose edges correspond to relationships between entities. Visualizations of relational structures are only useful to the degree that the associated diagrams effectively convey information to the people that use them. A good diagram helps the reader understand the system, but a poor diagram can be confusing (see Figure 1.1.1) [11].

The method for laying out data-flow diagrams due to Knuth [24] was one of the first graph drawing algorithms used for visualization purposes [11]. Graph drawing has seen extensive



(a) (b)

Figure 1.1.1: Two diagrams that represent a simple class hierarchy; vertices represent classes of geometric shapes, and edges describe the *is-a relation*. Each vertex represents a class, and a directed edge between two vertices represents the class-subclass relationship. Diagram (a) is more difficult to follow than diagram (b).

research in the last few years. For more results see [11].

The automatic generation of drawings of graphs finds many applications, such as

- software engineering (data flow diagrams, subroutine-call graphs, program nesting trees, object-oriented class hierarchies),
- databases (entity-relationship diagrams),
- information systems (organization charts),
- real-time systems (Petri nets, state-transition diagrams),
- decision support systems (PERT networks, activity trees),
- VLSI (circuit schematics),
- artificial intelligence (knowledge-representation diagrams),

- logic programming (SLD-trees).

Further applications can be found in other science and engineering disciplines, such as

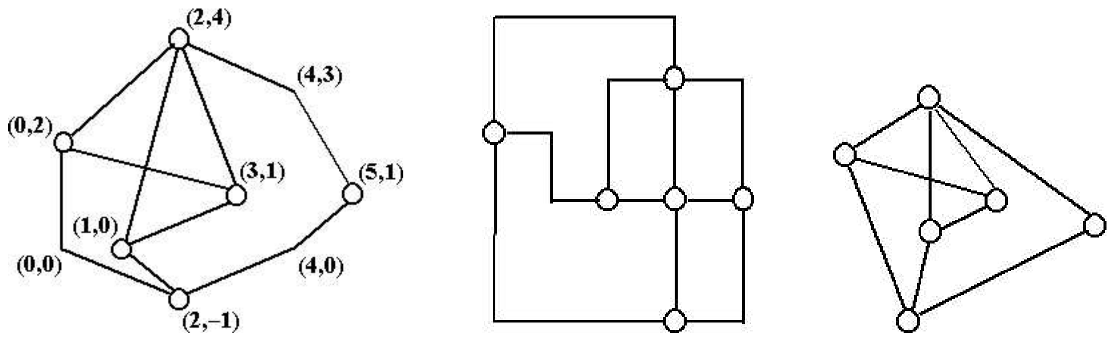
- medical science (concept lattices),
- biology (evolutionary trees),
- chemistry (molecular drawings),
- civil engineering (floor plan maps),
- cartography (map schematics).

The usefulness of a drawing of a graph depends on its *readability*, i.e. its capability of conveying the information contained in the graph quickly and clearly.

Graph drawing algorithms are methods that produce graph drawings which are easy to read. Algorithms for drawing graphs are typically based on some graph-theoretic insight into the structure of the graph. The input to a graph drawing algorithm is a graph G that needs to be drawn. The output is a drawing Γ which maps each vertex of G to a distinct point in the 2D space and each edge (u,v) of G to a simple Jordan curve with endpoints u and v.

1.2 Graph Drawing Conventions

In this thesis we consider planar straight-line grid drawings. Now we explain the properties of these drawings and the motivation behind using them.



(a) (b) (c)

Figure 1.2.1: Grid drawings of the same graph: (a) polyline; (b) planar; (c) straight-line. In (a) we label each vertex and edge bend by their integer coordinates.

1.2.1 Grid Drawings

A *grid drawing* is one in which each vertex is placed at integer coordinates (see Figure 1.2.1(a)). Grid drawings guarantee at least unit distance separation between nodes, and the integer coordinates of nodes allow such drawings to be rendered on displays, such as computer screen, without any distortions due to truncation and round-off errors. We assume that the plane is covered by *horizontal* and *vertical channels*, with unit distance between two consecutive channels. The meeting point of a horizontal and a vertical channel is called a *grid-point*. The smallest rectangle with horizontal and vertical sides parallel to the X and Y axis, that covers the entire grid drawing, is called the *enclosing rectangle*. The *area* of a grid drawing is defined as the number of grid points contained in its enclosing rectangle. Drawings with small area can be drawn with greater resolution on a fixed-size page. The *aspect ratio* of a grid drawing is defined as the ratio of the length of the longest side to the length of the shortest side of its enclosing rectangle. Giving the users control over the aspect ratio of a drawing allows them to display the drawing in different kinds of displays surfaces with different aspect ratios.

The optimal use of the screen space is achieved by minimizing the area of the drawing and by providing user-controlled aspect ratio.

1.2.2 Planar Drawings

A *planar drawing* is a drawing in which no two edges cross (see Figure 1.2.1(a)). Planar drawings are normally easier to understand than non-planar drawings, i.e. drawings with edge-crossings. Planarity is also an important graph theoretic concept, which has been widely studied. Necessary and sufficient conditions for a graph to be planar have been given in [25] and [45]. Linear time algorithms for recognizing planar graphs have been given in [23] and [2]. It has also been shown that every planar graph admits a straight-line planar drawing [44], [15], and [36]. Algorithms for planar straight-line grid drawings of planar graphs with $O(n^2)$ area have been developed independently in [9] and [33]. Extensive research has been done on various kinds of planar drawings. For example, [5, 12, 13, 16, 27, 28, 31, 37, 39–41] provide important results. For more results on planar drawings see [10, 11].

1.2.3 Straight-line Drawings

It is natural to draw each edge of a graph as a straight line between its end-vertices. The so called *straight-line* graph drawings have each edge drawn as a straight line segment (see Figure 1.2.1(c)). Straight-line drawings are easier to understand than polyline drawings, i.e. drawings in which edges have bends (more than one line segment).

The experimental study of the human perception of graph drawings has concluded that minimizing the number of edge crossings and minimizing the number of bends increases the understandability of drawings of graphs [29, 30, 38]. Ideally, the drawings should have no edge crossings, i.e. they should be planar drawings, and should have no edge-bends, i.e. they should be straight-line drawings.

1.3 Contributions and Outline of This Thesis

As mentioned in Section 1.2, planar straight-line drawings are easier to understand than non-planar polyline drawings (see Figure 1.1.1). In this thesis, we study the problem of constructing area-efficient planar straight-line grid drawings of trees and outerplanar graphs. We now outline the structure of this thesis and summarize the principal results obtained: (Note that each chapter is self-contained)

- In Chapter 1 (this Chapter), we give an overview of graph drawing, providing the motivation for the results presented in the reminder of this thesis.

- In Chapter 2, we show that a binary tree admits a planar straight-line grid drawing with optimal linear area and user-defined arbitrary aspect ratio.

- In Chapter 3, we extend the result in Chapter 2 by showing that a degree-d tree with n nodes, where $d = O(n^\delta)$ and $0 \leq \delta < 1/2$ is a constant, admits a planar straight-line grid drawing with optimal linear area and user-defined arbitrary aspect ratio.

- An *ordered tree* T is one with a pre-specified counterclockwise ordering of the edges incident on each node. Ordered trees arise commonly in practice. Examples of ordered trees include binary search trees, arithmetic expression trees, BSP-trees, B-trees, and range-trees. An *order-preserving drawing* of T is one in which the counterclockwise ordering of the edges incident on a node is the same as their pre-specified ordering in T. Ordered trees are generally drawn using order-preserving planar straight-line grid drawings, as any undergraduate textbook on data-structures will show. In Chapter 4, we develop several area-efficient algorithms for constructing order-preserving planar straight-line grid drawings of ordered trees. In particular, we show that:

  - An ordered tree admits an order-preserving planar straight-line grid drawing with $O(n \log n)$ area,

  - An ordered binary tree admits an order-preserving planar straight-line grid drawing with $O(n \log\log n)$ area,

  - An ordered binary tree admits an order-preserving upward planar straight-line grid drawing with optimal $O(n \log n)$ area,

  - In the case when the aspect ratio is arbitrarily defined by the user, we establish the trade-off between the area and aspect ratio of order-preserving planar straight-line grid drawings of ordered binary trees.

- An *outerplanar graph* is a planar graph for which there exists an embedding with all vertices on the exterior face. In Chapter 5, we show that an outerplanar graph with n vertices and degree d admits a planar straight-line grid drawing with $O(dn^{1.48})$ area. This result implies that if $d = O(n^{\delta})$, where $0 \leq \delta < 0.52$ is a constant, then the graph can be drawn in sub-quadratic area.

- In Chapter 6, we summarize the main achievements of this thesis and identify several open problems in grid drawing.

Note that all algorithms presented in this thesis are time-efficient. The algorithms presented in Chapters 2, and 3 run in $O(n \log n)$ time and the algorithms presented in Chapters 4 and 5 run in $O(n)$ time, where n is the number of vertices in the graph that needs to be drawn.

Chapter 2

Planar Straight-line Grid Drawings of Binary Trees with Linear Area and Arbitrary Aspect Ratio

2.1 Introduction

Trees are very common data-structures, which are used to model information in a variety of applications such as Software Engineering (hierarchies of object-oriented programs), Business Administration (organization charts), and Web-site Design (structure of a Web-site). A *drawing* Γ of a tree T maps each node of T to a distinct point in the plane, and each edge (u,v) of T to a simple Jordan curve with endpoints u and v. Γ is a *straight-line* drawing (see Figure 2.1.1(a)), if each edge is drawn as a single line-segment. Γ is a *polyline* drawing (see Figure 2.1.1(b)), if each edge is drawn as a connected sequence of one or more line-segments, where the meeting point of consecutive line-segments is called a *bend*. Γ is an *orthogonal* drawing (see Figure 2.1.1(c)), if each edge is drawn as a chain of alternating horizontal and vertical segments. Γ is a *grid* drawing if all the nodes and edge-bends have integer coordinates. Γ is a *planar* drawing if edges do not intersect each other in the drawing (for example, all the drawings in Figure 2.1.1 are planar drawings). Γ is an *upward* drawing (see Figure 2.1.1(a,b)), if the parent is always assigned either the same or higher y-coordinate than its children. In this chapter, we concentrate on grid drawings. So, we will assume that the plane is covered by a rectangular grid. Let R be a rectangle with sides parallel to the X- and Y-axes. The *width* (*height*) of R is equal to the number of grid points with the same y (x) coordinate contained within R. The *area* of R is equal to the number of grid points contained within R. The *aspect ratio* of R is the ratio of its width and height. R is the *enclosing rectangle* of Γ, if it is the smallest rectangle that covers the entire drawing. The *width*, *height*, *area*, and *aspect ratio* of Γ is equal to the width, height, area, and aspect ratio, respectively, of its enclosing rectangle. T is a binary tree if each node has at most two children. We denote by $T[v]$, the *subtree* of T rooted at a node v of T. $T[v]$ consists of v and all the descendents of v. Γ has the *subtree separation* property [3] if, for any two node-disjoint subtrees $T[u]$ and $T[v]$ of T, the enclosing rectangles of the drawings of $T[u]$ and $T[v]$ do not overlap with each other. Drawings with subtree separation property are more aesthetically pleasing than those without subtree separation property. The subtree

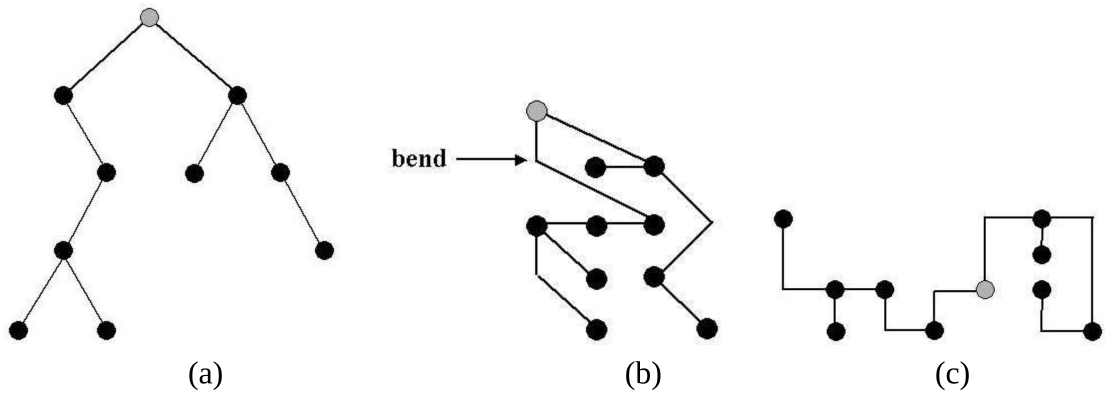


Figure 2.1.1: Various kinds of drawings of the same tree: (a) straight-line, (b) polyline, and (c) orthogonal. Also note that the drawings shown in Figures (a) and (b) are upward drawings, whereas the drawing shown in Figure (c) is not. The root of the tree is shown as a shaded circle, whereas other nodes are shown as black circles.

separation property also allows for a focus+context style [32] rendering of the drawing, so that if the tree has too many nodes to fit in the given drawing area, then the subtrees closer to focus can be shown in detail, whereas those further away from the focus can be contracted and simply shown as filled-in rectangles.

Planar straight-line drawings are more aesthetically pleasing than non-planar polyline drawings. Grid drawings guarantee at least unit distance separation between the nodes of the tree, and the integer coordinates of the nodes and edge-bends allow the drawings to be displayed in a display surface, such as a computer screen, without any distortions due to truncation and rounding-off errors. Giving users control over the aspect ratio of a drawing allows them to display the drawing in different kinds of display surfaces with different aspect ratios. The subtree separation property makes it easier for the user to detect the subtrees in the drawing, and also allows for a focus+context style [32] rendering of the drawing. Finally, it is important to minimize the area of a drawing, so that the users can display a tree in as small drawing area as possible.

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

In this chapter, we answer this question in affirmative, by giving an algorithm that constructs a planar straight-line grid drawing of a binary tree with n nodes with $O(n)$ area in $O(n \log n)$ time. Moreover, the drawing can be parameterized for its aspect ratio, i.e., for any constant α, where $0 \leq \alpha < 1$, the algorithm can construct a drawing with any user-specified aspect ratio in the range $[n^{-\alpha}, n^{\alpha}]$. Theorem 2.4.1 summarizes our overall result. In particular, our result shows that optimal area (equal to $O(n)$) and optimal aspect ratio (equal to 1) is simultaneously achievable (see Corollary 2.4.1). It is also interesting to note that the drawings constructed by our algorithm also exhibit the subtree separation property.

We have also implemented our algorithm, and experimentally evaluated its performance for randomly-generated binary trees with up to 50,000 nodes, and for complete binary trees with up to $65,535 = 2^{16} - 1$ nodes. Our experiments show that it constructs area-efficient drawings in practice, with area at most 10 times the number of nodes in the tree.

An earlier version of this algorithm was presented in [18]. The algorithm presented here (will appear in [21]) achieves a better area bound in practice then the version given in [18].

2.2 Previous Results

Previously, the best-known upper bound on the area of a planar straight-line grid drawing of an n-node binary tree was $O(n \log \log n)$, which was shown in [3] and [35]. This bound is very close to $O(n)$, but still it does not settle the question whether an n-node binary tree can be drawn in this fashion in *optimal* $O(n)$ area. Thus, our result is significant from a theoretical view-point. In fact, we already know of one category of drawings, namely, planar upward orthogonal polyline grid drawings, for which $n \log \log n$ is a tight bound [17], i.e., any binary tree can be drawn in this fashion in $O(n \log \log n)$ area, and there exists a family of binary trees that requires $\Omega(n \log \log n)$ area in any such drawing. So, a natural question arises, if $n \log \log n$ is a tight bound for planar straight-line grid drawings also. Of course, our result implies that this is not the case. Besides, our drawing technique and proofs are significantly different from those of [3] and [35]. Moreover, the drawing constructed by the algorithms of [3] and [35] has a fixed aspect ratio, equal to $\theta(\log^2 n/(n \log \log n))$, whereas the aspect ratio of the drawing constructed by our algorithm can be specified by the user.

We now summarize some other known results on planar grid drawings of binary trees (for more results, see [11]). Let T be an n-node binary tree. [17] presents an algorithm for constructing an upward polyline drawing of T with $O(n)$ area, and any user-specified aspect ratio in the range $[n^{-\alpha}, n^{\alpha}]$, where α is any constant, such that $0 \leq \alpha < 1$. [26] and [43] present algorithms for constructing a (non-upward) orthogonal polyline drawing of T with $O(n)$ area. [3] gives an algorithm for constructing an upward orthogonal straight-line drawing of T with $O(n \log n)$ area, and any user-specified aspect ratio in the range $[\log n/n, n/\log n]$.

It also shows that $n \log n$ is also a tight bound for such drawings. [35] gives an algorithm for constructing an upward straight-line drawing of T with $O(n \log\log n)$ area. If T is a Fibonacci tree, (AVL tree, complete binary tree), then [6, 42] ([8], [6], respectively) give algorithms for constructing an upward straight-line drawing of T with $O(n)$ area.

Table 2.2.1 summarizes these results.

Tree Type	Drawing Type	Area	Aspect Ratio	Reference
Fibonacci	Upward Straight-line	$O(n)$	$\theta(1)$	[6, 42]
AVL	Upward Straight-line	$O(n)$	$\theta(1)$	[8]
Complete Binary	Upward Straight-line	$O(n)$	$\theta(1)$	[6]
General Binary	Upward Orthogonal Polyline	$O(n \log\log n)$	$\theta(\log^2 n/(n \log\log n))$	[17, 35]
	(Non-upward) Orthogonal Polyline	$O(n)$	$\theta(1)$	[26, 43]
	Upward Orthogonal Straight-line	$O(n \log n)$	$[\log n/n, n/\log n]$	[3]
	Upward Polyline	$O(n)$	$[n^{-\alpha}, n^{\alpha}]$	[17]
	Upward Straight-line	$O(n \log\log n)$	$\theta(\log^2 n/(n \log\log n))$	[35]
	(Non-upward) Straight-line	$O(n \log\log n)$	$\theta(\log^2 n/(n \log\log n))$	[3]
		$O(n)$	$[n^{-\alpha}, n^{\alpha}]$	*this chapter*

Table 2.2.1: Bounds on the areas and aspect ratios of various kinds of planar grid drawings of an n-node binary tree. Here, α is a constant, such that $0 \leq \alpha < 1$.

2.3 Preliminaries

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

Let T be a tree, with one distinguished node v, which has at most one child. v is called the *link* node of T. Let n be the number of nodes in T. T is an *ordered* tree if the children of each node are assigned a left-to-right order. A *partial tree* of T is a connected subgraph of T. If T is an ordered tree, then the *leftmost path* p of T is the maximal path consisting of nodes that are leftmost children, except the first one, which is the root of T. The last node of p is called the *leftmost* node of T. Two nodes of T are *siblings* if they have the same parent in T. T is an *empty tree*, i.e., $T = \phi$, if it has zero nodes in it.

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

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

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

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

Theorem 2.3.1 (Separator Theorem [43]) *Every n-node binary tree T contains an edge e, called a* separator edge, *such that removing e from T splits T into two trees T_1 and T_2, with n_1 and n_2 nodes, respectively, such that for some x, where $1/3 \leq x \leq 2/3$, $n_1 \leq xn$, and $n_2 \leq (1-x)n$. Moreover, e can be found in $O(n)$ time.*

Let v be a node of tree T located at grid point (i, j) in Γ. Let Γ be a drawing of T. Assume that the root r of T is located at the grid point $(0,0)$ in Γ. We define the following operations on Γ (see Figure 2.3.1):

- *rotate operation*: rotate Γ counterclockwise by δ degrees around the z-axis passing through r. After a rotation by δ degrees of Γ, node v will get relocated to the point $(i\cos\delta - j\sin\delta, i\sin\delta + j\cos\delta)$. In particular, after rotating Γ by $90°$, node v will get relocated to the grid point $(-j, i)$.

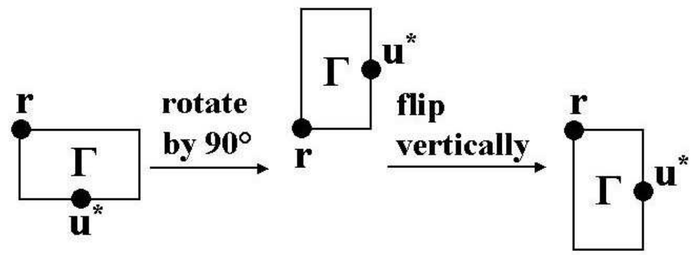


Figure 2.3.1: Rotating a drawing Γ by 90°, followed by flipping it vertically. Note that initially node u^* was located at the bottom boundary of Γ, but after the rotate operation, u^* is on the right boundary of Γ.

- *flip operation*: flip Γ vertically or horizontally. After a horizontal flip of Γ, node v will be located at grid point $(-i, j)$. After a vertical flip of Γ, node v will be located at grid point $(i, -j)$.

2.4 Binary Tree Drawing Algorithm

Let T be a binary tree with a link node u^*. Let n be the number of nodes in T. Let A and ε be two numbers such that $0 < \varepsilon < 1$, and A is in the range $[n^{-\varepsilon}, n^{\varepsilon}]$. A is called the *desirable aspect ratio* for T.

Our tree drawing algorithm, called *DrawTree*, takes ε, A, and T as input, and uses a simple divide-and-conquer strategy to recursively construct a feasible drawing Γ of T, by performing the following actions at each recursive step (as we will prove later, Γ will fit inside a rectangle with area $O(n)$ and aspect ratio A):

- *Split Tree*: Split T into at most five partial trees by removing at most two nodes and their incident edges from it. Each partial tree has at most $(2/3)n$ nodes. Based on

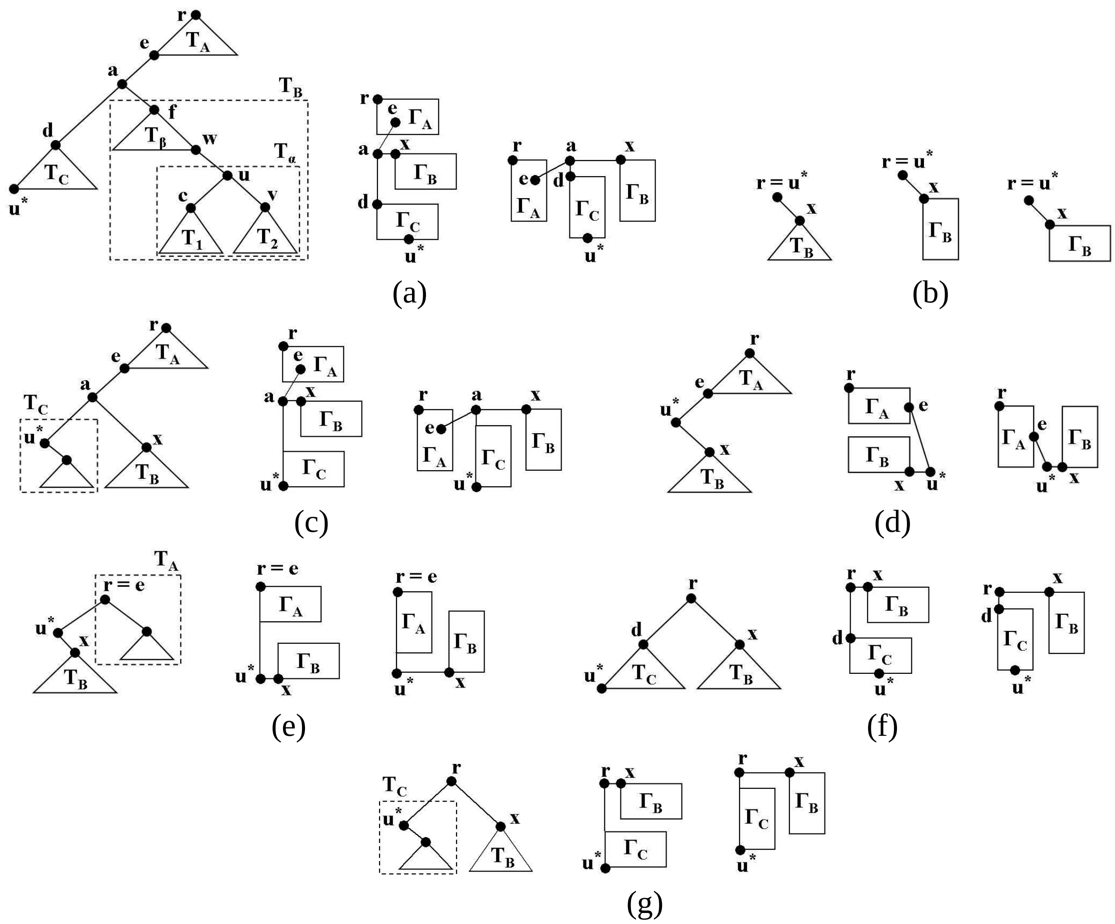


Figure 2.4.1: Drawing T in all the seven subcases of Case 1 (when the separator (u,v) is not in the leftmost path of T): (a) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $d \neq u^*$, (b) $T_A = \emptyset$, $T_C = \emptyset$, (c) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $d = u^*$, (d) $T_A \neq \emptyset$, $T_C = \emptyset$, $r \neq e$, (e) $T_A \neq \emptyset$, $T_C = \emptyset$, $r = e$, (f) $T_A = \emptyset$, $T_C \neq \emptyset$, $d \neq u^*$, and (g) $T_A = \emptyset$, $T_C \neq \emptyset$, $d = u^*$. For each subcase, we first show the structure of T for that subcase, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. Here, x is the same as f if $T_\beta \neq \phi$ and is the same as the root of T_α if $T_\beta = \phi$. In Subcases (a) and (c), for simplicity, e is shown to be in the interior of Γ_A, but actually, either it is the same as r, or if $A < 1$ ($A \geq 1$), then it is placed on the bottom (right) boundary of Γ_A. For simplicity, we have shown Γ_A, Γ_B, and Γ_C as identically sized boxes, but in actuality, they may have different sizes.

the arrangement of these partial trees within T, we get two cases, which are shown in Figures 2.4.1 and 2.4.2, and described later in Section 2.4.1.

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

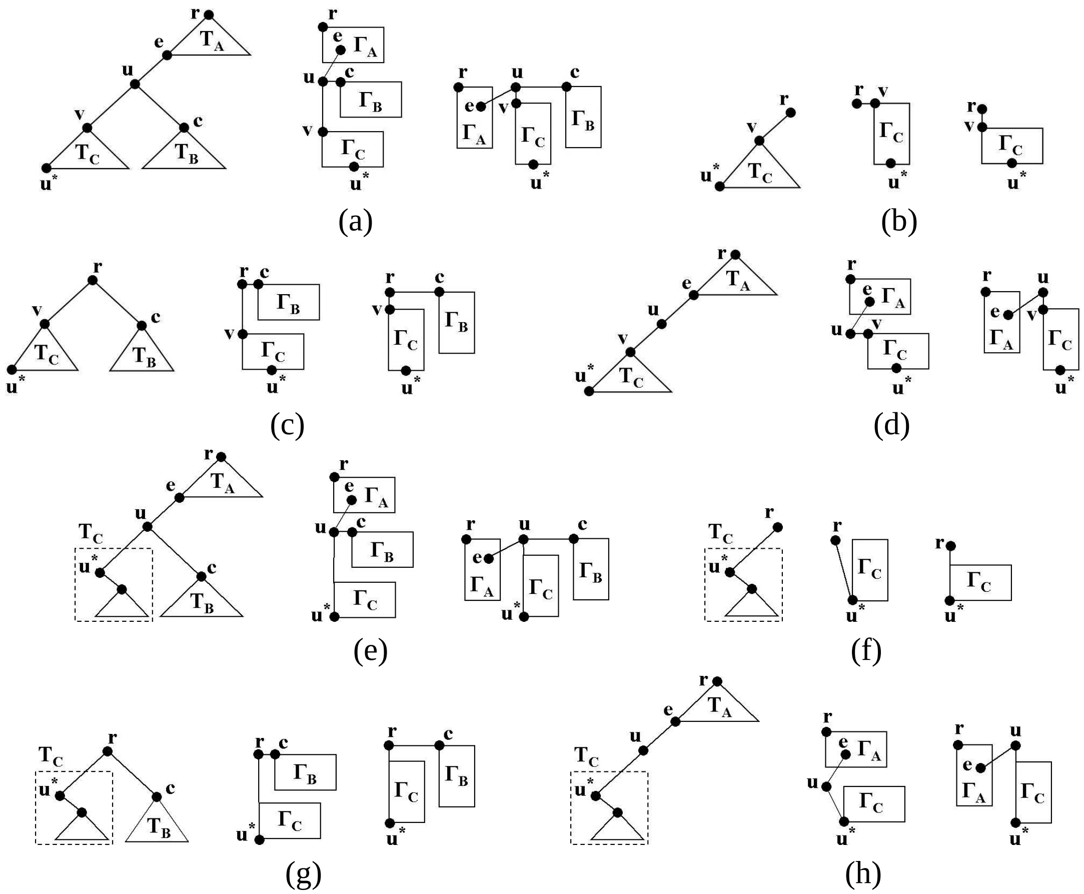


Figure 2.4.2: Drawing T in all the eight subcases of Case 2 (when the separator (u,v) is in the leftmost path of T): (a) $T_A \neq \emptyset$, $T_B \neq \emptyset$, $v \neq u^*$, (b) $T_A = \emptyset$, $T_B = \emptyset$, $v \neq u^*$, (c) $T_A = \emptyset$, $T_B \neq \emptyset$, $v \neq u^*$, (d) $T_A \neq \emptyset$, $T_B = \emptyset$, $v \neq u^*$, (e) $T_A \neq \emptyset$, $T_B \neq \emptyset$, $v = u^*$, (f) $T_A = \emptyset$, $T_B = \emptyset$, $v = u^*$, (g) $T_A = \emptyset$, $T_B \neq \emptyset$, $v = u^*$, and (h) $T_A \neq \emptyset$, $T_B = \emptyset$, $v = u^*$. For each subcase, we first show the structure of T for that subcase, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. In Subcases (a), (d), (e), and (h), for simplicity, e is shown to be in the interior of Γ_A, but actually, either it is same as r, or if $A < 1$ ($A \geq 1$), then it is placed on the bottom (right) boundary of Γ_A. For simplicity, we have shown Γ_A, Γ_B, and Γ_C as identically sized boxes, but in actuality, they may have different sizes.

in T_k.

- *Draw Partial Trees*: Recursively construct a feasible drawing of each partial tree T_k with A_k as its desirable aspect ratio.

- *Compose Drawings*: Arrange the drawings of the partial trees, and draw the nodes and edges, that were removed from T to split it, such that the drawing Γ of T thus obtained is a feasible drawing. Note that the arrangement of these drawings is done based on the cases shown in Figures 2.4.1 and 2.4.2. In each case, if $A < 1$, then the drawings of the partial trees are stacked one above the other, and if $A \geq 1$, then they are placed side-by-side.

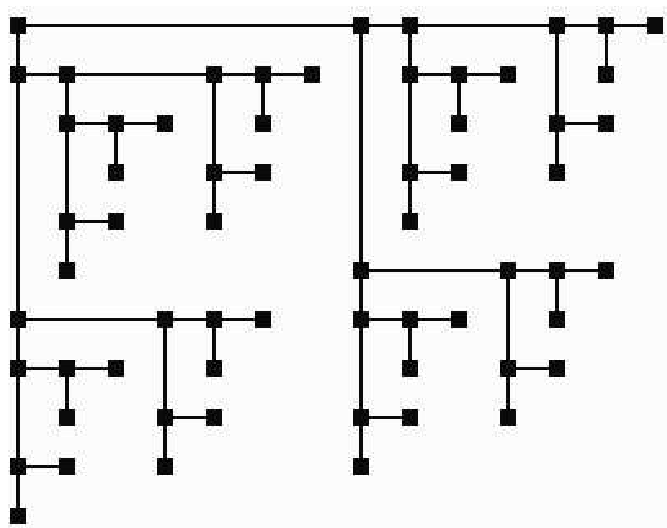


Figure 2.4.3: Drawing of the complete binary tree with 63 nodes constructed by Algorithm *DrawTree*, with $A = 1$ and $\varepsilon = 0.2$.

Figure 2.4.3 shows a drawing of a complete binary tree with 63 nodes constructed by Algorithm *DrawTree*, with $A = 1$ and $\varepsilon = 0.2$.

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

2.4.1 Split Tree

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

- Order the children of each node such that u^* becomes the leftmost node of T.

- Using Theorem 2.3.1, find a separator edge (u,v) of T, where u is the parent of v.

- Based on whether, or not, (u,v) is in the leftmost path of T, we get two cases:

  - *Case 1: The separator edge (u,v) is not in the leftmost path of T.* We get seven subcases: (a) In the general case, T has the form as shown in Figure 2.4.1(a). In this figure:

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

    In addition to this general case, we get six special cases: (b) when $T_A = \emptyset$ and $T_C = \emptyset$ (see Figure 2.4.1(b)), (c) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $d = u^*$ (see Figure 2.4.1(c)), (d) $T_A \neq \emptyset$, $T_C = \emptyset$, $r \neq e$ (see Figure 2.4.1(d)), (e) $T_A \neq \emptyset$, $T_C = \emptyset$, $r = e$ (see

Figure 2.4.1(e)), (f) $T_A = \emptyset$, $T_C \neq \emptyset$, $d \neq u^*$ (see Figure 2.4.1(f)), and (g) $T_A = \emptyset$, $T_C \neq \emptyset$, $d = u^*$ (see Figure 2.4.1(g)). (The reason we get these seven subcases is as follows: T_2 has at least $n/3$ nodes in it because of Theorem 2.3.1. Hence $T_2 \neq \phi$, and so, $T_B \neq \phi$. Based on whether $T_A = \phi$ or not, $T_C = \phi$ or not, $d = u^*$ or not, and $r = e$ or not, we get totally sixteen cases. From these sixteen cases, we obtain the above seven subcases, by grouping some of these cases together. For example, the cases $T_A = \phi$, $T_C = \phi$, $d \neq u^*$, $r = u^*$, and $T_A = \phi$, $T_C = \phi$, $d \neq u^*$, $r \neq u^*$ are grouped together to give Case (a), i.e., $T_A = \phi$, $T_C = \phi$, $d \neq u^*$. So, Case (a) corresponds to 2 cases. Similarly, Cases (c), (d), (e), (f), and (g) correspond to 2 cases each, and Case (b) corresponds to 4 cases.) In each case, we remove nodes a and u, and their incident edges, to split T into at most five partial trees T_A, T_C, T_β, T_1, and T_2. We also designate e as the link node of T_A, w as the link node of T_β, and u^* as the link node of T_C. We arbitrarily select any node of T_1 that has at most one child, and any node of T_2 that has at most one child, and designate them as the link nodes of T_1 and T_2, respectively.

– *Case 2: The separator edge (u,v) is in the leftmost path of T.* We get eight subcases: (a) In the general case, T has the form as shown in Figure 2.4.2(a). In this figure,

  * r is the root of T,
  * c is the right child of u,
  * T_B is the subtree of T rooted at c,
  * e is the parent of u,

- T_A is the maximal tree rooted at r that contains e but not u,
- T_C is the tree rooted at v, and
- $v \neq u^*$.

In addition to the general case, we get the following seven special cases: (b) $T_A = \emptyset$, $T_B = \emptyset$, $v \neq u^*$ (see Figure 2.4.2(b)), (c) $T_A = \emptyset$, $T_B \neq \emptyset$, $v \neq u^*$ (see Figure 2.4.2(c)), (d) $T_A \neq \emptyset$, $T_B = \emptyset$, $v \neq u^*$ (see Figure 2.4.2(d)), (e) $T_A \neq \emptyset$, $T_B \neq \emptyset$, $v = u^*$ (see Figure 2.4.2(e)), (f) $T_A = \emptyset$, $T_B = \emptyset$, $v = u^*$ (see Figure 2.4.2(f)), (g) $T_A = \emptyset$, $T_B \neq \emptyset$, $v = u^*$ (see Figure 2.4.2(g)), and (h) $T_A \neq \emptyset$, $T_B = \emptyset$, $v = u^*$ (see Figure 2.4.2(h)). (The reason we get these eight subcases is as follows: T_C has at least $n/3$ nodes in it because of Theorem 2.3.1. Hence, $T_C \neq \phi$. Based on whether $T_A = \phi$ or not, $T_B = \phi$ or not, and $v = u^*$ or not, we get the eight subcases given above.) In each case, we remove node u, and its incident edges, to split T into at most three partial trees T_A, T_B, and T_C. We also designate e as the link node of T_A, and u^* as the link node of T_C. We arbitrarily select any node of T_B that has at most one child and designate it as the link node of T_B.

2.4.2 Assign Aspect Ratios

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

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

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

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

In Step *Assign Aspect Ratios*, we assign a desirable aspect ratio A_k to each nonempty T_k as follows: Let $x_k = n_k/n$.

- If $A \geq 1$: If T_k is a large partial tree of T, then $A_k = x_k A$, otherwise (i.e., if T_k is a small partial tree of T) $A_k = n_k^{-\varepsilon}$.
- If $A < 1$: If T_k is a large partial tree of T, then $A_k = A/x_k$, otherwise (i.e., if T_k is a small partial tree of T) $A_k = n_k^{\varepsilon}$.

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

2.4.3 Draw Partial Trees

First, we change the desirable aspect ratios assigned to T_A and T_β in some cases as follows: Suppose T_A and T_β get assigned desirable aspect ratios equal to m and p, respectively, where m and p are some positive numbers. In Subcase (d) of Case 1, and if $A \geq 1$, then in Subcases (a) and (c) of Case 1, and Subcases (a), (d), (e), and (h) of Case 2, we change the value of the desirable aspect ratio of T_A to $1/m$. In Case 1, if $A \geq 1$, we change the value of the desirable aspect ratio of T_β to $1/p$. We make these changes because, as explained later in Section 2.4.4, in these cases, we need to rotate the drawings of T_A and T_β by 90° during the *Compose Drawings* step. Drawing T_A and T_β with desirable aspect ratios $1/m$ and $1/p$, respectively, compensates for this rotation, and ensures that the drawings of T_A and T_β used to draw T have the desirable aspect ratios, m and p, respectively.

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

2.4.4 Compose Drawings

Let Γ_k denote the drawing of a partial tree T_k constructed in Step *Draw Partial Trees*. We now describe the construction of a feasible drawing Γ of T from the drawings of its partial trees in both Cases 1 and 2.

In Case 1, we first construct a feasible drawing Γ_α of the partial tree T_α by composing Γ_1 and Γ_2 as shown in Figure 2.4.4, then construct a feasible drawing Γ_B of T_B by composing Γ_α and Γ_β as shown in Figure 2.4.5, and finally construct Γ by composing Γ_A, Γ_B and Γ_C as shown in Figure 2.4.1.

Γ_α is constructed as follows (see Figure 2.4.4): (Recall that if $u \neq a$ then T_α is the subtree of T rooted at u, otherwise $T_\alpha = T_2$)

- If $u \neq a$, and $T_1 \neq \emptyset$ (see Figure 2.4.4(a)), then, if $A < 1$, place Γ_1 above Γ_2 such that the left boundary of Γ_1 is one unit to the right of the left boundary of Γ_2. Place u in the same vertical channel as v and in the same horizontal channel as c. If $A \geq 1$, place Γ_1 one unit to the left of Γ_2, such that the top boundary of Γ_1 is one unit below the top boundary of Γ_2. Place u in the same vertical channel as c and in the same horizontal channel as v. Draw edges (u,c) and (u,v).

- If $u \neq a$, and $T_1 = \emptyset$ (see Figure 2.4.4(b)), then, if $A < 1$, place u in the same horizontal channel and at one unit to the left of v; otherwise (i.e. $A \geq 1$), place u in the same vertical channel and at one unit above v. Draw edge (u,v).

- Otherwise (i.e., if $u = a$), Γ_α is the same as Γ_2 (see Figure 2.4.4(c)).

Γ_B is constructed as follows (see Figure 2.4.5): Let y be the root of T_α. Note that $y = u$ if $u \neq a$, and $y = v$ otherwise.

- if $T_\beta \neq \emptyset$ (see Figure 2.4.5(a)) then, if $A < 1$, then place Γ_β one unit above Γ_α such that the left boundaries of Γ_β and Γ_α are aligned; otherwise (i.e., if $A \geq 1$), first rotate Γ_β by $90°$ and then flip it vertically, then place Γ_β one unit to the left of Γ_α such that the top boundaries of Γ_β and Γ_α are aligned. Draw edge (w,y).

- Otherwise (i.e., if $T_\beta = \emptyset$), Γ_B is same as Γ_α (see Figure 2.4.5(b)).

Γ is constructed from Γ_A, Γ_B, and Γ_C as follows (see Figure 2.4.1): Let x be the root of T_B. Note that $x = f$ if $T_\beta \neq \emptyset$, and $x = y$ otherwise.

- In Subcase (a), as shown in Figure 2.4.1(a), if $A < 1$, stack Γ_A, Γ_B, and Γ_C one above the other, such that they are separated by unit vertical distance from each other, and the left boundaries of Γ_A and Γ_C are aligned with each other and are placed at unit horizontal distance to the left of the left boundary of Γ_B. If $A \geq 1$, then first rotate Γ_A by 90°, and then flip it vertically. Then, place Γ_A, Γ_C, and Γ_B from left-to-right in that order, separated by unit horizontal distances, such that the top boundaries of Γ_A and Γ_B are aligned, and are at unit vertical distance above the top boundary of Γ_C. Then, move Γ_C down until u^* becomes the lowest node of Γ. Place node a in the same vertical channel as d and in the same horizontal channel as r and x. Draw edges (e,a), (a,x), and (a,d).

- In Subcase (b), for both $A < 1$ and $A \geq 1$, place node r one unit above and left of the top boundary of Γ_B (see Figure 2.4.1(b)). Draw edge (r,x).

- The drawing procedure for Subcase (c) is similar to the one in Subcase (a), except that we also flip Γ_C vertically (see Figure 2.4.1(c)).

- In Subcase (d), as shown in Figure 2.4.1(d), if $A < 1$, first flip Γ_B vertically, and then flip it horizontally, so that its root x gets placed at its lower-right corner. Then, first rotate Γ_A by 90°, and then flip it vertically. Next, place Γ_A above Γ_B with unit vertical separation, such that their left boundaries are aligned, next move node e (which is the link node of T_A) to the right until it is either to the right of, or aligned with the right boundary of Γ_B (since Γ_A is a feasible drawing of T_A, by Property 2, as given in Section 2.3, moving e will not create any edge-crossings), and then place u^* in the same horizontal channel as x and one unit to the right of e. If $A \geq 1$, first rotate Γ_A by $90°$, and then flip it vertically. Then flip Γ_B vertically. Then, place Γ_A, u^*, and Γ_B left-to-right in that order separated by unit horizontal distances, such that the top boundaries of Γ_A and Γ_B are aligned, and u^* is placed in the same horizontal channel with the bottom boundary of the drawing among Γ_A and Γ_B with greater height. Draw edges (u^*, e) and (u^*, x).

- In Subcase (e), as shown in Figure 2.4.1(e), if $A < 1$, first flip Γ_B vertically, then place Γ_A and Γ_B one above the other with unit vertical separation, such that the left boundary of Γ_A is at unit horizontal distance to the left of the left boundary of Γ_B. If $A \geq 1$, then first flip Γ_B vertically, place Γ_A to the left of Γ_B at unit horizontal distance, such that their top boundaries are aligned. Next, move Γ_B down until its bottom boundary is at least one unit below the bottom boundary of Γ_A. Place u^* in the same vertical channel as r and in the same horizontal channel as x. Draw edges (r, u^*) and (u^*, x). Note that, since Γ_A is a feasible drawing of T_A, by Property 3 (see Section 2.3), drawing (u^*, r) will not create any edge-crossings.

- The drawing procedure in Subcase (f) is similar to the one in Subcase (a), except that we do not have Γ_A here (see Figure 2.4.1(f)).

- The drawing procedure in Subcase (g) is similar to the one in Subcase (f), except that we also flip Γ_C vertically (see Figure 2.4.1(g).

In Case 2, we construct Γ by composing Γ_A, Γ_B, and Γ_C, as follows (see Figure 2.4.2):

- The drawing procedures in Subcases (a) and (c) are similar to those in Subcases (a) and (f), respectively, of Case 1 (see Figures 2.4.2(a,c)).
- The drawing procedure in Subcase (b) is similar to that in Case (b) of drawing T_α (see Figure 2.4.4(b)).
- In Subcase (d), as shown in Figure 2.4.2(d), if $A > 1$, we place Γ_A above Γ_C, separated by unit vertical distance such that the left boundary of Γ_C is one unit to the right of the left boundary of Γ_A. Place u in the same vertical channel as r and in the same horizontal channel as v. If $A \geq 1$, then first rotate Γ_A by 90°, and then flip it vertically. Then, place Γ_A to the left of Γ_C, separated by unit horizontal distance, such that the top boundary of Γ_C is one unit below the top boundary of Γ_A. Then, move Γ_C down until u^* becomes the lowest node of Γ. Place u in the same vertical channel as v and in the same horizontal channel as r. Draw edges (u,v) and (u,e).
- The drawing procedures in Subcases (e), (f), (g), and (h) are similar to those in Subcases (a), (b), (c), and (d), respectively, (see Figures 2.4.2(e,f,g,h)), except that we also flip Γ_C vertically.

2.4.5 Proof of Correctness

Lemma 2.4.1 (Planarity) *Given a binary tree T with a link node u^*, Algorithm* DrawTree *will construct a feasible drawing Γ of T.*

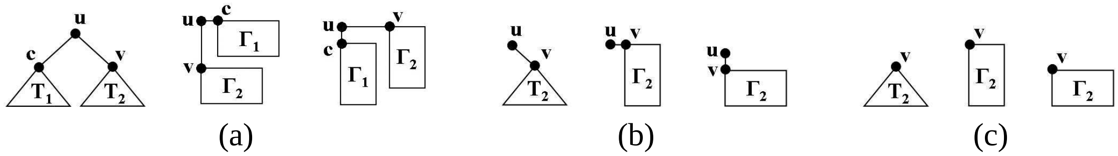


Figure 2.4.4: Drawing T_α, when: (a) $u \neq a$ and $T_1 \neq \emptyset$, (b) $u \neq a$ and $T_1 = \emptyset$, and (c) $u = a$. For each case, we first show the structure of T_α for that case, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. For simplicity, we have shown Γ_1 and Γ_2 as identically sized boxes, but in actuality, their sizes may be different.

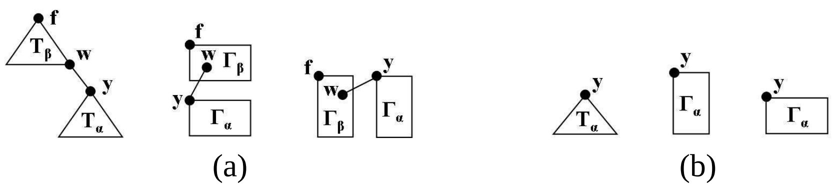


Figure 2.4.5: Drawing T_B when: (a) $T_\beta \neq \emptyset$, and (b) $T_\beta = \emptyset$. Node y shown here is either node u or v. For each case, we first show the structure of T_B for that case, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. In Case (a), for simplicity, w is shown to be in the interior of Γ_β, but actually, it is either same as f, or if $A < 1$ ($A \geq 1$), then is placed on the bottom (right) boundary of Γ_β. For simplicity, we have shown Γ_β and Γ_α as identically sized boxes, but in actuality, their sizes may be different.

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

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

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

Hence, from Figure 2.4.4, it can be easily seen that the drawing Γ_α of T_α is also a feasible drawing.

From Figure 2.4.5, it can be easily seen that the drawing Γ_B of T_B is also a feasible drawing. Note that because Γ_β is a feasible drawing of T_β and w is its link node, w is either at the bottom of Γ_β (from Property 2, see Section 2.3), or at the top-left corner of Γ_β and no other edge or node of T_β is placed on, or crosses the vertical channel occupied by it (Properties 1 and 3, see Section 2.3). Hence, in Figure 2.4.5(a), in the case $A < 1$, drawing edge (w,x) will not cause any edge crossings. Also, in Figure 2.4.5(a), in the case $A \geq 1$, drawing edge (w,x) will not cause any edge crossings because after rotating Γ_β by 90° and flipping it vertically, w will either be at the right boundary of Γ_β (see Property 2), or at the top-left corner of Γ_β and no other edge or node of T_β will be placed on, or cross the horizontal channel occupied by it (see Properties 1 and 3).

Finally, by considering each of the seven subcases shown in Figure 2.4.1 one-by-one, we can show that Γ is also a feasible drawing of T:

- *Subcase (a):* See Figure 2.4.1(a). Γ_A is a feasible drawing of T_A and e is the link node of T_A. Hence, e is either at the bottom of Γ_A (from Property 2), or is at the top-left corner of Γ_A, and no other edge or node of T_A is placed on, or crosses the horizontal and vertical channels occupied by it (from Properties 1 and 3). Hence, in the case $A < 1$, drawing edge (e,a) will not create any edge-crossings, and Γ will also be a feasible drawing of T. In the case $A \geq 1$ also, drawing edge (e,a) will not create any edge-crossings because after rotating Γ_A by 90° and flipping it vertically, e will either be at the right boundary of Γ_A (see Property 2), or at the top-left corner of Γ_β and no other edge or node of T_A will be placed on, or cross the horizontal channel occupied by it (see Properties 1 and 3). Thus, for the case $A \geq 1$ also, Γ will also be a feasible drawing of T.

- *Subcase (b):* See Figure 2.4.1(b). Because Γ_B is a feasible drawing of T_B, it is straightforward to see that Γ is also a feasible drawing of T for both the cases when $A < 1$ and $A \geq 1$.

- *Subcase (c):* See Figure 2.4.1(c). The proof is similar to the one for Subcase (a).

- *Subcase (d):* See Figure 2.4.1(d). Γ_A is a feasible drawing of T_A, e is the link node of T_A, and $e \neq r$. Hence, from Property 2, e is located at the bottom of Γ_A. Rotating Γ_A by 90° and flipping it vertically will move e to the right boundary of Γ_A. Moving e to the right until it is either to the right of, or aligned with the right boundary of Γ_B will not cause any edge-crossings because of Property 2. It can be easily seen that in both the cases, $A < 1$ and $A \geq 1$, drawing edge (e, u^*) does not create any edge-crossings, and Γ is a feasible drawing of T.

- *Subcase (e):* See Figure 2.4.1(e). Γ_A is a feasible drawing of T_A, e is the link node of T_A, and $e = r$. Hence, from Properties 1 and 3, e is at the top-left corner of Γ_A, and no other edge or node of T_A is placed on, or crosses the horizontal and vertical channels occupied by it. Hence, in both the cases, $A < 1$ and $A \geq 1$, drawing edge (e, u^*) will not create any edge-crossings, and Γ is a feasible drawing of T.

- *Subcase (f):* See Figure 2.4.1(f). It is straightforward to see that Γ is a feasible drawing of T for both the cases when $A < 1$ and $A \geq 1$.

- *Subcase (g):* See Figure 2.4.1(g). Γ_C is a feasible drawing of T_C, u^* is the link node of T_C, and u^* is also the root of T_C. Hence, from Properties 1 and 3, u^* is at the top-left corner of Γ_C, and no other edge or node of T_C is placed on, or crosses the horizontal and vertical channels occupied by it. Flipping Γ_C vertically will move u^* to the bottom-left corner of Γ_C and no other edge or node of T_C will be on or crosses the vertical channel occupied by it. Hence, drawing edge (r, u^*) will not create any edge-crossings, and Γ will be a feasible drawing of T.

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

Lemma 2.4.2 (Time) *Given an n-node binary tree T with a link node u^*, Algorithm* DrawTree *will construct a drawing Γ of T in $O(n \log n)$ time.*

Proof: From Theorem 2.3.1, each partial tree into which Algorithm *DrawTree* would split T will have at most $(2/3)n$ nodes in it. Hence, it follows that the depth of the recursion for Algorithm *DrawTree* is $O(\log n)$. At the first recursive level, the algorithm will split T into partial trees, assign aspect ratios to the partial trees and compose the drawings of the partial trees to construct a drawing of T. At the next recursive level, it will split all of these partial trees into smaller partial trees, assign aspect ratios to these smaller partial trees, and compose the drawings of these smaller partial trees to construct the drawings of all the partial trees. This process will continue until the bottommost recursive level is reached. At each recursive level, the algorithm takes $O(m)$ time to split a tree with m nodes into partial trees, assign aspect ratios to the partial trees, and compose the drawings of partial trees to construct a drawing of the tree. At each recursive level, the total number of nodes in all the trees that the algorithm considers for drawing is at most n. Hence, at each recursive level, the algorithm totally spends $O(n)$ time. Hence, the running time of the algorithm is

$O(n) \cdot O(\log n) = O(n \log n)$. □

In Lemma 2.4.4 given below, we prove that the algorithm will draw the tree in $O(n)$ area. Note that the proof given below is different from the one given in [18], which used Theorem 6 of [43]. We believe that the proof given below is more straight-forward, and easier to understand.

Lemma 2.4.3 *Let R be a rectangle with area D and aspect ratio A. Let W and H be the width and height, respectively, of R. Then, $W = \sqrt{AD}$ and $H = \sqrt{D/A}$.*

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

Lemma 2.4.4 (Area) *Let T be a binary tree with a link node u^*. Let n be the number of nodes in T. Let ε and A be two numbers such that $0 < \varepsilon < 1$, and A is in the range $[n^{-\varepsilon}, n^{\varepsilon}]$. Given T, ε, and A as input, Algorithm* DrawTree *will construct a drawing Γ of T that can fit inside a rectangle R with $O(n)$ area and aspect ratio A.*

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

We now give the proof for the case when $A \geq 1$ (the proof for the case $A < 1$ is symmetrical). Algorithm *DrawTree* will split T into at most 5 partial trees. Let T_k be a non-empty partial tree of T, where T_k is one of $T_A, T_\beta, T_1, T_2, T_C$ in Case 1, and is one of T_A, T_B, T_C in Case 2. Let n_k be the number of nodes in T_k, and let $x_k = n_k/n$. Let $P_k = c_1 n - c_2 n^\beta / x_k^{1-\beta}$. From Theorem 2.3.1, it follows that $n_k \leq (2/3)n$, and hence, $x_k \leq 2/3$. Hence, $P_k \leq c_1 n - c_2 n^\beta/(2/3)^{1-\beta} = c_1 n - c_2 n^\beta (3/2)^{1-\beta}$. Let $P' = c_1 n - c_2 n^\beta (3/2)^{1-\beta}$. Thus, $P_k \leq P'$.

From the inductive hypothesis, Algorithm *DrawTree* will construct a drawing Γ_k of T_k that can fit inside a rectangle R_k with aspect ratio A_k and area $D(n_k)$, where A_k is as defined in Section 2.4.2, and $D(n_k) \leq c_1 n_k - c_2 n_k^\beta$. Since $x_k = n_k/n$, $D(n_k) \leq c_1 n_k - c_2 n_k^\beta = c_1 x_k n - c_2 (x_k n)^\beta = x_k (c_1 n - c_2 n^\beta / x_k^{1-\beta}) = x_k P_k \leq x_k P'$.

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

- *T_k is a small partial tree of T:* Then, $n_k < (n/A)^{1/(1+\varepsilon)}$, and also, as explained in Section 2.4.2, $A_k = 1/n_k^\varepsilon$. From Lemma 2.4.3, $W_k = \sqrt{A_k D(n_k)} \leq \sqrt{(1/n_k^\varepsilon)(x_k P')} = \sqrt{(1/n_k^\varepsilon)(n_k/n)P'} = \sqrt{n_k^{1-\varepsilon} P'/n}$. Since $n_k < (n/A)^{1/(1+\varepsilon)}$, $W_k < \sqrt{(n/A)^{(1-\varepsilon)/(1+\varepsilon)} P'/n} = \sqrt{(1/A^{(1-\varepsilon)/(1+\varepsilon)}) P'/n^{2\varepsilon/(1+\varepsilon)}} \leq \sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}}$ since $A \geq 1$.

  From Lemma 2.4.3, $H_k = \sqrt{D(n_k)/A_k} \leq \sqrt{x_k P'/(1/n_k^\varepsilon)} = \sqrt{(n_k/n) P' n_k^\varepsilon} = \sqrt{n_k^{1+\varepsilon} P'/n}$. Since $n_k < (n/A)^{1/(1+\varepsilon)}$, $H_k < \sqrt{(n/A)^{(1+\varepsilon)/(1+\varepsilon)} P'/n} = \sqrt{(n/A)P'/n} = \sqrt{P'/A}$.

- *T_k is a large partial tree of T:* Then, as explained in Section 2.4.2, $A_k = x_k A$. From

Lemma 2.4.3, $W_k = \sqrt{A_k D(n_k)} \leq \sqrt{x_k A x_k P'} = x_k\sqrt{AP'}$.

From Lemma 2.4.3, $H_k = \sqrt{D(n_k)/A_k} \leq \sqrt{x_k P'/(x_k A)} = \sqrt{P'/A}$.

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

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

and

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

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

R' does not have aspect ratio equal to A, but it is contained within a rectangle R with aspect ratio A, area $D(n)$, width W, and height H, where

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

and

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

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

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

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

Since $P' < c_1n$,

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

where c_{12} and c_{13} are some constants (because, since $0 < \varepsilon < 1$, $(1-\varepsilon)/(2(1+\varepsilon)) < (1-\varepsilon)/(1+\varepsilon) < 1/(1+\varepsilon)$, $\varepsilon < (1+\varepsilon)/2$, and $1/2 < (1+\varepsilon)/2$).

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

Hence, $D(n) \leq c_1n - c_2n^\beta(1+c_{14}) + c_{12}n^{1/(1+\varepsilon)} + c_{13}n^{(1+\varepsilon)/2} = c_1n - c_2n^\beta - (c_{14}n^\beta - c_{12}n^{1/(1+\varepsilon)} - c_{13}n^{(1+\varepsilon)/2})$. Thus, for a large enough constant n_0, and large enough β, where

$1 > \beta > \max\{1/(1+\varepsilon),(1+\varepsilon)/2\}$, for all $n \geq n_0$, $c_{14}n^{\beta} - c_{12}n^{1/(1+\varepsilon)} - c_{13}n^{(1+\varepsilon)/2} \geq 0$, and hence $D(n) \leq c_1 n - c_2 n^{\beta}$.

The proof for the case $A < 1$ uses the same reasoning as for the case $A \geq 1$. With T_k, R_k, W_k, H_k, R', W', H', R, W, and H defined as above, and A_k as defined in Section 2.4.2, we get the following values for W_k, H_k, W', H', W, H, and $D(n)$:

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

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

Theorem 2.4.1 (Main Theorem) *Let T be a binary tree with n nodes. Given any number A, where $n^{-\alpha} \leq A \leq n^{\alpha}$, for some constant α, where $0 \leq \alpha < 1$, we can construct in*

$O(n \log n)$ time, a planar straight-line grid drawing of T with $O(n)$ area, and aspect ratio A.

Proof: Let ε be a constant such that $n^{-\varepsilon} \leq A \leq n^{\varepsilon}$ and $0 < \varepsilon < 1$. Designate any node of T that has at most one child as its link node. Construct a drawing Γ of T in R by calling Algorithm *DrawTree* with T, A and ε as input. From Lemmas 2.4.1, 2.4.2, and 2.4.4, Γ will be a planar straight-line grid drawing of T contained entirely within a rectangle with $O(n)$ area, and aspect ratio A. □

Corollary 2.4.1 *Let T be a binary tree with n nodes. We can construct in $O(n \log n)$ time, a planar straight-line grid drawing of T with optimal (equal to $O(n)$) area, and optimal aspect ratio (equal to 1).*

Proof: Immediate from Theorem 2.4.1, with $A = 1$. □

2.5 Experimental Results

We have implemented the algorithm using C++. The implementation consists of about 2,100 lines of code. We have also experimentally evaluated the algorithm on two types of binary trees, namely, randomly-generated, consisting of up to 50,000 nodes, and complete, consisting of up to $65,535 = 2^{16} - 1$ nodes.

Each randomly-generated tree T_n with n nodes is generated by constructing a sequence of trees $T_0, T_1, \ldots, T_n$, where T_0 is the empty tree, and T_{i+1} is generated from T_i by inserting a new node in it, using the Algorithm *InsertRandomly*:

Algorithm *InsertRandomly*(u, T): $\{u$ is a new node to be inserted in tree $T\}$

1. If T is the empty tree then
   - Set u as its root.
2. Else, randomly choose to perform one of the following two pairs of steps ($a-b$ or $c-d$):
   (a) If $T \rightarrow$ left is NULL, then $T \rightarrow$ left = u.
   (b) Else *InsertRandomly*(u, $T \rightarrow$ left).
   (c) If $T \rightarrow$ right is NULL, then $T \rightarrow$ right = u.
   (d) Else *InsertRandomly*(u, $T \rightarrow$ right).

(in our implementation, a tree T is stored as a pointer to its root, and $T \rightarrow$ left and $T \rightarrow$ right represent pointers to the left and right children of the root of T, respectively)

Recall that the algorithm takes three values as input: a binary tree T with n nodes, a number ε, where $0 < \varepsilon < 1$, and a number A in the range $[n^{-\varepsilon}, n^{\varepsilon}]$.

The performance criteria we have used to evaluate the algorithm is the ratio c of the area of the drawing constructed of a tree T, and the number of nodes in T. Recall that the area and aspect ratio of a drawing is defined as the area and aspect ratio, respectively, of its enclosing rectangle.

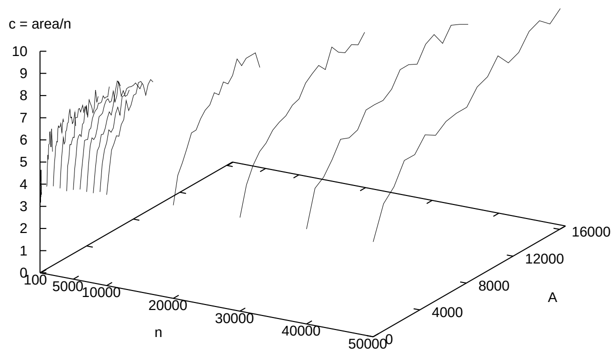
To evaluate the algorithm, we varied n to up to $50,000$, for randomly-generated trees, and to up to $65,535 = 2^{16} - 1$, for complete trees. For each n, we used five different values for ε, namely, 0.1, 0.25, 0.5, 0.75, and 0.9. For each (n, ε) pair, we used 20 different values of A uniformly distributed in the range $[1, n^{\varepsilon}]$. The performance of the algorithm is symmetrical for $A < 1$ and $A > 1$. Hence, we varied A only from 1 through n^{ε}, not from $n^{-\varepsilon}$ through n^{ε} (the only difference between $A < 1$ and $A > 1$ is that for $A < 1$ the algorithm constructs drawings with longer height than width, whereas for $A > 1$, it constructs drawings with longer width than height). Hence, in the rest of the section, we will assume that $A \geq 1$. For each type of tree (randomly-generated and complete), and for each triplet (n, A, ε), we generated three trees of that type. We constructed a drawing of each tree using the algorithm, and computed the value of c. Next, we averaged the values of c obtained for the three trees to get a single value for each triplet (n, A, ε) for each tree-type.

Our experiments show that the value of c is generally small, and varies between 3 and 10. More specifically, it varies between 3 and 10 for randomly-generated, and 3 and 8 for complete trees. Figures 2.5.1 and 2.5.2 show how c varies with n, A, and ε for randomly-generated and complete trees, respectively.

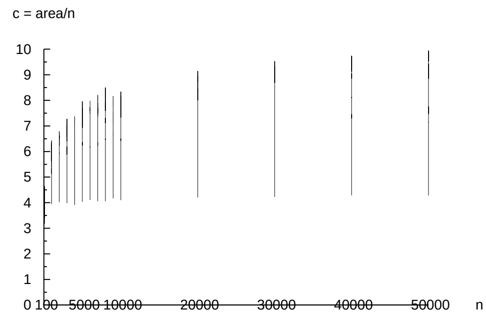
We also discovered that c increases with A for a given n and ε. However, the rate of increase is very small. Consequently, for a given n and ε, the range for c over all the values of A is small (see Figure 2.5.1(b,d,f,h,j) and Figure 2.5.2(b,d,f,h,j)). For example, for $n = 10,000$, and $\varepsilon = 0.5$, the range for c is $[4.2, 5.2]$. Similarly, for a given n and A, c increases with ε.

Finally, we would like to comment that the aspect ratio of the drawing constructed is, in general, different from the input aspect ratio A. This is so because of two reasons. First, while large partial trees get an aspect ratio that is proportional to their sizes, small partial trees get an aspect ratio that is larger than what they would have got had they been assigned aspect ratios proportional to their sizes. Second, the algorithm adds some horizontal and vertical channels to place vertices a and u.

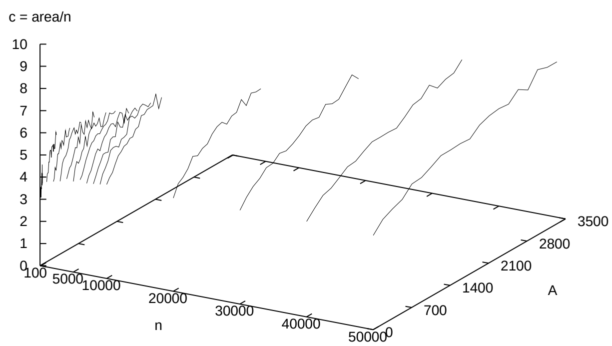
We computed the ratio r of the aspect ratio of the drawing constructed by the algorithm and input aspect ratio A. We discovered that r is close to 1 for $A = 1$, generally decreases as we increase A, and can get as low as 0.1 for $A = n^{\varepsilon}$. However, we also discovered that for a large range of values for A, namely, $[1, \min\{n^{\varepsilon}, n/\log^2 n\}]$, r stays within the range $[0.8, 1.2]$, and so is close to 1. Hence, even in applications, that require the drawing to be of exactly the same aspect ratio as A, we can obtain a drawing with small area and aspect ratio exactly equal to A by adding enough "white space" to the drawing constructed by our drawing algorithm. Adding the white space will increase the area of the drawing by a factor of at most $1/0.8 = 1.25$ (assuming that A is in the above-mentioned range). Hence, the area of the drawing will still be relatively small, namely, at most $10 \times 1.25 = 12.5$ times the number of nodes in the tree.



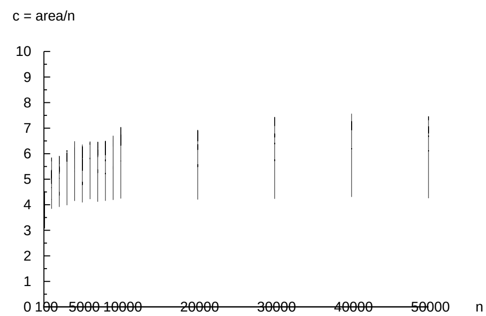
(a) $\varepsilon = 0.9$



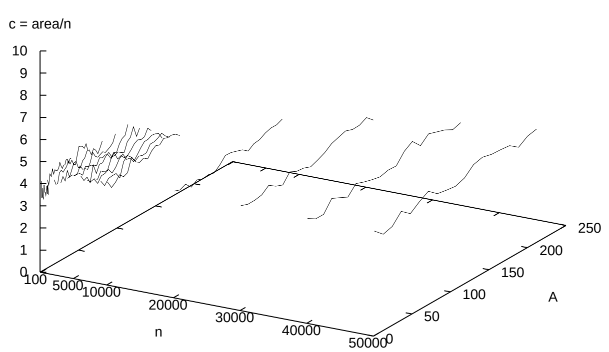
(b) $\varepsilon = 0.9$



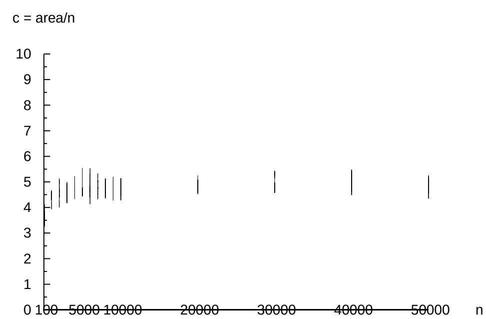
(c) $\varepsilon = 0.75$



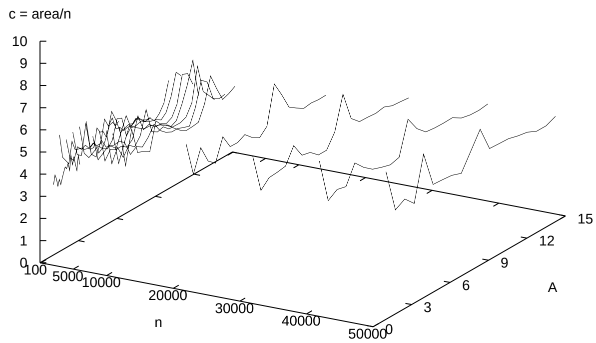
(d) $\varepsilon = 0.75$



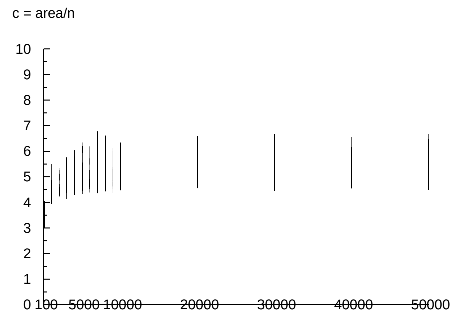
(e) $\varepsilon = 0.5$



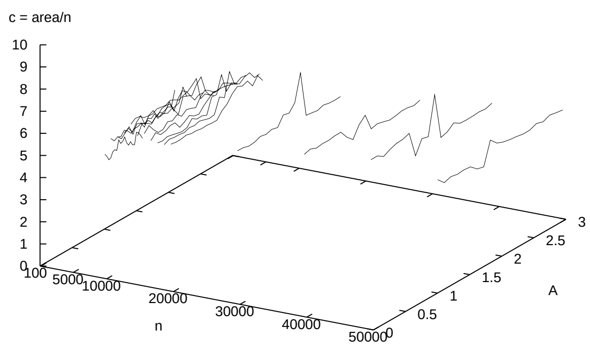
(f) $\varepsilon = 0.5$



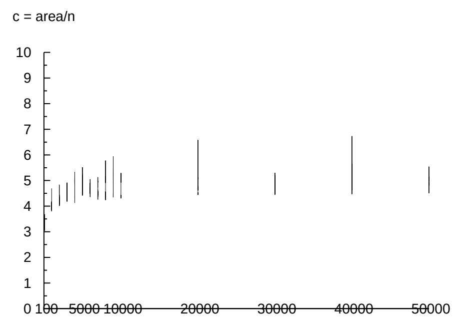
(g) $\varepsilon = 0.25$



(h) $\varepsilon = 0.25$

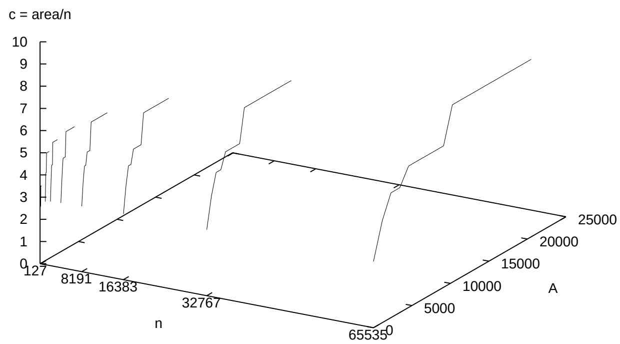


(i) $\varepsilon = 0.1$

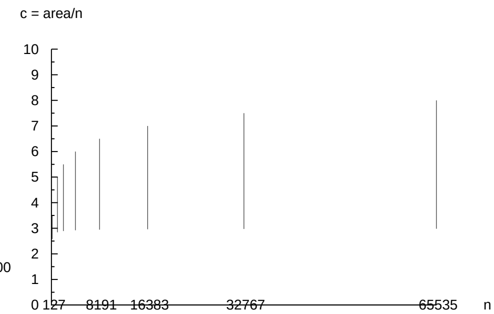


(j) $\varepsilon = 0.1$

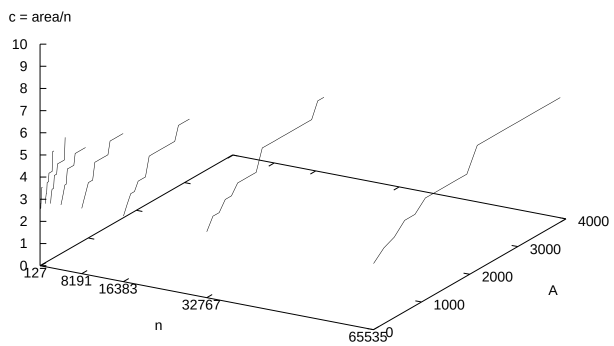
Figure 2.5.1: Performance of the algorithm, as given by the value of *c*, for drawing a randomly-generated binary tree *T* with different values of *A* and ε, where *c=area of drawing/number of nodes n in T*: (a) $\varepsilon = 0.9$, (c) $\varepsilon = 0.75$, (e) $\varepsilon = 0.5$, (g) $\varepsilon = 0.25$, and (i) $\varepsilon = 0.1$. Figures (b), (d), (f), (h), and (j) contain the projections on the XZ-plane of the plots shown in Figures (a), (c), (e), (g), and (i), respectively, and show for each ε, the ranges for the values of *c* for different values of *A* for each *n*.



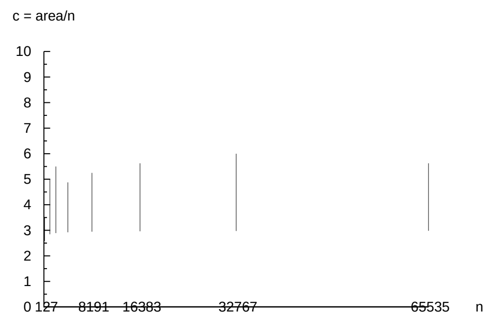
(a) $\varepsilon = 0.9$



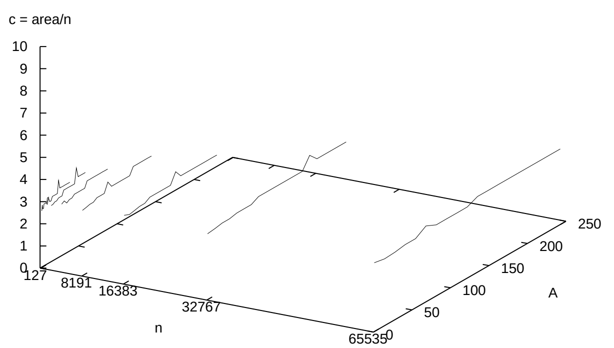
(b) $\varepsilon = 0.9$



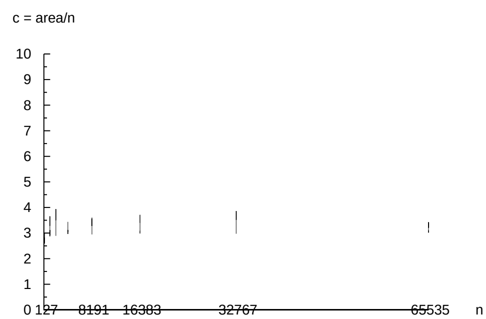
(c) $\varepsilon = 0.75$



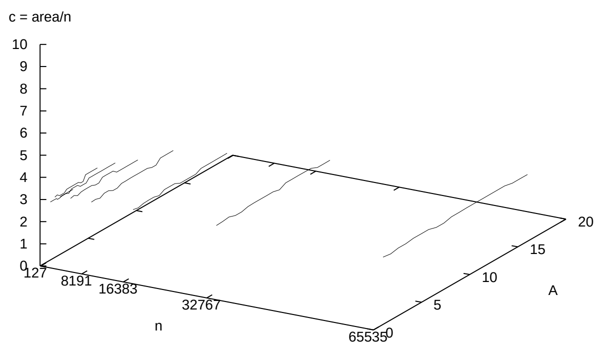
(d) $\varepsilon = 0.75$



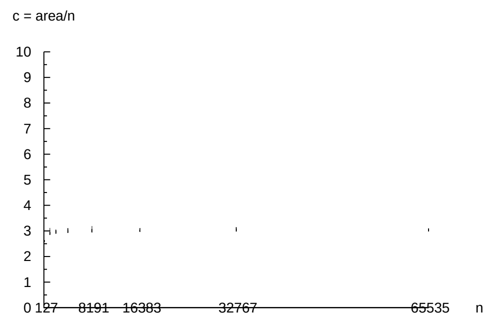
(e) $\varepsilon = 0.5$



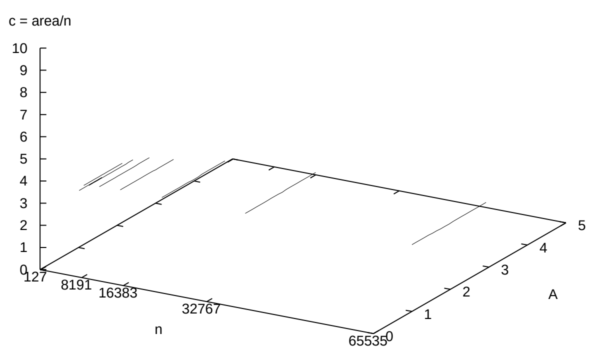
(f) $\varepsilon = 0.5$



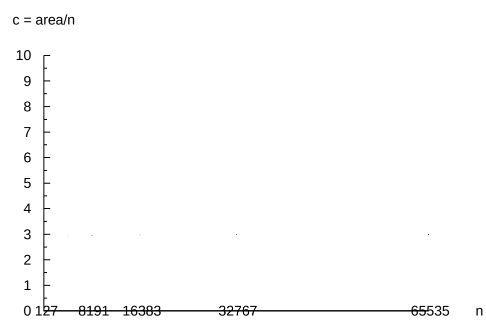
(g) $\varepsilon = 0.25$



(h) $\varepsilon = 0.25$



(i) $\varepsilon = 0.1$



(j) $\varepsilon = 0.1$

Figure 2.5.2: Performance of the algorithm, as given by the value of c, for drawing a complete binary tree T with different values of A and ε, where *c=area of drawing/number of nodes n in T*: (a) $\varepsilon = 0.9$, (c) $\varepsilon = 0.75$, (e) $\varepsilon = 0.5$, (g) $\varepsilon = 0.25$, and (i) $\varepsilon = 0.1$. Figures (b), (d), (f), (h), and (j) contain the projections on the XZ-plane of the plots shown in Figures (a), (c), (e), (g), and (i), respectively, and show for each ε, the ranges for the values of c for different values of A for each n.

Chapter 3

Planar Straight-line Grid Drawings of General Trees with Linear Area and Arbitrary Aspect Ratio

3.1 Introduction

Trees are very common data-structures, which are used to model information in a variety of applications. such as Software Engineering (hierarchies of object-oriented programs), Business Administration (organization charts), and Web-site Design (structure of a Web-site). A *drawing* Γ of a tree T maps each node of T to a distinct point in the plane, and each edge (u,v) of T to a simple Jordan curve with endpoints u and v. Γ is a *straight-line* drawing (see Figure 3.1.1(a)), if each edge is drawn as a single line-segment. Γ is a *polyline* drawing (see Figure 3.1.1(b)), if each edge is drawn as a connected sequence of one or more line-segments, where the meeting point of consecutive line-segments is called a *bend*. Γ is an *orthogonal* drawing (see Figure 3.1.1(c)), if each edge is drawn as a chain of alternating horizontal and vertical segments. Γ is a *grid* drawing if all the nodes and edge-bends have integer coordinates. Γ is a *planar* drawing if edges do not intersect each other in the drawing (for example, all the drawings in Figure 3.1.1 are planar drawings). Γ is an *upward* drawing (see Figure 3.1.1(a,b)), if the parent is always assigned either the same or higher y-coordinate than its children. In this chapter, we concentrate on grid drawings. So, we will assume that the plane is covered by a rectangular grid. Let R be a rectangle with sides parallel to the X- and Y-axes. The *width* (*height*) of R is equal to the number of grid points with the same y (x) coordinate contained within R. The *area* of R is equal to the number of grid points contained within R. The *aspect ratio* of R is the ratio of its width and height. R is the *enclosing rectangle* of Γ, if it is the smallest rectangle that covers the entire drawing. The *width*, *height*, *area*, and *aspect ratio* of Γ is equal to the width, height, area, and aspect ratio, respectively, of its enclosing rectangle. The *degree* of a node of T is the number of edges incident on it. The *degree* of T is the maximum degree of any node in it. T is a *binary* tree if it has degree 3. We denote by $T[v]$, the *subtree* of T rooted at a node v of T. $T[v]$ consists of v and all the descendents of v. Γ has the *subtree separation* property [3] if, for any two node-disjoint subtrees $T[u]$ and $T[v]$ of T, the enclosing rectangles of the drawings of $T[u]$ and $T[v]$ do not overlap with each other. Drawings with subtree separation property are more aesthetically pleasing than those without subtree separation property. The subtree

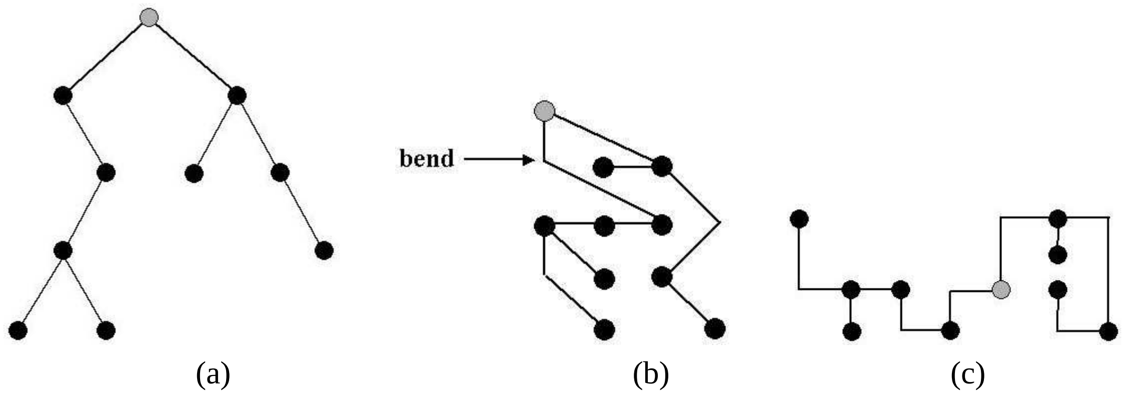


Figure 3.1.1: Various kinds of drawings of the same tree: (a) straight-line, (b) polyline, and (c) orthogonal. Also note that the drawings shown in Figures (a) and (b) are upward drawings, whereas the drawing shown in Figure (c) is not. The root of the tree is shown as a shaded circle, whereas other nodes are shown as black circles.

separation property also allows for a focus+context style [32] rendering of the drawing, so that if the tree has too many nodes to fit in the given drawing area, then the subtrees closer to focus can be shown in detail, whereas those further away from the focus can be contracted and simply shown as filled-in rectangles.

Planar straight-line drawings are more aesthetically pleasing than non-planar polyline drawings. Grid drawings guarantee at least unit distance separation between the nodes of the tree, and the integer coordinates of the nodes and edge-bends allow the drawings to be displayed in a display surface, such as a computer screen, without any distortions due to truncation and rounding-off errors. Giving users control over the aspect ratio of a drawing allows them to display the drawing in different kinds of display surfaces with different aspect ratios. The subtree separation property makes it easier for the user to detect the subtrees in the drawing, and also allows for a focus+context style [32] rendering of the drawing. Finally, it is important to minimize the area of a drawing, so that the users can display a tree in as small drawing area as possible.

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

In Chapter 2 we showed that a binary tree can be drawn in this fashion in $O(n)$ area. However, trees with degree greater than 3 appear quite commonly in practical applications. Hence, an important natural question arises, if this result can be generalized to higher degree trees also. In this chapter, we partially answer this question in affirmative, by giving an algorithm that constructs a planar straight-line grid drawing of a degree-d tree with n nodes, where $d = O(n^{\delta})$ is a positive integer and $0 \leq \delta < 1/2$ is a constant, with $O(n)$ area in $O(n \log n)$ time. Moreover, the drawing can be parameterized for its aspect ratio, i.e., for any constant α, where $0 \leq \alpha < 1$, the algorithm can construct a drawing with any user-specified aspect ratio in the range $[n^{-\alpha}, n^{\alpha}]$. Theorem 3.4.1 summarizes our overall result. In particular, our result shows that optimal area (equal to $O(n)$) and optimal aspect ratio (equal to 1) is simultaneously achievable (see Corollary 3.4.1). It is also interesting to note that the drawings constructed by our algorithm also exhibit the subtree separation property.

3.2 Previous Results

Previously, the best-known bound on area of planar straight-line grid drawings of general trees was $O(n \log n)$, which can be achieved by a simple modification of the HV-drawing algorithm of [6].

Most of the research on drawing trees has dealt with binary trees. In Chapter 2 we present an algorithm for constructing a planar straight-line grid drawing of a binary tree with area $O(n)$.

We now summarize some other known results on planar grid drawings of binary trees (for more results, see [11]). Let T be an n-node binary tree. [17] presents an algorithm for constructing an upward polyline drawing of T with $O(n)$ area, and any user-specified aspect ratio in the range $[n^{-\alpha}, n^{\alpha}]$, where α is any constant, such that $0 \leq \alpha < 1$. [26] and [43] present algorithms for constructing a (non-upward) orthogonal polyline drawing of T with $O(n)$ area. [3] gives an algorithm for constructing an upward orthogonal straight-line drawing of T with $O(n \log n)$ area, and any user-specified aspect ratio in the range $[\log n/n, n/\log n]$. It also shows that $n \log n$ is also a tight bound for such drawings. [35] gives an algorithm for constructing an upward straight-line drawing of T with $O(n \log \log n)$ area. If T is a Fibonacci tree, (AVL tree, complete binary tree), then [6, 42] ([8], [6], respectively) give algorithms for constructing an upward straight-line drawing of T with $O(n)$ area.

Table 3.2.1 summarizes these results.

Tree Type	Drawing Type	Area	Aspect Ratio	Reference
Fibonacci	Upward Straight-line	$O(n)$	$\theta(1)$	[6,42]
AVL	Upward Straight-line	$O(n)$	$\theta(1)$	[8]
Complete Binary	Upward Straight-line	$O(n)$	$\theta(1)$	[6]
General Binary	Upward Orthogonal Polyline	$O(n \log \log n)$	$\theta(\log^2 n/(n \log \log n))$	[17,35]
	(Non-upward) Orthogonal Polyline	$O(n)$	$\theta(1)$	[26,43]
	Upward Orthogonal Straight-line	$O(n \log n)$	$[\log n/n, n/\log n]$	[3]
	Upward Polyline	$O(n)$	$[n^{-\alpha}, n^{\alpha}]$	[17]
	Upward Straight-line	$O(n \log \log n)$	$\theta(\log^2 n/(n \log \log n))$	[35]
	(Non-upward) Straight-line	$O(n \log \log n)$	$\theta(\log^2 n/(n \log \log n))$	[3]
		$O(n)$	$[n^{-\alpha}, n^{\alpha}]$	[18]
Degree-$O(n^{\delta})$ $0 \leq \delta < 1/2$	(Non-upward) Straight-line	$O(n)$	$[n^{-\alpha}, n^{\alpha}]$	*this chapter*

Table 3.2.1: Bounds on the areas and aspect ratios of various kinds of planar grid drawings of an n-node tree. Here, α is a constant, such that $0 \leq \alpha < 1$.

The paper based on this Chapter has been presented in [22].

3.3 Preliminaries

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

Let T be a degree-d tree, with one distinguished node v, which has at most $d-1$ children. v is called the *link* node of T. Let n be the number of nodes in T. T is an *ordered* tree if the children of each node are assigned a left-to-right order. A *partial tree* of T is a connected subgraph of T. If T is an ordered tree, then the *leftmost path* p of T is the maximal path consisting of nodes that are leftmost children, except the first one, which is the root of T. The last node of p is called the *leftmost* node of T. Two nodes of T are *siblings* if they have the same parent in T. T is an *empty tree*, i.e., $T = \phi$, if it has zero nodes in it.

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

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

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

- **Property 1**: The root r is placed at the top-left corner of Γ.
- **Property 2**: If $u^* \neq r$, then u^* is placed at the bottom boundary of Γ. Moreover, we

can move u^* downwards in its vertical channel by any distance without causing any edge-crossings in Γ.

- **Property 3**: If $u^* = r$, then no other node or edge of T is placed on, or crosses the vertical and horizontal channels occupied by r.

Theorem 3.3.1 *In any degree-d tree T, there is a node u, such that removing u and its incident edges splits T into at most d trees, where each tree has at most $n/2$ nodes in it, and $n \geq 2$ is the number of nodes in T. Node u is called a* separator node *of T. Moreover, u can be found in $O(n)$ time.*

Proof: To obtain u, we invoke Algorithm *FindSeparator* with the root of T as its input. Also, as a pre-process step, before invoking Algorithm *FindSeparator*, we first compute the number of nodes $n(v)$ in the subtree rooted at each node v of T. We store the value of $n(v)$ in each node v. We now describe Algorithm *FindSeparator*.

Algorithm *FindSeparator*(u): $\{u$ is a node of $T\}$

1. Let $s_1, \ldots, s_j$ be the children of u.
2. Let s_i be the child of u such that $n(s_i) = \max\{n(s_1), \ldots, n(s_j)\}$.
3. If $n(s_i) > n/2$, then Return *FindSeparator*(s_i).
4. Else Return u.

The algorithm clearly runs in $O(n)$ time.

Let u be the node of T returned on calling Algorithm *FindSeparator*(r), where r is the root of T. We will prove that u is a separator node of T. It is easy to see from the description of the algorithm that $n(u) > n/2$ and $n(s) \leq n/2$, for each child s of u. Let T' be the partial tree of T obtained by removing the subtree rooted at u from T. Since $n(u) > n/2$, the number of nodes in T' is less than $n - n/2 = n/2$. Also recall that $n(s) \leq n/2$, for each child s of u. Hence, removing u and its incident edges will split T into at most d trees, each containing at most $n/2$ nodes. Hence, u is indeed a separator node of T. □

Let v be a node of tree T located at grid point (i, j) in Γ. Let Γ be a drawing of T. Assume that the root r of T is located at the grid point $(0,0)$ in Γ. We define the following operations on Γ (see Figure 3.3.1):

- *rotate operation*: rotate Γ counterclockwise by δ degrees around the z-axis passing through r. After a rotation by δ degrees of Γ, node v will get relocated to the point $(i\cos\delta - j\sin\delta, i\sin\delta + j\cos\delta)$. In particular, after rotating Γ by 90°, node v will get relocated to the grid point $(-j, i)$.

- *flip operation*: flip Γ vertically or horizontally. After a horizontal flip of Γ, node v will be located at grid point $(-i, j)$. After a vertical flip of Γ, node v will be located at grid point $(i, -j)$.

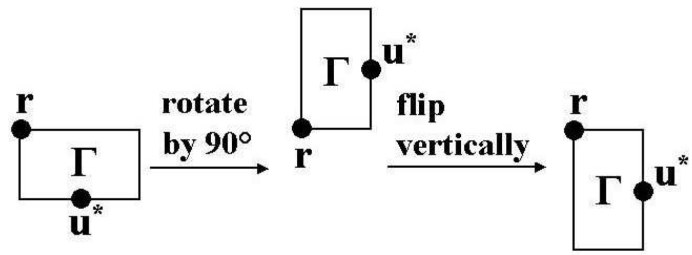


Figure 3.3.1: Rotating a drawing Γ by 90°, followed by flipping it vertically. Note that initially node u^* was located at the bottom boundary of Γ, but after the rotate operation, u^* is on the right boundary of Γ.

3.4 Tree Drawing Algorithm

Let T be a degree-d tree with a link node u^*, where $d = O(n^\delta)$ is a positive integer, $0 \leq \delta < 1/2$ is a constant, and n is the number of nodes in T. Let A and ε be two numbers such that $\delta/(1-\delta) < \varepsilon < 1$, and A is in the range $[n^{-\varepsilon}, n^{\varepsilon}]$. A is called the *desirable aspect ratio* for T.

Our tree drawing algorithm, called *DrawTree*, takes ε, A, and T as input, and uses a simple divide-and-conquer strategy to recursively construct a feasible drawing Γ of T, by performing the following actions at each recursive step (as we will prove later, Γ will fit inside a rectangle with area $O(n)$ and aspect ratio A):

- *Split Tree*: Split T into at most $2d-1$ partial trees by removing at most two nodes and their incident edges from it. Each partial tree has at most $n/2$ nodes. Based on the arrangement of these partial trees within T, we get two cases, which are shown in Figures 3.4.1 and 3.4.2, and described later in Section 3.4.1.
- *Assign Aspect Ratios*: Correspondingly, assign a desirable aspect ratio A_k to each

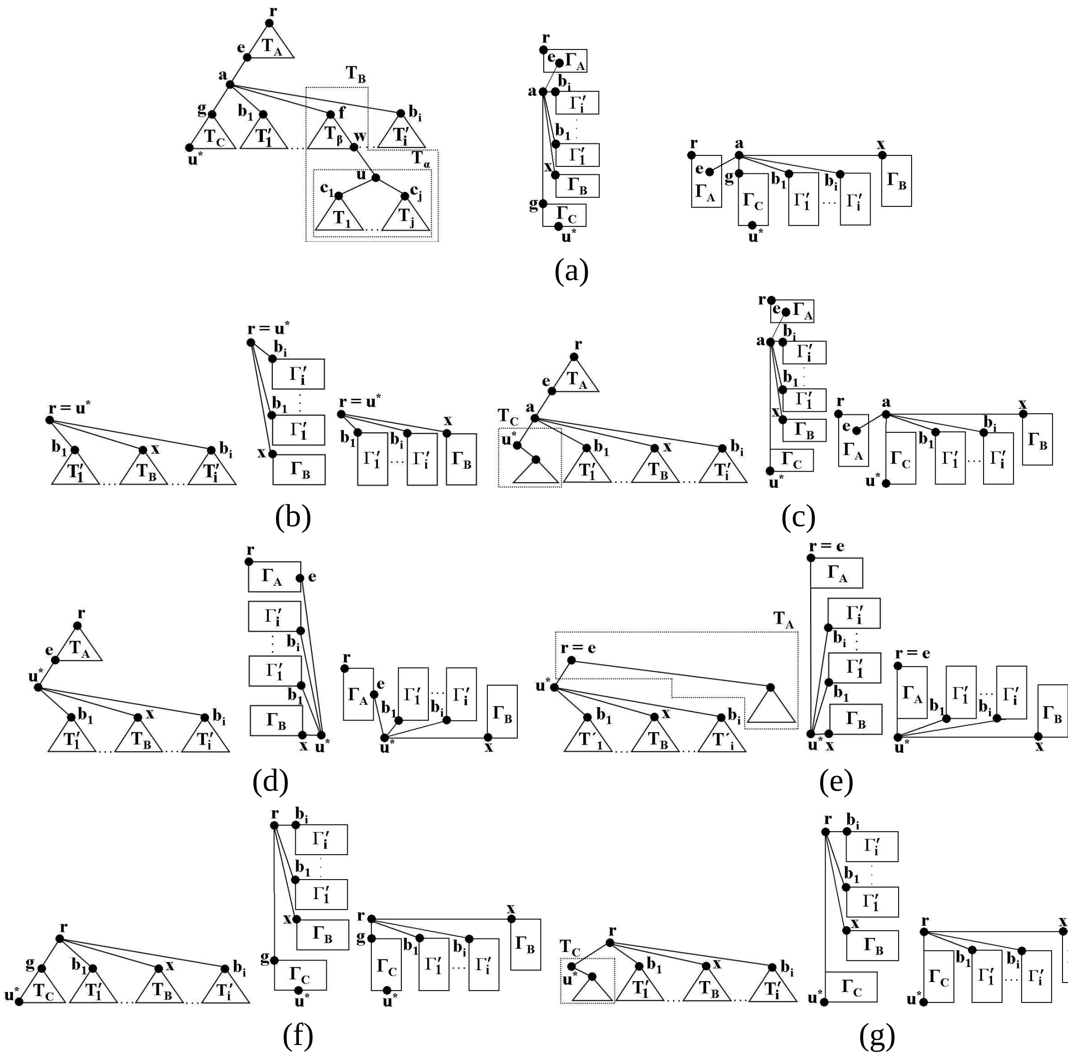


Figure 3.4.1: Drawing T in all the seven subcases of Case 1 (when the separator node u is not in the leftmost path of T): (a) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $g \neq u^*$, $0 \leq i \leq d-3$, (b) $T_A = \emptyset$, $T_C = \emptyset$, $0 \leq i \leq d-3$, (c) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $g = u^*$, $0 \leq i \leq d-3$, (d) $T_A \neq \emptyset$, $T_C = \emptyset$, $r \neq e$, $0 \leq i \leq d-3$, (e) $T_A \neq \emptyset$, $T_C = \emptyset$, $r = e$, $0 \leq i \leq d-3$, (f) $T_A = \emptyset$, $T_C \neq \emptyset$, $g \neq u^*$, $0 \leq i \leq d-3$, and (g) $T_A = \emptyset$, $T_C \neq \emptyset$, $g = u^*$, $0 \leq i \leq d-3$. For each subcase, we first show the structure of T for that subcase, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. Here, x is the same as f if $T_\beta \neq \phi$, and is the same as the root of T_α if $T_\beta = \phi$. In Subcases (a) and (c), for simplicity, e is shown to be in the interior of Γ_A, but actually, either it is the same as r, or if $A < 1$ ($A \geq 1$), then it is placed on the bottom (right) boundary of Γ_A. For simplicity, we have shown Γ_A, Γ_B, and Γ_C as identically sized boxes, but in actuality, they may have different sizes.

partial tree T_k. The value of A_k is based on the value of A, and the number of nodes in T_k.

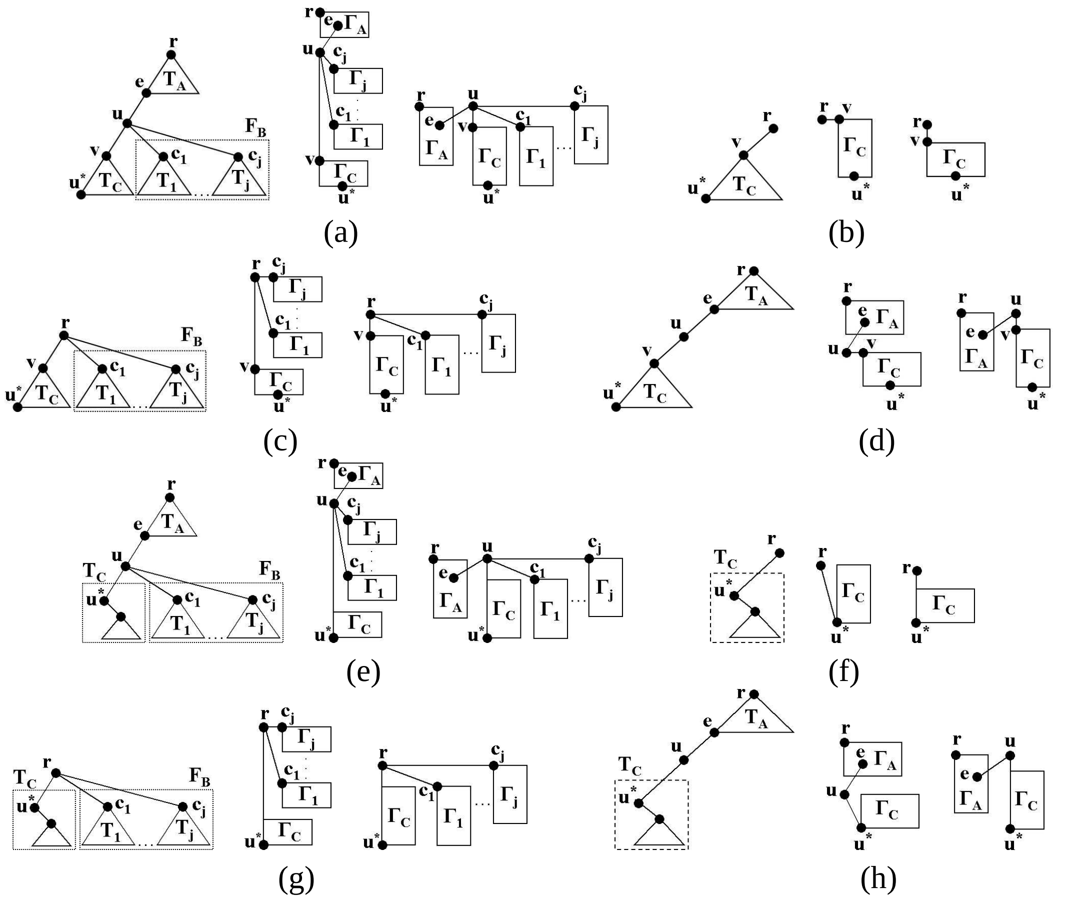


Figure 3.4.2: Drawing T in all the eight subcases of Case 2 (when the separator node u is in the leftmost path of T): (a) $T_A \neq \emptyset, T_C \neq \emptyset$, $v \neq u^*$, $1 \leq j \leq d-2$, (b) $T_A = \emptyset, T_C \neq \emptyset$, $v \neq u^*$, $j = 0$, (c) $T_A = \emptyset, T_C \neq \emptyset$, $v \neq u^*$, $1 \leq j \leq d-2$, (d) $T_A \neq \emptyset, T_C \neq \emptyset$, $v \neq u^*$, $j = 0$, (e) $T_A \neq \emptyset$, $T_B \neq \emptyset$, $v = u^*$, $1 \leq j \leq d-2$, (f) $T_A = \emptyset$, $T_B = \emptyset$, $v = u^*$, $j = 0$, (g) $T_A = \emptyset$, $T_B \neq \emptyset$, $v = u^*$, $1 \leq j \leq d-2$, and (h) $T_A \neq \emptyset$, $T_B = \emptyset$, $v = u^*$, $j = 0$. For each subcase, we first show the structure of T for that subcase, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. In Subcases (a) and (d), for simplicity, e is shown to be in the interior of Γ_A, but actually, either it is same as r, or if $A < 1$ ($A \geq 1$), then it is placed on the bottom (right) boundary of Γ_A. For simplicity, we have shown Γ_A, Γ_B, and Γ_C as identically sized boxes, but in actuality, they may have different sizes.

- *Draw Partial Trees*: Recursively construct a feasible drawing of each partial tree T_k with A_k as its desirable aspect ratio.

- *Compose Drawings*: Arrange the drawings of the partial trees, and draw the nodes and edges, that were removed from T to split it, such that the drawing Γ of T thus

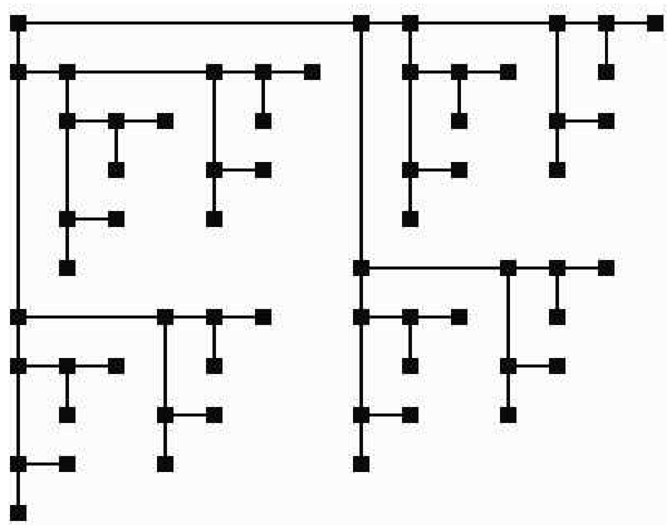


Figure 3.4.3: Drawing of the complete binary tree with 63 nodes constructed by Algorithm *DrawTree*, with $A = 1$ and $\varepsilon = 0.2$.

obtained is a feasible drawing. Note that the arrangement of these drawings is done based on the cases shown in Figures 3.4.1 and 3.4.2. In each case, if $A < 1$, then the drawings of the partial trees are stacked one above the other, and if $A \geq 1$, then they are placed side-by-side.

Figure 3.4.3 shows a drawing of a complete binary tree with 63 nodes constructed by Algorithm *DrawTree*, with $A = 1$ and $\varepsilon = 0.2$.

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

3.4.1 Split Tree

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

- Order the children of each node such that u^* becomes the leftmost node of T.

- Using Theorem 3.3.1, find a separator node u of T.

- Based on whether, or not, u is in the leftmost path of T, we get two cases:

  - *Case 1: The separator node u is not in the leftmost path of T.* We get seven subcases:

    (a) In the general case, T has the form as shown in Figure 3.4.1(a). In this figure:

    * r is the root of T,
    * $c_1, \ldots, c_j$ are the children of u, $0 \leq j \leq d-1$,
    * $T_1, \ldots, T_j$ are the trees rooted at $c_1, \ldots, c_j$ respectively, $0 \leq j \leq d-1$,
    * T_α is the subtree rooted at u,
    * w is the parent of u,
    * a is the last common node of the path $r \rightsquigarrow v$ and the leftmost path of T,
    * f is the child of a that is contained in the path $r \rightsquigarrow v$,
    * T_β is the maximal tree rooted at f that contains w but not u,
    * T_B is the tree consisting of the trees T_α and T_β, and the edge (w, u),
    * e is the parent of a,
    * g is the leftmost child of a,
    * T_A is the maximal tree rooted at r that contains e but not a,
    * T_C is the tree rooted at g,
    * $b_1, \ldots, b_i$ are the siblings of f and g,
    * $T'_1, \ldots, T'_i$ are the trees rooted at $b_1, \ldots, b_i$ respectively, $0 \leq i \leq d-3$, and

* $g \neq u^*$.

In addition to this general case, we get six special cases: (b) $T_A = \emptyset$, $T_C = \emptyset$, $0 \leq i \leq d-3$ (see Figure 3.4.1(b)), (c) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $g = u^*$, $0 \leq i \leq d-3$ (see Figure 3.4.1(c)), (d) $T_A \neq \emptyset$, $T_C = \emptyset$, $r \neq e$, $0 \leq i \leq d-3$ (see Figure 3.4.1(d)), (e) $T_A \neq \emptyset$, $T_C = \emptyset$, $r = e$, $0 \leq i \leq d-3$ (see Figure 3.4.1(e)), (f) $T_A = \emptyset$, $T_C \neq \emptyset$, $g \neq u^*$, $0 \leq i \leq d-3$ (see Figure 3.4.1(f)), and (g) $T_A = \emptyset$, $T_C \neq \emptyset$, $g = u^*$, $0 \leq i \leq d-3$ (see Figure 3.4.1(g)). (The reason we get these seven subcases is as follows: T_α has at least $n/2$ nodes in it because of Theorem 3.3.1. Hence $T_\alpha \neq \phi$, and so, $T_B \neq \phi$. Based on whether $T_A = \phi$ or not, $T_C = \phi$ or not, $g = u^*$ or not, and $r = e$ or not, we get totally sixteen cases. From these sixteen cases, we obtain the above seven subcases, by grouping some of these cases together. For example, the cases $T_A = \phi$, $T_C = \phi$, $d \neq u^*$, $r = u^*$, and $T_A = \phi$, $T_C = \phi$, $d \neq u^*$, $r \neq u^*$ are grouped together to give Case (a), i.e., $T_A = \phi$, $T_C = \phi$, $d \neq u^*$. So, Case (a) corresponds to 2 cases. Similarly, Cases (c), (d), (e), (f), and (g) correspond to 2 cases each, and Case (b) corresponds to 4 cases.) In each case, we remove nodes a and u, and their incident edges, to split T into at most $2d-1$ partial trees T_A, T_C, T_β, $T'_1, \ldots, T'_i$, $0 \leq i \leq d-3$, and $T_1, \ldots, T_j$, $0 \leq j \leq d-1$. We also designate e as the link node of T_A, w as the link node of T_β, and u^* as the link node of T_C. We arbitrarily select a leaf e_i of T'_i, $0 \leq i \leq d-3$, and a leaf e_j of T_j, $0 \leq j \leq d-1$, and designate them as the link nodes of T'_i and T_j, respectively.

– *Case 2: The separator node u is in the leftmost path of T.* We get eight subcases:

(a) In the general case, T has the form as shown in Figure 3.4.2(a). In this figure,

- r is the root of T,
- v is the leftmost child of u,
- $c_1, \ldots, c_j$ are the siblings of v, $1 \leq j \leq d-2$,
- $T_1, \ldots, T_j$ are the trees rooted at $c_1, \ldots, c_j$ respectively, $1 \leq j \leq d-2$,
- e is the parent of u,
- T_A is the maximal tree rooted at r that contains e but not u,
- T_C is the subtree of T rooted at v,
- F_B is the forest composed by trees $T_1, \ldots, T_j$, $1 \leq j \leq d-2$, and
- $v \neq u^*$.

In addition to the general case, we get the following seven special cases: (b) $T_A = \emptyset$, $j = 0$, $v \neq u^*$ (see Figure 3.4.2(b)), (c) $T_A = \emptyset$, $1 \leq j \leq d-2$, $v \neq u^*$ (see Figure 3.4.2(c)), (d) $T_A \neq \emptyset$, $j = 0$, $v \neq u^*$ (see Figure 3.4.2(d)), (e) $T_A \neq \emptyset$, $1 \leq j \leq d-2$, $v = u^*$ (see Figure 3.4.2(e)), (f) $T_A = \emptyset$, $j = 0$, $v = u^*$ (see Figure 3.4.2(f)), (g) $T_A = \emptyset$, $1 \leq j \leq d-2$, $v = u^*$ (see Figure 3.4.2(g)), and (h) $T_A \neq \emptyset$, $j = 0$, $v = u^*$ (see Figure 3.4.2(h)). (The reason we get these eight subcases is as follows: T_C has at least $n/2$ nodes in it because of Theorem 3.3.1. Hence, $T_C \neq \phi$. Based on whether $T_A = \phi$ or not, $F_B = \phi$ or not, and $v = u^*$ or not, we get the eight subcases given above.) In each case, we remove node u, and its incident edges, to split T into at most d partial trees T_A, T_C, and $T_1, \ldots, T_j$, $0 \leq j \leq d-2$. We also designate e as the link node of T_A, and u^* as the link node of T_C. We randomly select a leaf e_j of T_j and designate it as the link node of T_j, $0 \leq j \leq d-2$.

3.4.2 Assign Aspect Ratios

Let T_k be a partial tree of T, where for Case 1, T_k is either T_A, T_C, T_β, $T_1', \ldots, T_i'$, $0 \leq i \leq d-3$, or $T_1, \ldots, T_j$, $0 \leq j \leq d-1$, and for Case 2, T_k is either T_A, T_C, or $T_1, \ldots, T_j$, $0 \leq j \leq d-2$. Let n_k be number of nodes in T_k.

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

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

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

In Step *Assign Aspect Ratios*, we assign a desirable aspect ratio A_k to each nonempty T_k as follows: Let $x_k = n_k/n$.

- If $A \geq 1$: If T_k is a large partial tree of T, then $A_k = x_k A$, otherwise (i.e., if T_k is a small partial tree of T) $A_k = n_k^{-\varepsilon}$.
- If $A < 1$: If T_k is a large partial tree of T, then $A_k = A/x_k$, otherwise (i.e., if T_k is a small partial tree of T) $A_k = n_k^{\varepsilon}$.

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

3.4.3 Draw Partial Trees

First, we change the desirable aspect ratios assigned to T_A and T_β in some cases as follows: Suppose T_A and T_β get assigned desirable aspect ratios equal to m and p, respectively, where m and p are some positive numbers. In Subcase (d) of Case 1, and if $A \geq 1$, then in Subcases (a) and (c) of Case 1, and Subcases (a), (d), (e), and (h) of Case 2, we change the value of the desirable aspect ratio of T_A to $1/m$. In Case 1, if $A \geq 1$, we change the value of the desirable aspect ratio of T_β to $1/p$. We make these changes because, as explained later in Section 3.4.4, in these cases, we need to rotate the drawings of T_A and T_β by $90°$ during the *Compose Drawings* step. Drawing T_A and T_β with desirable aspect ratios $1/m$ and $1/p$, respectively, compensates for this rotation, and ensures that the drawings of T_A and T_β used to draw T have the desirable aspect ratios, m and p, respectively.

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

3.4.4 Compose Drawings

Let Γ_k denote the drawing of a partial tree T_k constructed in Step *Draw Partial Trees*. We now describe the construction of a feasible drawing Γ of T from the drawings of its partial trees in both Cases 1 and 2.

In Case 1, we first construct a feasible drawing Γ_α of the partial tree T_α by composing $\Gamma_1,\ldots,\Gamma_j$, $0 \leq j \leq d-1$, as shown in Figure 3.4.4, then construct a feasible drawing Γ_B of T_B by composing Γ_α and Γ_β as shown in Figure 3.4.5, and finally construct Γ by composing Γ_A, Γ_B, Γ_C, $\Gamma'_1,\ldots,\Gamma'_i$, $0 \leq i \leq d-3$, as shown in Figure 3.4.1.

Γ_α is constructed as follows (see Figure 3.4.4): If $A < 1$, place $\Gamma_j,\ldots,\Gamma_2,\Gamma_1$, $1 \leq j \leq d-1$, one above the other, in this order, separated by unit vertical distance, such that the left boundaries of $\Gamma_j,\ldots,\Gamma_2$ are aligned, and one unit to the right of the left boundary of Γ_1. Place u in the same vertical channel as c_1 and in the same horizontal channel as c_j. If $A \geq 1$, place $\Gamma_1,\Gamma_2,\ldots,\Gamma_j$, $1 \leq j \leq d-1$ in a left-to-right order, separated by unit horizontal distance, such that the top boundaries of $\Gamma_1,\Gamma_2,\ldots,\Gamma_{j-1}$ are aligned, and one unit below the top boundary of Γ_j. Place u in the same vertical channel as c_1 and in the same horizontal channel as c_j.

Γ_B is constructed as follows (see Figure 3.4.5):

- if $T_\beta \neq \emptyset$ (see Figure 3.4.5(a)) then, if $A < 1$, then place Γ_β one unit above Γ_α such that the left boundaries of Γ_β and Γ_α are aligned; otherwise (i.e., if $A \geq 1$), first rotate

Γ_β by 90° and then flip it vertically, then place Γ_β one unit to the left of Γ_α such that the top boundaries of Γ_β and Γ_α are aligned. Draw edge (w,y).

- Otherwise (i.e., if $T_\beta = \emptyset$), Γ_B is same as Γ_α (see Figure 3.4.5(b)).

Γ is constructed from Γ_A, Γ_B, Γ_C, $\Gamma'_1, \ldots, \Gamma'_i$, $0 \le i \le d-3$, as follows (see Figure 3.4.1): Let x be the root of T_B. Note that $x = f$ if $T_\beta \ne \emptyset$, and $x = u$ otherwise.

- In Subcase (a), as shown in Figure 3.4.1(a), if $A < 1$, stack Γ_A, $\Gamma'_i, \ldots, \Gamma'_1$, Γ_B, Γ_C one above the other, in this order, such that they are separated by unit vertical distance from each other, and the left boundaries of $\Gamma'_{i-1}, \ldots, \Gamma'_1, \Gamma_B$ are aligned with each other and are placed at unit horizontal distance to the right of the left boundaries of Γ_A and Γ_C. Place node a in the same vertical channel as r and g and in the same horizontal channel as b_i. If $A \ge 1$, then first rotate Γ_A by 90°, and then flip it vertically. Then, place Γ_A, Γ_C, $\Gamma'_1, \ldots, \Gamma'_i$, Γ_B from left-to-right in this order, separated by unit horizontal distances, such that the top boundaries of Γ_C, $\Gamma'_1, \ldots, \Gamma'_i$, are aligned, and are at unit vertical distance below the top boundaries of Γ_A and Γ_B. Then, move Γ_C down until u^* becomes the lowest node of Γ. Place node a in the same vertical channel as g and in the same horizontal channel as r and x. Draw edges (a,e), (a,x), (a,g), $(a,b_1), \ldots, (a,b_i)$.

- In Subcase (b), as shown in Figure 3.4.1(b), if $A < 1$, stack $\Gamma'_i, \ldots, \Gamma'_1$, Γ_B, one above the other, such that they are separated by unit vertical distance from each other, and their left boundaries are aligned. Place node r one unit above and left of the top boundary of Γ'_i. If $A \geq 1$, place $\Gamma'_1, \ldots, \Gamma'_i$, Γ_B in a left-to-right order such that they are separated by unit horizontal distance from each other, and their top boundaries are aligned. Place node r one unit above and left of the top boundary of Γ'_1. Draw edges $(r, b_1), \ldots, (r, b_i)$, (r, x).

- The drawing procedure for Subcase (c) is similar to the one in Subcase (a), except that we also flip Γ_C vertically (see Figure 3.4.1(c)).

- In Subcase (d), as shown in Figure 3.4.1(d), if $A < 1$, flip $\Gamma'_i, \ldots, \Gamma'_1$, Γ_B first vertically, and then horizontally, so that their roots get placed at their lower-right corners. Then, first rotate Γ_A by 90°, and then flip it vertically. Next, place Γ_A, $\Gamma'_i, \ldots, \Gamma'_1$, Γ_B one above the other, in this order, with unit vertical separation, such that their left boundaries are aligned, next move node e (which is the link node of T_A) to the right until it is either to the right of, or aligned with the rightmost boundary among $\Gamma'_i, \ldots, \Gamma'_1$, Γ_B (since Γ_A is a feasible drawing, by Property 2, as given in Section 3.3, moving e will not create any edge-crossings), and then place u^* in the same horizontal channel as x and one unit to the right of e. If $A \geq 1$, first rotate Γ_A by 90°, and then flip it vertically. Then flip $\Gamma'_i, \ldots, \Gamma'_1, \Gamma_B$ vertically. Then, place Γ_A, u^*, $\Gamma'_1, \ldots, \Gamma'_i$, Γ_B left-to-right in this order, separated by unit horizontal distances, such that the bottom boundaries of $\Gamma'_1, \ldots, \Gamma'_i$, are aligned, and are at unit vertical distance above the bottom boundary of Γ_B. Move Γ_B down until its bottom boundary is either aligned with or below the bottom boundary of Γ_A. Also, u^* is in the same horizontal channel with x. Draw edges (u^*, e), $(u^*, b_1), \ldots, (u^*, b_i)$, (u^*, x).

- In Subcase (e), as shown in Figure 3.4.1(e), if $A < 1$, first flip $\Gamma'_i, \ldots, \Gamma'_1$, Γ_B, vertically, then place Γ_A, $\Gamma'_i, \ldots, \Gamma'_1$, Γ_B one above the other, in this order, with unit vertical separation, such that the left boundaries of $\Gamma'_i, \ldots, \Gamma'_1$, Γ_B are aligned, and the left boundary of Γ_A is at unit horizontal distance to the left of the left boundary of Γ_B. Place u^* in the same vertical channel with r and in the same horizontal channel with x . If $A \geq 1$, then first flip $\Gamma'_1, \ldots, \Gamma'_i$, Γ_B vertically, next place Γ_A, $\Gamma'_1, \ldots, \Gamma'_i$, Γ_B in a left-to-right order at unit horizontal distance, such that the top boundaries Γ_A, $\Gamma'_1, \ldots, \Gamma'_i$ are aligned, and the bottom boundary of Γ_B is one unit below the bottom boundary of the drawing among Γ_A, $\Gamma'_1, \ldots, \Gamma'_i$ with greater height. Then, place u^* in the same vertical channel as r and in the same horizontal channel as r. Draw edges (u^*, r), $(u^*, b_1), \ldots, (u^*, b_i)$, (u^*, x). Note that, since Γ_A is a feasible drawing, by Property 3 (see Section 3.3), drawing (u^*, r) will not create any edge-crossings.

- The drawing procedure in Subcase (f) is similar to the one in Subcase (a), except that we do not have Γ_A here (see Figure 3.4.1(f)).

- The drawing procedure in Subcase (g) is similar to the one in Subcase (f), except that we also flip Γ_C vertically (see Figure 3.4.1(g).

In Case 2, we construct Γ by composing Γ_A, $\Gamma_1, \ldots, \Gamma_j$, Γ_C as follows (see Figure 3.4.2):

- The drawing procedures in Subcases (a) and (c) are similar to those in Subcases (a) and (f), respectively, of Case 1 (see Figures 3.4.2(a,c)).

- In Subcase (b) as shown in Figure 3.4.4(b), if $A < 1$, place u in the same horizontal channel and at one unit to the left of v; otherwise (i.e. $A \geq 1$), place u in the same vertical channel and at one unit above v. Draw edge (r,v).

- In Subcase (d), as shown in Figure 3.4.2(d), if $A > 1$, we place Γ_A above Γ_C, separated by unit vertical distance such that the left boundary of Γ_C is one unit to the right of the left boundary of Γ_A. Place u in the same vertical channel as r and in the same horizontal channel as v. If $A \geq 1$, then first rotate Γ_A by 90°, and then flip it vertically. Then, place Γ_A to the left of Γ_C, separated by unit horizontal distance, such that the top boundary of Γ_C is one unit below the top boundary of Γ_A. Then, move Γ_C down until u^* becomes the lowest node of Γ. Place u in the same vertical channel as v and in the same horizontal channel as r. Draw edges (u,v) and (u,e).

- The drawing procedures in Subcases (e), (f), (g), and (h) are similar to those in Subcases (a), (b), (c), and (d), respectively, (see Figures 3.4.2(e,f,g,h)), except that we also flip Γ_C vertically.

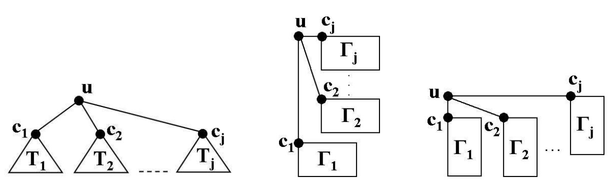


Figure 3.4.4: Drawing T_α. Here, we first show the structure of T_α, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. For simplicity, we have shown $\Gamma_1, \ldots, \Gamma_j$ as identically sized boxes, but in actuality, their sizes may be different.

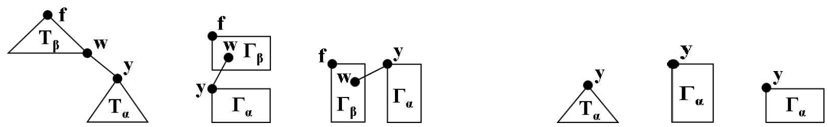


Figure 3.4.5: Drawing T_B when: (a) $T_\beta \neq \emptyset$, and (b) $T_\beta = \emptyset$. For each case, we first show the structure of T_B for that case, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. In Case (a), for simplicity, w is shown to be in the interior of Γ_β, but actually, it is either same as f, or if $A < 1$ ($A \geq 1$), then is placed on the bottom (right) boundary of Γ_β. For simplicity, we have shown Γ_β and Γ_α as identically sized boxes, but in actuality, their sizes may be different.

3.4.5 Proof of Correctness

Lemma 3.4.1 (Planarity) *Given an n-node degree-d tree T, where $d = O(n^\delta)$ is a positive integer and $0 \leq \delta < 1$, with a link node u^*, Algorithm* DrawTree *will construct a feasible drawing Γ of T.*

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

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

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

Hence, from Figure 3.4.4, it can be easily seen that the drawing Γ_α of T_α is also a feasible drawing.

From Figure 3.4.5, it can be easily seen that the drawing Γ_B of T_B is also a feasible drawing. Note that because Γ_β is a feasible drawing of T_β and w is its link node, w is either at the bottom of Γ_β (from Property 2, see Section 3.3), or at the top-left corner of Γ_β and no other edge or node of T_β is placed on, or crosses the vertical channel occupied by it (Properties 1 and 3, see Section 3.3). Hence, in Figure 3.4.5(a), in the case $A < 1$, drawing edge (w,x) will not cause any edge crossings. Also, in Figure 3.4.5(a), in the case $A \geq 1$, drawing edge (w,x) will not cause any edge crossings because after rotating Γ_β by 90° and flipping it vertically, w will either be at the right boundary of Γ_β (see Property 2), or at the top-left corner of Γ_β and no other edge or node of T_β will be placed on, or cross the horizontal channel occupied by it (see Properties 1 and 3).

Finally, by considering each of the seven subcases shown in Figure 3.4.1 one-by-one, we can show that Γ is also a feasible drawing of T:

- *Subcase (a):* See Figure 3.4.1(a). Γ_A is a feasible drawing of T_A and e is the link node of T_A. Hence, e is either at the bottom of Γ_A (from Property 2), or is at the top-left corner of Γ_A, and no other edge or node of T_A is placed on, or crosses the horizontal and vertical channels occupied by it (from Properties 1 and 3). Hence, in the case $A < 1$, drawing edge (e,a) will not create any edge-crossings, and Γ will also be a feasible drawing of T. In the case $A \geq 1$ also, drawing edge (e,a) will not create any edge-crossings because after rotating Γ_A by 90° and flipping it vertically, e will either be at the right boundary of Γ_A (see Property 2), or at the top-left corner of Γ_β and no other edge or node of T_A will be placed on, or cross the horizontal channel occupied by it (see Properties 1 and 3). Thus, for the case $A \geq 1$ also, Γ will also be a feasible drawing of T.

- *Subcase (b):* See Figure 3.4.1(b). Because $\Gamma'_1, \ldots, \Gamma'_i$, Γ_B are feasible drawings of $T'_1, \ldots, T'_i$, T_B respectively, it is straightforward to see that Γ is also a feasible drawing of T for both the cases when $A < 1$ and $A \geq 1$.

- *Subcase (c):* See Figure 3.4.1(c). The proof is similar to the one for Subcase (a).

- *Subcase (d):* See Figure 3.4.1(d). Γ_A is a feasible drawing of T_A, e is the link node of T_A, and $e \neq r$. Hence, from Property 2, e is located at the bottom of Γ_A. Rotating Γ_A by 90° and flipping it vertically will move e to the right boundary of Γ_A. Moving e to the right until it is either to the right of, or aligned with the right boundary of Γ_B will not cause any edge-crossings because of Property 2. It can be easily seen that in both the cases, $A < 1$ and $A \geq 1$, drawing edge (e, u^*) does not create any edge-crossings, and Γ is a feasible drawing of T.

- *Subcase (e):* See Figure 3.4.1(e). Γ_A is a feasible drawing of T_A, e is the link node of T_A, and $e = r$. Hence, from Properties 1 and 3, e is at the top-left corner of Γ_A, and no other edge or node of T_A is placed on, or crosses the horizontal and vertical channels occupied by it. Hence, in both the cases, $A < 1$ and $A \geq 1$, drawing edge (e, u^*) will not create any edge-crossings, and Γ is a feasible drawing of T.

- *Subcase (f):* See Figure 3.4.1(f). It is straightforward to see that Γ is a feasible drawing of T for both the cases when $A < 1$ and $A \geq 1$.

- *Subcase (g):* See Figure 3.4.1(g). Γ_C is a feasible drawing of T_C, u^* is the link node of T_C, and u^* is also the root of T_C. Hence, from Properties 1 and 3, u^* is at the top-left corner of Γ_C, and no other edge or node of T_C is placed on, or crosses the horizontal and vertical channels occupied by it. Flipping Γ_C vertically will move u^* to the bottom-left corner of Γ_C and no other edge or node of T_C will be on or crosses the vertical channel occupied by it. Hence, drawing edge (r, u^*) will not create any edge-crossings, and Γ will be a feasible drawing of T.

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

Lemma 3.4.2 (Time) *Given an n-node degree-d tree T, where $d = O(n^\delta)$ is a positive integer and $0 \leq \delta < 1$, with a link node u^*, Algorithm* DrawTree *will construct a drawing Γ of T in $O(n \log n)$ time.*

Proof: From Theorem 3.3.1, each partial tree into which Algorithm *DrawTree* would split T will have at most $n/2$ nodes in it. Hence, it follows that the depth of the recursion for Algorithm *DrawTree* is $O(\log n)$. At the first recursive level, the algorithm will split T into partial trees, assign aspect ratios to the partial trees and compose the drawings of the partial trees to construct a drawing of T. At the next recursive level, it will split all of these partial trees into smaller partial trees, assign aspect ratios to these smaller partial trees, and compose the drawings of these smaller partial trees to construct the drawings of all the partial trees. This process will continue until the bottommost recursive level is reached. At each recursive level, the algorithm takes $O(m)$ time to split a tree with m nodes into partial trees, assign aspect ratios to the partial trees, and compose the drawings of partial trees to construct a drawing of the tree. At each recursive level, the total number of nodes in all the trees that the algorithm considers for drawing is at most n. Hence, at each recursive level, the algorithm totally spends $O(n)$ time. Hence, the running time of the algorithm is $O(n) \cdot O(\log n) = O(n \log n)$.

□

In Lemma 3.4.4 given below, we prove that the algorithm will draw a degree-d tree, where $d = O(n^{\delta})$ is a positive integer and $0 \leq \delta < 1$, in $O(n)$ area.

Lemma 3.4.3 *Let R be a rectangle with area D and aspect ratio A. Let W and H be the width and height, respectively, of R. Then, $W = \sqrt{AD}$ and $H = \sqrt{D/A}$.*

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

Lemma 3.4.4 (Area) *Let T be an n-node degree-d tree, where $d = O(n^{\delta})$ is a positive integer and $0 \leq \delta < 1$, with a link node u^*. Let ε and A be two numbers such that $\delta/(1-\delta) < \varepsilon < 1$, and A is in the range $[n^{-\varepsilon}, n^{\varepsilon}]$. Given T, ε, and A as input, Algorithm* DrawTree *will construct a drawing Γ of T that can fit inside a rectangle R with $O(n)$ area and aspect ratio A.*

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

We now give the proof for the case when $A \geq 1$ (the proof for the case $A < 1$ is symmetrical). Algorithm *DrawTree* will split T into at most $2d-1$ partial trees. Let T_k be a non-empty partial tree of T, where T_k is one of $T_A, T_C, T_\beta, T'_1, \ldots, T'_i$, $0 \leq i \leq d-3$, $T_1, \ldots, T_j$, $0 \leq j \leq d-1$, in Case 1, and is one of $T_A, T_C, T_1, \ldots, T_j$, $0 \leq j \leq d-2$, in Case 2. Let n_k be the number of nodes in T_k, and let $x_k = n_k/n$. Let $P_k = c_1 n - c_2 n^\beta / x_k^{1-\beta}$. From Theorem 3.3.1, it follows that $n_k \leq n/2$, and hence, $x_k \leq 1/2$. Hence, $P_k \leq c_1 n - c_2 n^\beta/(1/2)^{1-\beta} = c_1 n - c_2 n^\beta 2^{1-\beta}$. Let $P' = c_1 n - c_2 n^\beta 2^{1-\beta}$. Thus, $P_k \leq P'$.

From the inductive hypothesis, Algorithm *DrawTree* will construct a drawing Γ_k of T_k that can fit inside a rectangle R_k with aspect ratio A_k and area $D(n_k)$, where A_k is as defined in Section 3.4.2, and $D(n_k) \leq c_1 n_k - c_2 n_k^\beta$. Since $x_k = n_k/n$, $D(n_k) \leq c_1 n_k - c_2 n_k^\beta = c_1 x_k n - c_2 (x_k n)^\beta = x_k(c_1 n - c_2 n^\beta / x_k^{1-\beta}) = x_k P_k \leq x_k P'$.

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

- *T_k is a small partial tree of T:* Then, $n_k < (n/A)^{1/(1+\varepsilon)}$, and also, as explained in Section 3.4.2, $A_k = 1/n_k^\varepsilon$. From Lemma 3.4.3, $W_k = \sqrt{A_k D(n_k)} \leq \sqrt{(1/n_k^\varepsilon)(x_k P')} = \sqrt{(1/n_k^\varepsilon)(n_k/n)P'} = \sqrt{n_k^{1-\varepsilon} P'/n}$. Since $n_k < (n/A)^{1/(1+\varepsilon)}$, $W_k < \sqrt{(n/A)^{(1-\varepsilon)/(1+\varepsilon)} P'/n} = \sqrt{(1/A^{(1-\varepsilon)/(1+\varepsilon)}) P'/n^{2\varepsilon/(1+\varepsilon)}} \leq \sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}}$ since $A \geq 1$.

  From Lemma 3.4.3, $H_k = \sqrt{D(n_k)/A_k} \leq \sqrt{x_k P'/(1/n_k^\varepsilon)} = \sqrt{(n_k/n)P' n_k^\varepsilon} = \sqrt{n_k^{1+\varepsilon} P'/n}$. Since $n_k < (n/A)^{1/(1+\varepsilon)}$, $H_k < \sqrt{(n/A)^{(1+\varepsilon)/(1+\varepsilon)} P'/n} = \sqrt{(n/A)P'/n} = \sqrt{P'/A}$.

- *T_k is a large partial tree of T:* Then, as explained in Section 3.4.2, $A_k = x_k A$. From Lemma 3.4.3, $W_k = \sqrt{A_k D(n_k)} \leq \sqrt{x_k A x_k P'} = x_k \sqrt{AP'}$.

  From Lemma 3.4.3, $H_k = \sqrt{D(n_k)/A_k} \leq \sqrt{x_k P'/(x_k A)} = \sqrt{P'/A}$.

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

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

and

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

(because $\sum_{T_k \text{ is a large partial tree}} x_k \leq 1$, and T is split into at most $2d-1$ partial trees)

R' does not have aspect ratio equal to A, but it is contained within a rectangle R with aspect ratio A, area $D(n)$, width W, and height H, where

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

and

$$H = \sqrt{P'/A} + 2 + ((2d-1)/A)\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + 1/A$$

Hence, $D(n) = WH = (\sqrt{AP'} + (2d-1)\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + 1 + 2A)(\sqrt{P'/A} + 2 + ((2d-1)/A)\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + 1/A) = P' + 2(2d-1)P'/\sqrt{An^{2\varepsilon/(1+\varepsilon)}} + 4\sqrt{AP'} + (2d-1)^2 P'/(An^{2\varepsilon/(1+\varepsilon)}) + 4(2d-1)\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + 4A + 4 + 1/A + 2\sqrt{P'/A} + 2(2d-1)\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}}/A$.

Since, $1 \le A \le n^\varepsilon$, we have that

$$\begin{aligned} D(n) &\le P' + c_3 dP'/\sqrt{n^{2\varepsilon/(1+\varepsilon)}} + c_4\sqrt{n^\varepsilon P'} + c_5 d^2 P'/n^{2\varepsilon/(1+\varepsilon)} + c_6 P'/n^{2\varepsilon/(1+\varepsilon)} \\ &\quad + c_7 d\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + c_8 n^\varepsilon + c_9 + c_{10}\sqrt{P'} \end{aligned}$$

where $c_3, c_4, \ldots, c_{10}$ are some constants.

Since $P' < c_1 n$,

$$\begin{aligned} D(n) &< P' + c_3 d c_1 n/\sqrt{n^{2\varepsilon/(1+\varepsilon)}} + c_4\sqrt{n^\varepsilon c_1 n} + c_5 d^2 c_1 n/n^{2\varepsilon/(1+\varepsilon)} + c_6 c_1 n/n^{2\varepsilon/(1+\varepsilon)} \\ &\quad + c_7 d\sqrt{c_1 n/n^{2\varepsilon/(1+\varepsilon)}} + c_8 n^\varepsilon + c_9 + c_{10}\sqrt{c_1} n^{1/2} \\ &= P' + c_3 d c_1 n^{1/(1+\varepsilon)} + c_4\sqrt{c_1} n^{(1+\varepsilon)/2} + c_5 d^2 c_1 n^{(1-\varepsilon)/(1+\varepsilon)} + c_6 c_1 n^{(1-\varepsilon)/(1+\varepsilon)} \\ &\quad + c_7 d\sqrt{c_1} n^{(1-\varepsilon)/(2(1+\varepsilon))} + c_8 n^\varepsilon + c_9 + c_{10}\sqrt{c_1} n^{1/2} \\ &\le P' + c_{11} n^{(1+\varepsilon)/2} + c_{12} d n^{1/(1+\varepsilon)} + c_{13} d^2 n^{(1-\varepsilon)/(1+\varepsilon)} \end{aligned}$$

where c_{11}, c_{12}, and c_{13} are large enough constants (because, since $0 \le \delta/(1-\delta) < \varepsilon < 1$, $(1-\varepsilon)/(2(1+\varepsilon)) < (1-\varepsilon)/(1+\varepsilon) < 1/(1+\varepsilon)$, $\varepsilon < (1+\varepsilon)/2$, and $1/2 < (1+\varepsilon)/2$).

Because $d = O(n^{\delta})$, for a large enough constant n_0, there exist constants c_{14} and c_{15} such that for all $n \geq n_0$, $D(n) \leq P' + c_{11}n^{(1+\varepsilon)/2} + c_{14}n^{\delta+1/(1+\varepsilon)} + c_{15}n^{2\delta+(1-\varepsilon)/(1+\varepsilon)}$.

$P' = c_1 n - c_2 n^{\beta} 2^{1-\beta} = c_1 n - c_2 n^{\beta}(1+c_{16})$, where c_{16} is a constant such that $1 + c_{16} = 2^{1-\beta}$.

Hence, $D(n) \leq c_1 n - c_2 n^{\beta}(1+c_{16}) + c_{11}n^{(1+\varepsilon)/2} + c_{14}n^{\delta+1/(1+\varepsilon)} + c_{15}n^{2\delta+(1-\varepsilon)/(1+\varepsilon)} = c_1 n - c_2 n^{\beta} - (c_{16}n^{\beta} - c_{11}n^{(1+\varepsilon)/2} - c_{14}n^{\delta+1/(1+\varepsilon)} - c_{15}n^{2\delta+(1-\varepsilon)/(1+\varepsilon)})$. Thus, for a large enough constant n_0, and large enough β, where $1 > \beta > \max\{(1+\varepsilon)/2, \delta+1/(1+\varepsilon), 2\delta+(1-\varepsilon)/(1+\varepsilon)\}$, for all $n \geq n_0$, $c_{16}n^{\beta} - c_{11}n^{(1+\varepsilon)/2} - c_{14}n^{\delta+1/(1+\varepsilon)} - c_{15}n^{2\delta+(1-\varepsilon)/(1+\varepsilon)} \geq 0$, and hence $D(n) \leq c_1 n - c_2 n^{\beta}$. Note that because $\varepsilon > \delta/(1-\delta)$, $\delta + 1/(1+\varepsilon) < 1$ and $2\delta + (1-\varepsilon)/(1+\varepsilon) < 1$, and because $\varepsilon < 1$, $(1+\varepsilon)/2 < 1$.

The proof for the case $A < 1$ uses the same reasoning as for the case $A \geq 1$. With T_k, R_k, W_k, H_k, R', W', H', R, W, and H defined as above, and A_k as defined in Section 3.4.2, we get the following values for W_k, H_k, W', H', W, H, and $D(n)$:

$$
\begin{aligned}
W_k &\leq \sqrt{AP'} \\
H_k &\leq \sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} \text{ if } T_k \text{ is a small partial tree} \\
&\leq x_k\sqrt{P'/A} \text{ if } T_k \text{ is a large partial tree} \\
W' &\leq \sqrt{AP'} + 2 \\
H' &\leq \sqrt{P'/A} + (2d-1)\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + 1 \\
W &\leq \sqrt{AP'} + 2 + (2d-1)A\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + A \\
H &\leq \sqrt{P'/A} + (2d-1)\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + 1 + 2/A \\
D(n) &\leq P' + c_{11}n^{(1+\varepsilon)/2} + c_{14}n^{\delta+1/(1+\varepsilon)} + c_{15}n^{2\delta+(1-\varepsilon)/(1+\varepsilon)}
\end{aligned}
$$

where c_{11}, c_{14}, and c_{15} are the same constants as in the case $A \geq 1$. Therefore, $D(n) \leq c_1 n - c_2 n^\beta$ for $A < 1$ too. (Notice that in the values that we get above for W_k, H_k, W', H', W, and H, if we replace A by $1/A$, exchange W_k with H_k, exchange W' with H', and exchange W with H, we will get the same values for W_k, H_k, W', H', W, and H as in the case $A \geq 1$. This basically reflects the fact that the cases $A \geq 1$ and $A < 1$ are symmetrical to each other.) □

Theorem 3.4.1 (Main Theorem) *Let T be an n-node degree-d tree, where $d = O(n^\delta)$ is a positive integer and $0 \leq \delta < 1/2$ is a constant. Given any number A, where $n^{-\alpha} \leq A \leq n^\alpha$, for some constant α, where $0 \leq \alpha < 1$, we can construct in $O(n \log n)$ time, a planar straight-line grid drawing of T with $O(n)$ area, and aspect ratio A.*

Proof: Let ε be a constant such that $n^{-\varepsilon} \leq A \leq n^\varepsilon$ and $\delta/(1-\delta) < \varepsilon < 1$. Designate any leaf of T as its link node. Construct a drawing Γ of T in R by calling Algorithm *DrawTree* with T, A and ε as input. From Lemmas 3.4.1, 3.4.2, and 3.4.4, Γ will be a planar straight-line grid drawing of T contained entirely within a rectangle with $O(n)$ area, and aspect ratio A. □

Corollary 3.4.1 *Let T be an n-node degree-d tree, where $d = O(n^\delta)$ is a positive integer and $0 \leq \delta < 1/2$ is a constant.. We can construct in $O(n \log n)$ time, a planar straight-line grid drawing of T with optimal (equal to $O(n)$) area, and optimal aspect ratio (equal to 1).*

Proof: Immediate from Theorem 3.4.1, with $A = 1$. □

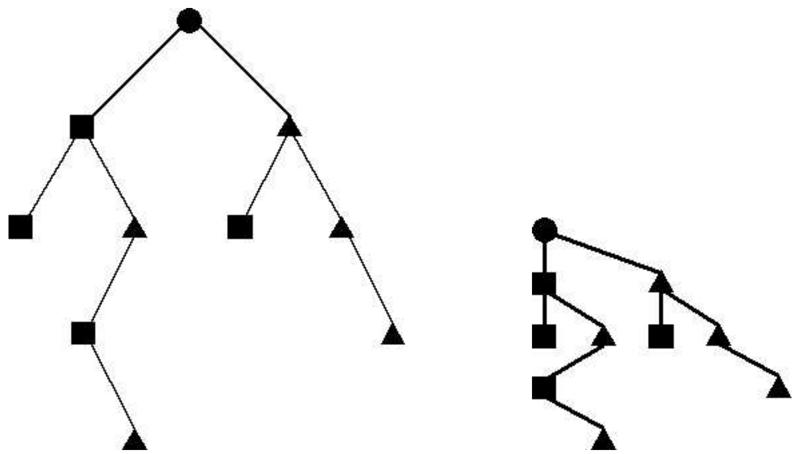
Chapter 4

Area-Efficient Order-Preserving Planar Straight-line Grid Drawings of Ordered Trees

4.1 Introduction

An *ordered tree* T is one with a prespecified counterclockwise ordering of the edges incident on each node. Ordered trees arise commonly in practice. Examples of ordered trees include binary search trees, arithmetic expression trees, BSP-trees, B-trees, and range-trees.

An *order-preserving drawing* of T is one in which the counterclockwise ordering of the edges incident on a node is the same as their prespecified ordering in T. A *planar drawing* of T is one with no edge-crossings. An *upward drawing* of T is one, where each node is placed either at the same *y*-coordinate as, or at a higher *y*-coordinate than the *y*-coordinates of its children. A *straight-line drawing* of T is one, where each edge is drawn as a single line-segment. A *grid drawing* of T is one, where each node is assigned integer *x*- and *y*-coordinates. For example, the drawing in Figure 4.1.1(b) is an order-preserving upward straight-line grid drawing of the tree in Figure 4.1.1(a).



(a) (b)

Figure 4.1.1: (a) A binary tree T. (b) An order-preserving upward planar straight-line grid drawing Γ of T. Here, the circle-shaped node is the root of T, the square-shaped nodes are left children of their respective parents, and the triangle-shaped nodes are right children of their respective parents.

Ordered trees are generally drawn using order-preserving planar straight-line grid drawings, as any undergraduate textbook on data-structures will show. An upward drawing is desirable because it makes it easier for the user to determine the parent-child relationships between the nodes.

We investigate the area-requirement of the order-preserving planar straight-line grid drawings of ordered trees, and present several results: Let T be an ordered tree with n nodes.

Result 1: We show that T admits an order-preserving planar straight-line grid drawing with $O(n \log n)$ area, $O(n)$ height, and $O(\log n)$ width, which can be constructed in $O(n)$ time.

Result 2: If T is a binary tree, then we show stronger results:

> ***Result 2a:*** T admits an order-preserving planar straight-line grid drawing with $O(n \log\log n)$ area, $O((n/\log n)\log\log n)$ height, and $O(\log n)$ width, which can be constructed in $O(n)$ time.
>
> ***Result 2b:*** T admits an order-preserving *upward* planar straight-line grid drawing with *optimal* $O(n \log n)$ area, $O(n)$ height, and $O(\log n)$ width, which can be constructed in $O(n)$ time.

An important issue is that of the *aspect ratio* of a drawing D. Let E be the smallest rectangle, with sides parallel to x and y-axis, respectively, enclosing D. The *aspect ratio* of D is defined as the ratio of the larger and smaller dimensions of E, i.e., if h and w are the height and width, respectively, of E, then the aspect ratio of D is equal to $\max\{h,w\}/\min\{h,w\}$. It is important to give the user control over the aspect ratio of a drawing because this will allow her to fit the drawing in an arbitrarily-shaped window defined by her application. It also allows the drawing to fit within display-surfaces with predefined aspect ratios, such as a computer-screen and a sheet of paper. We consider the problem of drawing binary trees with arbitrary aspect ratio, and prove the following result:

Result 3: Let T be a binary tree with n nodes. Let $2 \leq A \leq n$ be any user-specified number.

T admits an order-preserving planar straight-line grid drawing Γ with width $O(A+\log n)$, height $O((n/A)\log A)$, and area $O((A+\log n)(n/A)\log A) = O(n\log n)$, which can be constructed in $O(n)$ time.

Also note that [17] shows an n-node binary tree that requires $\Omega(n)$ height and $\Omega(\log n)$ width in any order-preserving upward planar grid drawing. Hence, the $O(n)$ height and $O(\log n)$ width achieved by Result *2b* is optimal in the worst case.

4.2 Previous Results

Throughout this section, n denotes the number of nodes in a tree. The *degree* of a tree is equal to the maximum number of edges incident on a node.

In spite of the natural appeal of order-preserving drawings, quite surprisingly, little work has been done on optimizing the area of such drawings. The previous best upper bound on the area-requirement of an order-preserving planar upward straight-line grid drawing of a tree was $O(n^{1+\varepsilon})$, where $\varepsilon > 0$ is any user-defined constant, which was shown in [4]. [34] has shown that a special class of balanced binary trees, which includes k-balanced, red-black, $BB[\alpha]$, and (a,b) trees, admits order-preserving planar upward straight-line grid drawings with area $O(n(\log\log n)^2)$. [6], [7], and [42] give order-preserving planar upward straight-line grid drawings of complete binary trees, logarithmic, and Fibonacci trees, respectively, with area $O(n)$. [17] has given an upper bound of $O(n\log n)$ on order-preserving planar upward *polyline* grid drawings. (A polyline drawing is one, where each edge is drawn as a connected sequence of one *or more* line-segments.)

As for the lower bound on the area-requirement of order-preserving drawings, [17] has shown a lower bound of $\Omega(n \log n)$ for order-preserving planar upward grid drawings. There is no known lower bound for non-upward order-preserving planar grid drawings other than the trivial $\Omega(n)$ bound.

We are not aware of any non-trivial results on order-preserving drawings of trees with user-defined arbitrary aspect-ratios. However, a few results are available on non-order-preserving drawings. [17] shows that any tree with degree d admits a non-order-preserving planar upward polyline grid drawing with height $h = O(n^{1-\alpha})$ and area $O(n + dh \log n)$, where $0 < \alpha < 1$ is any user-specified constant. This result implies that any tree with degree $O(n^{\beta})$, where $0 \leq \beta < 1$ is any constant, can be drawn in this fashion in $O(n)$ area with aspect ratio $O(n^{\gamma})$, where γ is any user-defined constant, such that $\max\{0, 2\beta - 1\} < \gamma < 1$. [3] shows that any binary tree admits a non-order-preserving upward planar straight-line *orthogonal* (each edge drawn as a horizontal or vertical line-segment) grid drawing with area $O(n \log n)$, and any user-specified aspect ratio in the range $[1, n/\log n]$. They also prove that the $O(n \log n)$ bound on area is also optimal for such drawings by showing that for any n and a number $2 \leq A \leq n$, there exists a binary tree with n nodes that requires $\Omega(n \log n)$ area in any upward planar straight-line orthogonal grid drawing with aspect ratio in the range $[1, n/\log n]$. [3] and [34] show that any binary tree admits a non-order-preserving non-upward planar straight-line orthogonal grid drawing with height $O(n/A) \log A$, width $O(A + \log n)$, where $2 \leq A \leq n$ is any user-specified number. This result also implies that we can draw any binary tree in this fashion in area $O(n\log\log n)$ (by setting $A = \log n$).

[18] shows that any binary tree admits a non-order-preserving planar non-upward straight-line drawing with area $O(n)$, and any user-specified aspect ratio in the range $[1, n^{\alpha}]$, where $0 \leq \alpha < 1$ is any constant. [22] extends this result to trees with degree $O(n^{\delta})$, where $0 \leq \delta < 1/2$ is any constant.

As for other kinds of drawings (non-order-preserving and with fixed aspect ratio), a variety of results are available. See [11] for a survey on these results.

Table 4.2.1 compares our results with the previously known results.

Tree Type	Drawing Type	Area	Aspect Ratio	Reference
Special Balanced Trees such as Red-black	Upward Order-preserving	$O(n(\log\log n)^2)$	$n/\log^2 n$	[35]
Binary	Upward Non-order-preserving	$O(n\log\log n)$	$(n\log\log n)/\log^2 n$	[35]
		$O(n\log n)$	$[1, n/\log n]$	[3]
	Upward Order-preserving	$O(n^{1+\varepsilon})$	$n^{1-\varepsilon}$	[4]
		$O(n\log n)$	$n/\log n$	this chapter
	Non-upward Non-order-preserving	$O(n)$	$[1, n^{\alpha}]$	[18]
	Non-upward Order-preserving	$O(n^{1+\varepsilon})$	$n^{1-\varepsilon}$	[4]
		$O(n\log n)$	$[1, n/\log n]$	this chapter
		$O(n\log\log n)$	$(n\log\log n)/\log^2 n$	this chapter
General	Non-upward Order-preserving	$O(n^{1+\varepsilon})$	$n^{1-\varepsilon}$	[4]
		$O(n\log n)$	$n/\log n$	this chapter

Table 4.2.1: Bounds on the areas and aspect ratios of various kinds of planar straight-line grid drawings of an n-node tree. Here, α and ε are user-defined constants, such that $0 \leq \alpha < 1$ and $0 < \varepsilon < 1$. $[a, b]$ denotes the range $a \ldots b$.

The paper based on this Chapter has been presented in [19].

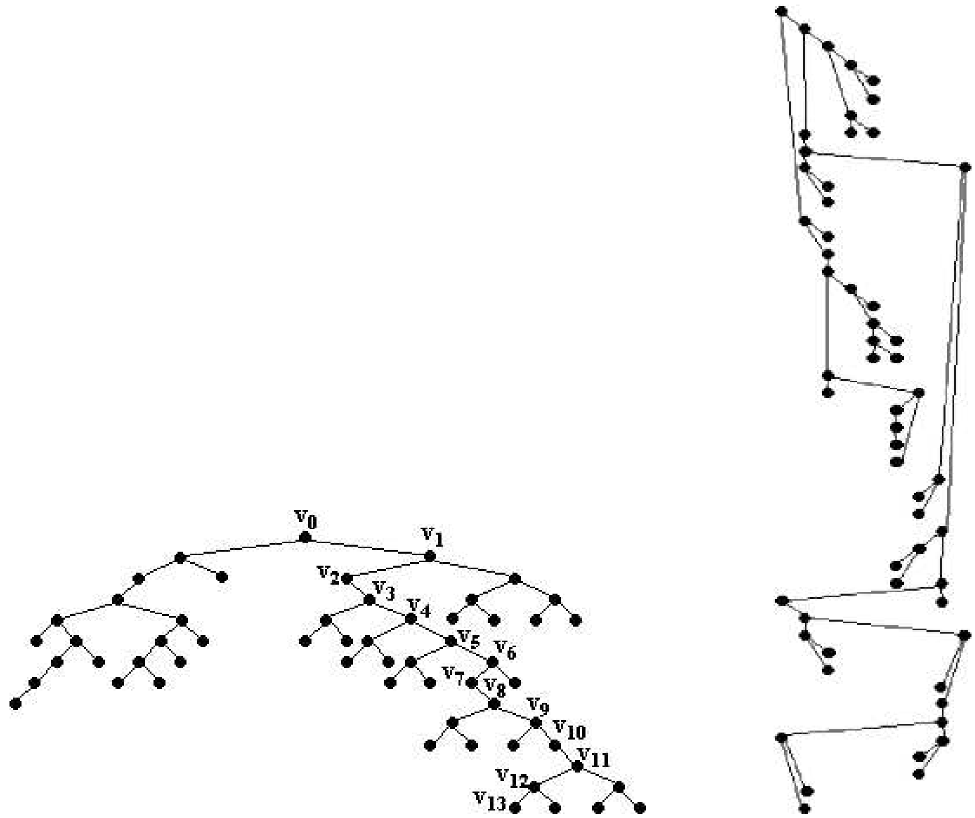
4.3 Preliminaries

We assume a 2-dimensional Cartesian space. We assume that this space is covered by an infinite rectangular grid, consisting of horizontal and vertical channels.

A *left-corner* drawing of an ordered tree T is one, where no node of T is to the left of, or above the root of T. The *mirror-image* of T is the ordered tree obtained by reversing the counterclockwise order of edges around each node. Let R be a rectangle with sides parallel to the x- and y-axis, respectively. The *height* (*width*) of R is equal to the number of grid-points with the same x-coordinate (y-coordinate) contained within R. The *area* of R is equal to the number of grid-points contained within R. The *enclosing rectangle* E of a drawing D is the smallest rectangle with sides parallel to the x- and y-axis covering the entire drawing. The *height* h, *width* w, and *area* of D is equal to the height, width, and area, respectively, of E. The *aspect ratio* of D is equal to $\max\{h,w\}/\min\{h,w\}$.

A *subtree* rooted at a node v of an ordered tree T is the maximal tree consisting of v and all its descendents. A *partial* tree of T is a connected subgraph of T. A *spine* of T is a path $v_0 v_1 v_2 \ldots v_m$, where $v_0, v_1, v_2, \ldots, v_m$ are nodes of T, that is defined recursively as follows (see Figure 4.3.1):

- v_0 is the same as the root of T;
- v_{i+1} is the child of v_i, such that the subtree rooted at v_{i+1} has the maximum number of nodes among all the subtrees that are rooted at the children of v_i.



(a) (b)

Figure 4.3.1: (a) A binary tree T with spine $v_0 v_1 \ldots v_{13}$. (b) The order-preserving planar upward straight-line grid drawing of T constructed by *Algorithm BT-Ordered-Draw*.

The concept of a spine has been used implicitly by several tree drawing algorithms, including those of [3, 4, 17]. In particular, [4] uses it to construct order-preserving drawings. However, our algorithms typically draw the spine as a more zig-zagging path than the algorithms of [4]. (In fact, some algorithms of [4] draw the spine completely straight as a single line-segment.) This enables our algorithms to draw a tree more compactly than the algorithms of [4].

4.4 Drawing Binary Trees

We now give our drawing algorithm for constructing order-preserving planar upward straight-line grid drawings of binary trees. In an ordered binary tree, each node has at most two children, called its *left* and *right* children, respectively.

Our drawing algorithm, which we call *Algorithm BT-Ordered-Draw*, uses the divide-and-conquer paradigm to draw an ordered binary tree T. In each recursive step, it breaks T into several subtrees, draws each subtree recursively, and then combines their drawings to obtain an upward left-corner drawing $D(T)$ of T. We now give the details of the actions performed by the algorithm to construct $D(T)$. Note that during its working, the algorithm will designate some nodes of T as either *left-knee*, *right-knee*, *ordinary-left*, *ordinary-right*, *switch-left* or *switch-right* nodes (for an example, see Figure 4.4.1):

1. Let $P = v_0v_1v_2\dots v_m$ be a spine of T. Define a *non-spine* node of T to be one that is not in P.

2. Designate v_0 as a *left-knee* node.

3. for $i = 0$ to m do (see Figure 4.4.1)

   Depending upon whether v_i is a *left-knee*, *right-knee*, *ordinary-left*, *ordinary-right*, *switch-left*, or *switch-right* node, do the following:

   (a) *v_i is a left-knee node:* If v_{i+1} has a left child, and this child is not v_{i+2}, then designate v_{i+1} as a *switch-right node,* otherwise designate it as an *ordinary-left node*. Recursively construct an upward left-corner drawing of the subtree of T rooted at the non-spine child of v_i.

(b) *v_i is an ordinary-left node:* If v_{i+1} has a left child, and this child is not v_{i+2}, then designate v_{i+1} as a *switch-right node*, otherwise designate it as an *ordinary-left node*. Recursively construct an upward left-corner drawing of the subtree of T rooted at the non-spine child of v_i.

(c) *v_i is a switch-right node:* Designate v_{i+1} as a *right-knee* node. Recursively construct an upward left-corner drawing of the subtree of T rooted at the non-spine child of v_i.

(d) *v_i is a right-knee, ordinary-right, or switch-left node:* Do the same as in the cases, where v_i is a left-knee, ordinary-left, or switch-right node, respectively, with "left" exchanged with "right", and instead of constructing an upward left-corner drawing of the subtree T_i of T rooted at the non-spine child of v_i, we recursively construct an upward left-corner drawing of the *mirror image* of T_i.

4. Let G be the drawing with the maximum width among the drawings constructed in Step 3. Let W be the width of G.

5. Place v_0 at the origin.

6. for $i = 0$ to m do (see Figures 4.4.1 and 4.4.2)

   Let H_i be the horizontal channel corresponding to the node placed lowest in the drawing of T constructed so far.

Depending upon whether v_i is a *left-knee, right-knee, ordinary-left*, *ordinary-right*, *switch-left*, or *switch-right* node, do the following:

(a) *v_i is a left-knee node:* If v_{i+1} is the only child of v_i, then place v_{i+1} on the horizontal channel H_i+1 and one unit to the right of v_i (see Figure 4.4.2(a)). Otherwise, let s be the child of v_i different from v_{i+1}. Let D be the drawing of the subtree rooted at s constructed in Step 3. If s is the right child of v_i, then place D such that its top boundary is at the horizontal channel H_i+1 and its left boundary is one unit to the right of v_i; place v_{i+1} one unit below D and one unit to the right of v_i (see Figure 4.4.2(b)). If s is the left child of v_i, then place v_{i+1} one unit below and one unit to the right of v_i (see Figure 4.4.2(a)) (the placement of D will be handled by the algorithm when it will consider a switch-right node later on).

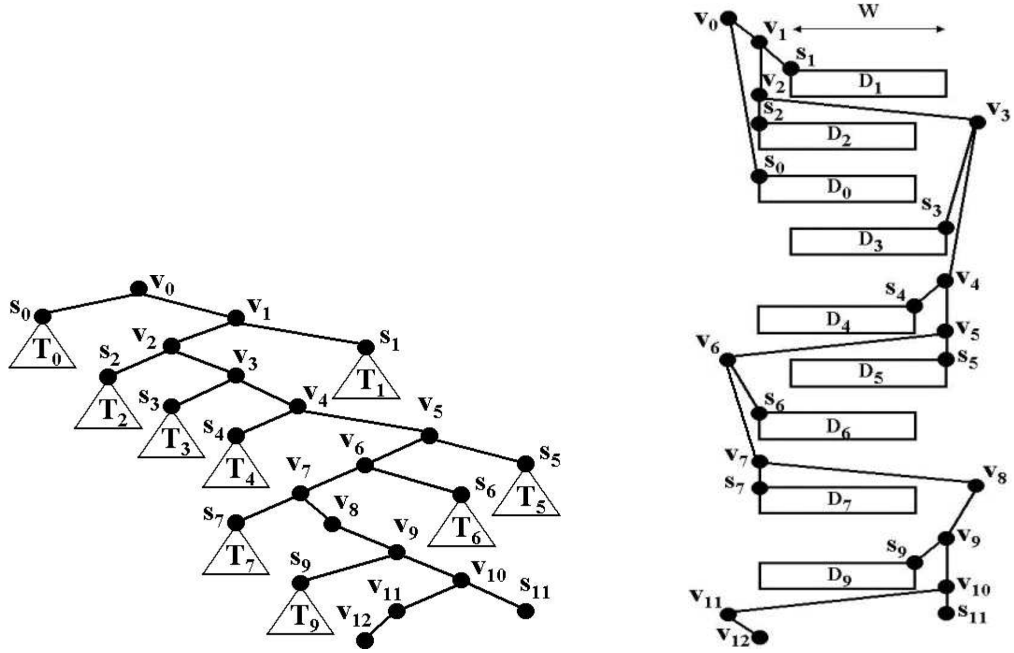
(b) *v_i is an ordinary-left node:* Since v_i is an ordinary-left node, either v_{i+1} will be the only child of v_i, or v_i will have a right child s, where $s \neq v_{i+1}$. If v_{i+1} is the only child of v_i, then place v_{i+1} one unit below v_i in the same vertical channel as it (see Figure 4.4.2(c)). Otherwise, let s be the right child of v_i. Let D be the drawing of the subtree rooted at s constructed in Step 3. Place D one unit below and one unit to the right of v_i; place v_{i+1} on the same horizontal channel as the bottom of D and in the same vertical channel as v_i (see Figure 4.4.2(d)).

(c) *v_i is a switch-right node:* Note that, since v_i is a switch-right node, it will have a left child s, where $s \neq v_{i+1}$. Let v_j be the left-knee node of P closest to v_i in the subpath $v_0v_1 \ldots v_i$ of P. v_j is called the *closest left-knee ancestor* of v_i. Place v_{i+1} one unit below and $W+1$ units to the right of v_i.

Let D be the drawing of the subtree rooted at s constructed in Step 3. Place D one unit below v_i such that s is in the same vertical channel as v_i (see Figure 4.4.2(e)). If v_j has a left child s', which is different from v_{j+1}, then let D' be the drawing of the subtree rooted at s' constructed in Step 3. Place D' one unit below D such that s' is in the same vertical channel as v_i (see Figure 4.4.2(f)).

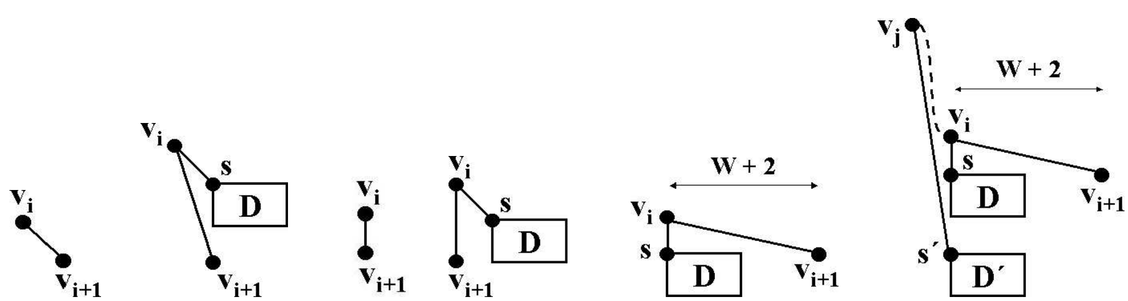
(d) v_i *is a right-knee, ordinary-right, or switch-left node:* These cases are the same as the cases, where v_i is a left-knee, ordinary-left, or switch-right node, respectively, except that "left" is exchanged with "right", and the left-corner drawing of the mirror image of the subtree rooted at the non-spine child of v_i, constructed in Step 3, is first flipped left-to-right and then is placed in $D(T)$.

To determine the area of $D(T)$, notice that the width of $D(T)$ is equal to $W+3$ (see the definition of W given in Step 3). From the definition of a spine, it follows easily that the number of nodes in each subtree rooted at a non-spine node of T is at most $n/2$, where n is the number of nodes in T. Hence, if we denote by $w(n)$, the width of $D(T)$, then, $W \leq w(n/2)$, and so, $w(n) \leq w(n/2)+3$. Hence, $w(n) = O(\log n)$. The height of $D(T)$ is trivially at most n. Hence, the area of $D(T)$ is $O(n \log n)$. It is easy to see that the Algorithm can be implemented such that it runs in $O(n)$ time.



(a) (b)

Figure 4.4.1: (a) A binary tree T with spine $v_0 v_1 \ldots v_{12}$. (b) A schematic diagram of the drawing $D(T)$ of T constructed by *Algorithm BT-Ordered-Draw*. Here, v_0 is a left-knee, v_1 is an ordinary-left, v_2 is a switch-right, v_3 is a right-knee, v_4 is an ordinary-right, v_5 is a switch-left, v_6 is a left-knee, v_7 is a switch-right, v_8 is a right-knee, v_9 is an ordinary-right, v_{10} is a switch-left, v_{11} is a left-knee, and v_{12} is an ordinary-left node. For simplicity, we have shown $D_0, D_1, \ldots, D_9$ with identically sized boxes but in actuality they may have different sizes.



(a) (b) (c) (d) (e) (f)

Figure 4.4.2: (a,b) Placement of v_i, v_{i+1}, and D in the case when v_i is a left-knee node: (a) v_{i+1} is the only child of v_i or s is the left child of v_i, (b) s is the right child of v_i. (c,d) Placement of v_i, v_{i+1}, and D in the case when v_i is an ordinary-left node: (c) v_{i+1} is the only child of v_i, (d) s is the right child of v_i. (e,f) Placement of v_i, v_{i+1}, D, and D in the case when v_i is a switch-right node. (e) v_j does not have a left child, (f) v_j has a left child s'. Here, D' is the drawing of the subtree rooted at s'.

[17] has shown a lower bound of $\Omega(n \log n)$ for order-preserving planar upward straight-line grid drawings of binary trees. Hence, the upper bound of $O(n \log n)$ on the area of $D(T)$ is also optimal. We therefore get the following theorem:

Theorem 4.4.1 *A binary tree with n nodes admits an order-preserving upward planar straight-line grid drawing with height at most n, width $O(\log n)$, and optimal $O(n \log n)$ area, which can be constructed in $O(n)$ time.*

We can also construct a non-upward left-corner drawing $D'(T)$ of T, such that $D'(T)$ has height $O(\log n)$ and width at most n, by first constructing a left-corner drawing of the mirror image of T using Algorithm *BT-Ordered-Draw*, then rotating it clockwise by $90°$, and then flipping it right-to-left. This gives Corollary 4.4.1.

Corollary 4.4.1 *Using Algorithm* BT-Ordered-Draw, *we can construct in $O(n)$ time, a non-upward left-corner order-preserving planar straight-line grid drawing of an n-node binary with area $O(n \log n)$, height $O(\log n)$, and width at most n.*

4.5 Drawing General Trees

In a general tree, a node may have more than two children. This makes it more difficult to draw a general tree.

In this section, we give an algorithm, which we call *Algorithm Ordered-Draw*, for constructing (non-upward) order-preserving planar straight-line grid drawing with $O(n \log n)$ area in $O(n)$ time. Algorithm *Ordered-Draw* is a modification of the algorithm for drawing binary trees presented in Section 4.4.

Let T be a tree with n nodes. In each recursive step, Algorithm *Ordered-Draw* breaks T into several subtrees, draws each subtree recursively, and then combines their drawings to obtain a left-corner drawing $D(T)$ of T.

We now give the details of the actions performed at each recursive step of the algorithm to construct a a left-corner drawing $D(T)$ of T. Note that the counterclockwise ordering of edges around each node, induces a counterclockwise ordering of the children of each node. During its working, the algorithm will designate some nodes of T as either *left-knee*, *right-knee*, *switch-left*, or *switch-right* nodes (for an example, see Figure 4.5.1):

1. Let $P = v_0 v_1 v_2 \dots v_m$ be a spine of T. Define a *non-spine* node of T to be one that is not in P.

2. Designate v_0 as a left-knee node.

3. for $i = 0$ to m do (see Figure 4.5.1)

   Depending upon whether v_i is a *left-knee*, *right-knee*, *ordinary-left*, *ordinary-right*, *switch-left*, or *switch-right* node, do the following:

   (a) v_i *is a left-knee node:* Designate v_{i+1} as a *switch-right node*. Recursively construct left-corner drawings of the subtrees of T rooted at all the non-spine children of v_i.

(b) *v_i is a switch-right node:* Designate v_{i+1} as a *right-knee* node. Recursively construct left-corner drawings of the subtrees of T rooted at all the non-spine children of v_i.

(c) *v_i is a right-knee, or switch-left node:* These cases are the same as the cases, where v_i is a left-knee node, or switch-right node, respectively, with "left" exchanged with "right", and instead of recursively constructing left-corner drawings of the subtrees of T rooted at all the non-spine children of v_i, we recursively construct the left-corner drawings of the *mirror images* of these subtrees.

4. Let G be the drawing with the maximum width among the drawings constructed in Step 3. Let W be the width of G.

5. Place v_0 at the origin. Let Y_0 be the horizontal channel one unit below the origin.

6. for $i = 0$ to m do (see Figures 4.5.1 and 4.5.2)

Depending upon whether v_i is a *left-knee*, *right-knee*, *ordinary-left*, *ordinary-right*, *switch-left*, or *switch-right* node, do the following:

(a) *v_i is a left-knee node:* Let $Q = s_1 s_2 \ldots s_k$ be the (possibly empty) sequence of the children of v_i that come after v_{i+1} in the counterclockwise order of the children of v_i (see Figure 4.5.2(a)). In this sequence, the s_j's, $1 \leq j \leq k$, are placed in the same order as they occur in the counterclockwise order of the children of v_i. Let D_j be the drawing of the subtree rooted at s_j constructed in Step 3. Place $D_1, D_2, \ldots, D_k$ in that order one above the other at unit vertical separation from each other, such that D_1 is at the bottom and D_k is at the top, the top of D_k is at the horizontal channel Y_i, and each D_j is placed one unit to the right of v_i (see Figure 4.5.2(b)).

Let Y_{i+1} be the horizontal channel one unit below D_1 if Q is not empty, and is the same as Y_i if Q is empty.

(b) *v_i is a switch-right node:* Note that, since v_i is a switch-right node, v_{i-1} must be a left-knee node.

Let $Q = s_1 s_2 \ldots s_k$ be the (possibly empty) sequence of the children of v_i that come after v_{i+1} in the counterclockwise order of the children of v_i (see Figure 4.5.2(c)). In this sequence, the s_j's, $1 \le j \le k$, are placed in the same order as they occur in the counterclockwise order of the children of v_i. Let D_j be the drawing of the subtree rooted at s_j constructed in Step 3. Place $D_1, D_2, \ldots, D_k$ in that order one above the other at unit vertical separation from each other, such that D_1 is at the bottom and D_k is at the top, the top of D_k is at horizontal channel Y_i, and each D_j is placed two units to the right of v_{i-1}.

Place v_i such that it is one unit to the right of v_{i-1}, and is one unit below D_1, if Q is not empty, and is at the horizontal channel Y_i if Q is empty.

Place v_{i+1} one unit below and $W+1$ units to the right of v_i (see Figure 4.5.2(d)).

Let $Q' = s'_1 s'_2 \ldots s'_r$ be the (possibly empty) sequence of the children of v_i that come before v_{i+1} in the counterclockwise order of the children of v_i (see Figure 4.5.2(c)). In this sequence, the s'_j's, $1 \leq j \leq r$, are placed in the same order as they occur in the counterclockwise order of the children of v_i. Let D'_j be the drawing of the subtree rooted at s'_j constructed in Step 3. Place $D'_1, D'_2, \ldots, D'_r$ in that order one above the other at unit vertical separation from each other, such that D'_1 is at the bottom and D'_r is at the top, s'_r is placed on the same vertical channel as v_{i+1}, and each D'_j is placed two units to the right of v_{i-1} (see Figure 4.5.2(d)).

Let H be the horizontal channel which is one unit below the bottom of D'_1 if Q' is not empty, and contains v_{i+1} if Q' is empty.

Let $Q'' = s''_1 s''_2 \ldots s''_t$ be the (possibly empty) sequence of the children of v_{i-1} that come before v_i in the counterclockwise order of the children of v_{i-1} (see Figure 4.5.2(c)). In this sequence, the s''_j's, $1 \leq j \leq t$, are placed in the same order as they occur in the counterclockwise order of the children of v_i. Let D''_j be the drawing of the subtree rooted at s''_j constructed in Step 3. Place $D''_1, D''_2, \ldots, D''_r$ in that order one above the other at unit vertical separation from each other, such that D''_1 is at the bottom and D''_t is at the top, the top of D''_t is at the horizontal channel H, and each D''_j is placed one unit to the right of v_{i-1} (see Figure 4.5.2(d)).

Let Y_{i+1} be the horizontal channel which is one unit below the bottom of D''_1 if Q'' is not empty, and is the same as H if Q'' is empty.

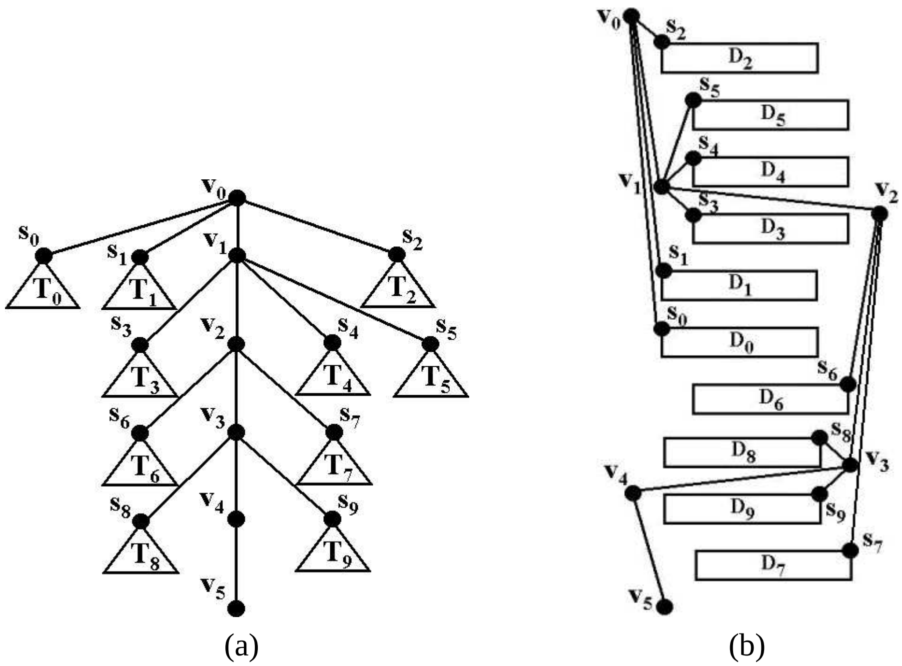


Figure 4.5.1: (a) A tree T with spine $v_0v_1 \ldots v_5$. (b) An $O(n \log n)$ area planar straight-line grid drawing of T. In this drawing, v_0 is left-knee node, v_1 is switch-right node, v_2 is right-knee node, v_3 is switch-left node, v_4 is left-knee node, v_5 is switch-right node.

(c) *v_i is a right-knee, or switch-left node:* These cases are the same as the cases, where v_i is a left-knee node, or switch-right node, respectively, except that "left" is exchanged with "right", "counterclockwise" is replaced by "clockwise", and the left-corner drawings of the mirror images of the subtrees rooted at the non-spine children of v_i, constructed in Step 3, are first flipped left-to-right and then are placed in $D(T)$.

Just as for *Algorithm BT-Ordered-Draw*, we can show that the width $w(n)$ of $D(T)$ satisfies the recurrence: $w(n) \leq w(n/2) + 3$. Hence, $w(n) = O(\log n)$. The height of $D(T)$ is trivially at most n. Hence, the area of $D(T)$ is $O(n \log n)$.

Theorem 4.5.1 *A tree with n nodes admits an order-preserving planar straight-line grid*

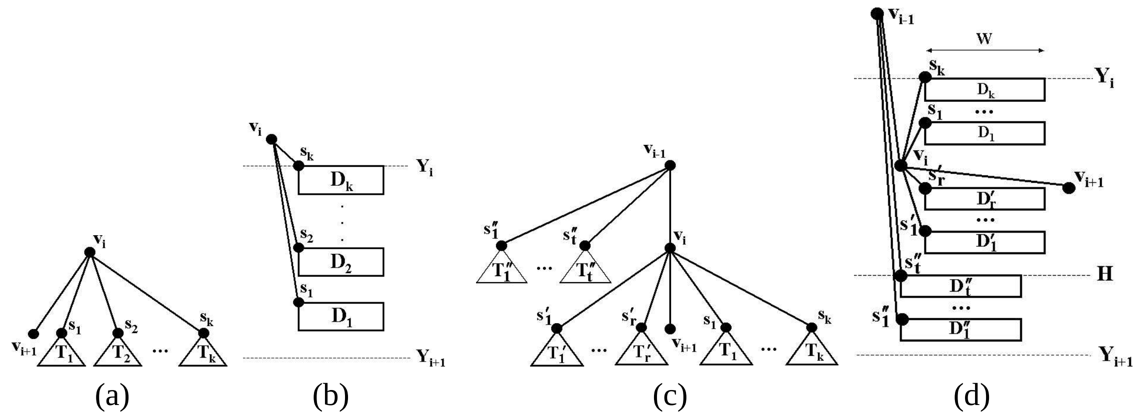


Figure 4.5.2: (a) $s_1, s_2, \ldots, s_k$ is the sequence of the children of v_i that come after v_{i+1} in the counterclockwise order of the children of v_i. (b) Placement of v_i, $s_1, s_2, \ldots, s_k$, and $D_1, D_2, \ldots, D_k$ in the case when v_i is a left-knee node. (c) $s_1, \ldots, s_k$ is the sequence of the children of v_i that come after v_{i+1} in the counterclockwise order of the children of v_i. $s'_1, \ldots, s'_k$ is the sequence of the children of v_i that come before v_{i+1} in the counterclockwise order of the children of v_i. $s''_1, \ldots, s''_k$ is the sequence of the children of v_{i-1} that come before v_i in the counterclockwise order of the children of v_{i-1}. (d) Placement of v_i, v_{i+1}, $s_1, \ldots, s_k$, $D_1, \ldots, D_k$, $s'_1, \ldots, s'_k$, $D'_1, \ldots, D'_k$, $s''_1, \ldots, s''_k$, $D''_1, \ldots, D''_k$ in the case when v_i is a switch-right node.

drawing with $O(n \log n)$ area, $O(\log n)$ width, and height at most n, which can be constructed in $O(n)$ time.

We can also construct a left-corner drawing $D'(T)$ of T, such that $D'(T)$ has height $O(\log n)$ and width at most n, by first constructing a left-corner drawing of the mirror image of T using Algorithm *Ordered-Draw*, then rotating it clockwise by 90°, and then flipping it right-to-left. This gives Corollary 4.5.1.

Corollary 4.5.1 *Let T be a tree with n nodes. Using Algorithm* Ordered-Draw*, we can construct in $O(n)$ time, a left-corner order-preserving planar straight-line grid drawing D of T with $O(n \log n)$ area, such that D has height $O(\log n)$, and width at most n.*

4.6 Drawing Binary Trees with Arbitrary Aspect Ratio

Let T be a binary tree. We show that, for any user-defined number A, where $2 \le A \le n$, we can construct an order-preserving planar straight-line grid drawing of T with $O((n/A)\log A)$ height and $O(A + \log n)$ width. Thus, by setting the value of A, users can control the aspect ratio of the drawing. This result also implies that we can construct such a drawing with area $O(n \log\log n)$ by setting $A = \log n$.

Our algorithm combines the approach of [3] for constructing non-upward non-order-preserving drawings of binary trees with arbitrary aspect ratio with our approach for constructing order-preserving drawings given in Sections 4.4 and 4.5. We will also use the following generalization of Lemma 3 of [3]:

Lemma 4.6.1 *Suppose $A > 1$, and f is a function such that:*

- *if $n \le A$, then $f(n) \le 1$; and*
- *if $n > A$, then $f(n) \le f(n^*) + f(n^+) + f(n'') + 1$ for some $n^*, n^+, n'' \le n - A$ with $n^* + n^+ + n'' \le n$.*

Then, $f(n) < 6n/A - 2$ for all $n > A$.

Proof: The proof is by induction over n, with the base case being $n = A + 1$.

If $n = A + 1$, then $n^*, n^+, n'' \le A$. Hence, $f(n^*), f(n^+), f(n'') \le 1$. Hence, $f(n) \le 1 + 1 + 1 + 1 = 4 < 6n/A - 2$.

Now we prove the induction. Suppose $f(m) < 6m/A - 2$ for all $m \leq n-1$. Consider $f(n)$. We have four cases:

- $n^*, n^+, n'' \leq A$: Then, $f(n^*), f(n^+), f(n'') \leq 1$. Hence, $f(n) \leq 1+1+1+1 = 4 < 6n/A - 2$.

- Exactly two of n^*, n^+, and n'' have values less than or equal to A: Assume without loss of generality that $n^*, n^+ \leq A$ and $n'' > A$. Then, $f(n^*), f(n^+) \leq 1$, and $f(n'') < 6n''/A - 2 \leq 6(n-A)/A - 2 = 6n/A - 6 - 2 = 6n/A - 8$. Hence, $f(n) \leq 1+1+6n/A - 8 + 1 = 6n/A - 5 < 6n/A - 2$.

- Exactly one of n^*, n^+, and n'' has value less than or equal to A: Assume without loss of generality that $n^* \leq A$, and $n^+, n'' > A$. Then, $f(n^*) \leq 1$, $f(n^+) + f(n'') < 6n^+/A - 2 + 6n''/A - 2 = 6(n^+ + n'')/A - 4 < 6n/A - 4$. Hence, $f(n) < 1 + 6n/A - 4 + 1 = 6n/A - 2$.

- $n^*, n^+, n'' > A$: $f(n) = f(n^*) + f(n^+) + f(n'') + 1 < 6n^*/A - 2 + 6n^+/A - 2 + 6n''/A - 2 + 1 = 6(n^* + n^+ + n'')/A - 5 \leq 6n/A - 5 < 6n/A - 2$.

□

An order-preserving planar straight-line grid drawing of a binary tree T is called a *feasible drawing*, if the root of T is placed on the left boundary and no node of T is placed between the root and the upper-left corner of the enclosing rectangle of the drawing. Note that a left-corner drawing is also a feasible drawing.

We now describe our algorithm, which we call *Algorithm BDAAR*, for drawing a binary tree T with arbitrary aspect ratio. Let m be the number of nodes in T. Let $2 \leq A \leq m$ be any number given as a parameter to *Algorithm BDAAR*.

Figure 4.6.1(a) and Figure 4.6.1(b) show the drawings of the tree of Figure 4.3.1(a) constructed by Algorithm *BDAAR* with $A = \sqrt{m}$ and using Corollary 4.4.1, and Corollary 4.5.1, respectively.

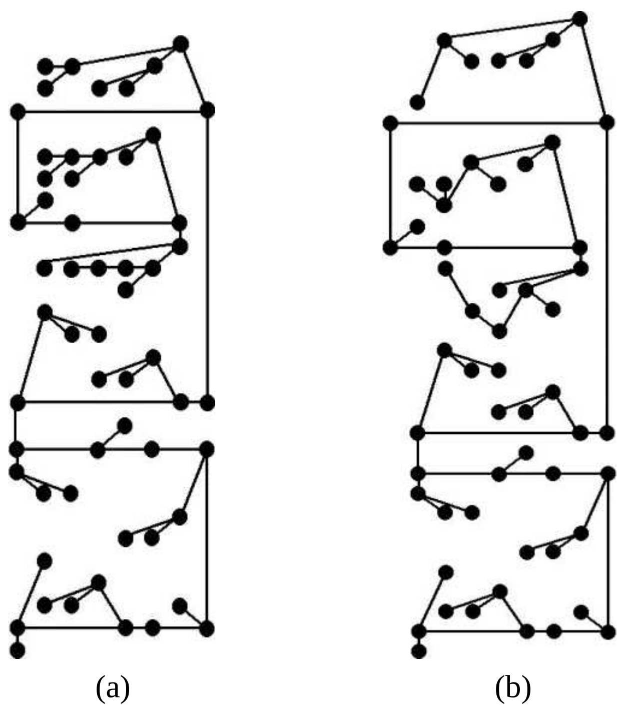


Figure 4.6.1: Drawings of the tree with $n = 57$ nodes of Figure 4.3.1(a) constructed by Algorithm *BDAAR* with $A = \sqrt{m} = \sqrt{57} = 7.55$ and using: (a) Corollary 4.4.1, and (b) Corollary 4.5.1, respectively.

Like Algorithm *BT-Ordered-Draw* of Section 4.4, *Algorithm BDAAR* is also a recursive algorithm. In each recursive step, it also constructs a feasible drawing of a subtree T' of T. If T' has at most A nodes in it, then it constructs a left-corner drawing of T' using Corollary 4.4.1 or Corollary 4.5.1, such that the drawing has width at most n and height

$O(\log n)$, where n is the number of nodes in T'. Otherwise, i.e., if T' has more than A nodes in it, then it constructs a feasible drawing of T' as follows:

1. Let $P = v_0 v_1 v_2 \ldots v_q$ be a spine of T'.

2. Let n_i be the number of nodes in the subtree of T' rooted at v_i. Let v_k be the vertex of P with the smallest value for k such that $n_k > n - A$ and $n_{k+1} \leq n - A$ (since T' has more than A nodes in it and $n_0, n_1, \ldots, n_q$ is a strictly decreasing sequence of numbers, such a k exists).

3. for each i, where $0 \leq i \leq k-1$, denote by T_i, the subtree rooted at the non-spine child of v_i (if v_i does not have any non-spine child, then T_i is the empty tree, i.e., the tree with no nodes in it). Denote by T^* and T^+, the subtrees rooted at the non-spine children of v_k and v_{k+1}, respectively, denote by T'', the subtree rooted at v_{k+1}, and denote by T''', the subtree rooted at v_{k+2} (if v_k and v_{k+1} do not have non-spine children, and $k+1 = q$, then T^*, T^+, and T''' are empty trees). For simplicity, in the rest of the algorithm, we assume that T^*, T^+, T''', and each T_i are non-empty. (The algorithm can be easily modified to handle the cases, when T^*, T^+, T''', or some T_i's are empty).

4. Place v_0 at origin.

5. We have two cases:

   - $k = 0$: Recursively construct a feasible drawing D^* of T^*. Recursively construct a feasible drawing D^+ of the mirror image of T^+. Recursively construct a feasible drawing D''' of the mirror image of T'''. Let s_0 be the root of T^* and s_1 be the root of T^+.

T' is drawn as shown in Figure 4.6.2(a,b,c,d). If s_0 is the left child of v_0, then place D^* one unit below v_0 with its left boundary aligned with v_0 (see Figure 4.6.2(a,c)). If s_0 is the right child of v_0, then place D^* one unit above and one unit to the right of v_0 (see Figure 4.6.2(b,d)). Let W^*, W^+, and W''' be the widths of D^*, D^+, and D''', respectively. v_1 is placed in the same horizontal channel as v_0 to its right at distance $\max\{W^*+1, W^+ +1, W'''-1\}$ from it. Let B_0 and C_0 be the lowest and highest horizontal channels, respectively, occupied by the subdrawing consisting of v_0 and D^*. If s_1 is the left child of v_1, then flip D^+ left-to-right and place it one unit below B_0 and one unit to the left of v_1 (see Figure 4.6.2(a,b)). If s_1 is the right child of v_1, then flip D^+ left-to-right, and place it one unit above C_0 and one unit to the left of v_1 (see Figure 4.6.2(c,d)). Let B_1 be the lowest horizontal channel occupied by the subdrawing consisting of v_0, D^*, v_1 and D^+. Flip D''' left-to-right and place it one unit below B_1 such that its right boundary is aligned with v_1 (see Figure 4.6.2(a,b,c,d)).

- $k > 0$: For each T_i, where $0 \le i \le k-1$, construct a left-corner drawing D_i of T_i using Corollary 4.4.1 or Corollary 4.5.1.

  Recursively construct feasible drawings D^* and D'' of the mirror images of T^* and T'', respectively.

  T' is drawn as shown in Figure 4.6.3(a,b,c,d). If T_0 is rooted at the left child of v_0, then D_0 is placed one unit below and with the left boundary aligned with v_0.

If T_0 is rooted at the right child of v_0, then D_0 is placed one unit above and one unit to the right of v_0. Each D_i and v_i, where $1 \leq i \leq k-1$, are placed such that:

- v_i is in the same horizontal channel as v_{i-1}, and is one unit to the right of D_{i-1}, and
- if T_i is rooted at the left child of v_i, then D_i is placed one unit below v_i with its left boundary aligned with v_i, otherwise (i.e., if T_i is rooted at the right child of v_i) D_i is placed one unit above and one unit to the right of v_i.

Let B_{k-1} and C_{k-1} be the lowest and highest horizontal channels, respectively, occupied by the subdrawing consisting of $v_0, v_1, v_2, \ldots, v_{k-1}$ and $D_0, D_1, D_2, \ldots, D_{k-1}$. Let d be the horizontal distance between v_0 and the right boundary of the subdrawing consisting of $v_0, v_1, v_2, \ldots, v_{k-1}$ and $D_0, D_1, D_2, \ldots, D_{k-1}$. Let W^* and W'' be the widths of D^* and D'', respectively. v_k is placed to the right of v_{k-1} in the same horizontal channel as it, such that the horizontal distance between v_k and v_0 is equal to $\max\{W''-1, W^*+1, d+1\}$. If T^* is rooted at the left-child of v_k, then D^* is flipped left-to-right and placed one unit below B_{k-1} and one unit left of v_k (see Figure 4.6.3(a,b)). If T^* is rooted at the right-child of v_k, then D^* is flipped left-to-right and placed one unit above C_{k-1} and one unit to the left of v_k (see Figure 4.6.3(c,d)) . Let B_k be the lowest horizontal channel occupied by the subdrawing consisting of $v_1, v_2, \ldots, v_k$, and $D_1, D_2, \ldots, D_{k-1}, D^*$. D'' is flipped left-to-right and placed one unit below B_k, such that its right boundary is aligned with v_k (see Figure 4.6.3(b,d)).

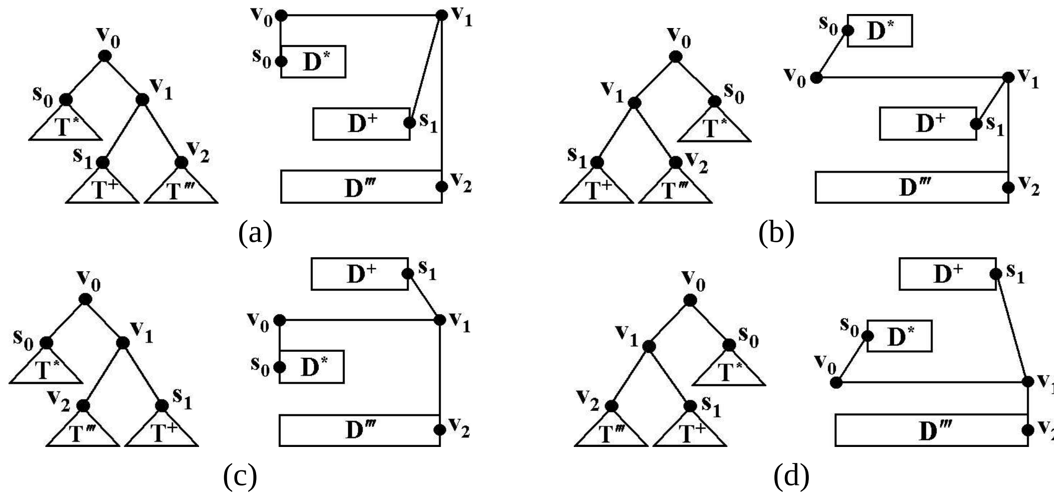


Figure 4.6.2: Case $k = 0$: (a) s_0 is the left child of v_0 and s_1 is the left child of v_1. (b) s_0 is the right child of v_0 and s_1 is the left child of v_1. (c) s_0 is the left child of v_0 and s_1 is the right child of v_1. (d) s_0 is the right child of v_0 and s_1 is the right child of v_1.

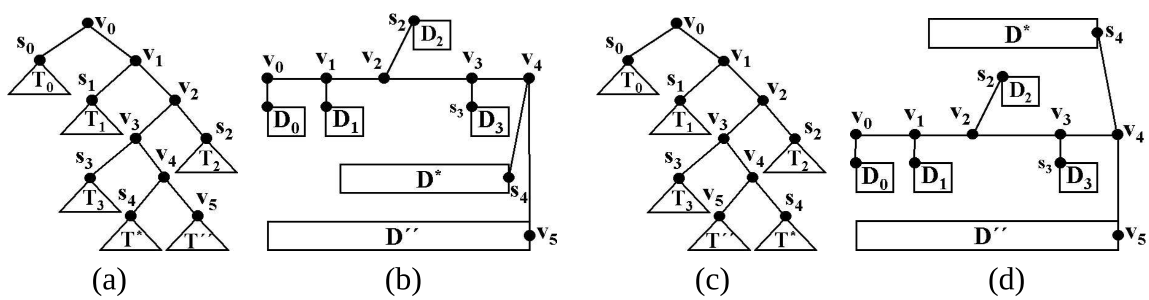


Figure 4.6.3: Case $k > 0$: Here $k = 4$, s_0, s_1, and s_3 are the left children of v_0, v_1, and v_3 respectively, s_2 is the right child of v_2, T_0, T_1, T_2, and T_3 are the subtrees rooted at v_0, v_1, v_2, and v_3 respectively. Let s_4 be the root of T^*. (a) s_4 is left child of v_4. (b) s_4 is the right child of v_4.

Let m_i be the number of nodes in T_i, where $0 \leq i \leq k-1$. From Corollaries 4.4.1 and 4.5.1, the height of each D_i is $O(\log m_i)$ and width at most m_i. Total number of nodes in the partial tree consisting of $T_0, T_1, \ldots, T_{k-1}$ and $v_0, v_1, \ldots, v_{k-1}$ is at most $A-1$. Hence, the height of the subdrawing consisting of $D_0, D_1, \ldots, D_{k-1}$ and $v_0, v_1, \ldots, v_{k-1}$ is $O(\log A)$ and width is at most $A-1$ (see Figure 4.6.3).

Suppose T', T^*, T^+, T'', and T''' have n, n^*, n^+, n'', and n''' nodes, respectively. If we denote by $H(n)$ and $W(n)$, the height and width of the drawing of T' constructed by Algorithm *BDAAR*, then:

$$\begin{aligned} H(n) &= H(n^*) + H(n^+) + H(n''') + 1 \text{ if n > A and k = 0} \\ &= H(n^*) + H(n^+) + H(n''') + O(\log A) \\ H(n) &= H(n^*) + H(n'') + O(\log A) \text{ if n > A and k > 0} \\ H(n) &= O(\log A) \text{ if n} \leq \text{A} \end{aligned}$$

Since $n^*, n^+, n'', n''' \leq n - A$, from Lemma 4.6.1, it follows that $H(n) = O(\log A)(6n/A - 2) = O((n/A)\log A)$. Also we have that:

$$\begin{aligned} W(n) &= \max\{W(n^*) + 2, W(n^+) + 2, W(n''')\} \text{ if n > A and k = 0} \\ W(n) &= \max\{A, W(n^*) + 2, W(n'')\} \text{ if n > A and k > 0} \\ W(n) &\leq A \text{ if n} \leq \text{A} \end{aligned}$$

Since, $n^*, n^+, n^* \leq n/2$, and $n'', n''' \leq n - A < n - 1$, we get that $W(n) \leq \max\{A, W(n/2) + 2, W(n-1)\}$. Therefore, $W(n) = O(A + \log n)$. We therefore get the following theorem:

Theorem 4.6.1 *Let T be a binary tree with n nodes. Let $2 \leq A \leq n$ be any number. T admits an order-preserving planar straight-line grid drawing with width $O(A + \log n)$, height $O((n/A)\log A)$, and area $O((A + \log n)(n/A)\log A) = O(n \log n)$, which can be constructed in $O(n)$ time.*

Setting $A = \log n$, we get that:

Corollary 4.6.1 *An n-node binary tree admits an order-preserving planar straight-line grid drawing with area $O(n \log\log n)$, which can be constructed in $O(n)$ time.*

Chapter 5

Area-Efficient Planar Straight-line Grid Drawings of Outerplanar Graphs

5.1 Introduction

A *drawing* Γ of a graph G maps each vertex of G to a distinct point in the plane, and each edge (u,v) of G to a simple Jordan curve with endpoints u and v. Γ is a *straight-line* drawing, if each edge is drawn as a single line-segment. Γ is a *polyline* drawing, if each edge is drawn as a connected sequence of one or more line-segments, where the meeting point of consecutive line-segments is called a *bend*. Γ is a *grid* drawing if all the nodes have integer coordinates. Γ is a *planar* drawing, if edges do not intersect each other in the drawing. In this chapter, we concentrate on grid drawings. So, we will assume that the plane is covered by a rectangular grid. Let R be a rectangle with sides parallel to the X- and Y-axes. The *width* (*height*) of R is equal to the number of grid points with the same y (x) coordinate contained within R. The *area* of R is equal to the number of grid points contained within R. R is the *enclosing rectangle* of Γ, if it is the smallest rectangle that covers the entire drawing. The *width*, *height*, and *area* of Γ is equal to the width, height, and area respectively, of its enclosing rectangle. The *degree* of a graph is equal to the maximum number of edges incident on a vertex.

It is well-known that a planar graph with n vertices admits a planar straight-line grid drawing with $O(n^2)$ area [9,33], and in the worst case it requires $\Omega(n^2)$ area. It is also known that a binary tree with n nodes admits a planar straight-line grid drawing with $O(n)$ area [18]. Thus, there is wide gap between the $\Theta(n^2)$ area-requirement of general planar graphs and the $\Theta(n)$ area-requirement of binary trees. It is therefore important to investigate special categories of planar graphs to determine if they can be drawn in $o(n^2)$ area.

Outerplanar graphs form an important category of planar graphs. We investigate the area-requirement of planar straight-line grid drawings of outerplanar graphs. Currently the best known bound on the area-requirement of such a drawing of an outerplanar graph with n vertices is $O(n^2)$, which is that same as for general planar graphs. Hence, a fundamental question arises: can we draw an outerplanar graph in this fashion in $o(n^2)$ area?

In this chapter, we provide a partial answer to this question by proving that an outerplanar graph with n vertices and degree d can be drawn in this fashion in area $O(dn^{1+0.48}) = O(dn^{1.48})$ in $O(n)$ time. This implies that an outerplanar graph with n vertices and degree

$O(n^{\delta})$, where $0 \leq \delta < 0.52$ is a constant, can be drawn in this fashion in $o(n^2)$ area.

From a broader perspective, our contribution is in showing a sufficiently large natural category of planar graphs that can be drawn in $o(n^2)$ area.

In Section 5.4, we present our drawing algorithm. This algorithm is based on a tree-drawing algorithm of [4]. The connection between the two algorithms comes from the fact that the dual of an outerplanar graph is a tree.

5.2 Previous Results

There has been little work done on planar straight-line grid drawings of outerplanar graphs. Let G be an outerplanar graph with n vertices. Currently the best known bound on the area-requirement of such a drawing of an outerplanar graph with n vertices is $O(n^2)$, which is that same as for general planar graphs. However, in 3D, we can construct a crossings-free straight-line graph drawing of G with $O(n)$ volume [14, 16].

[1] shows that G admits a planar polyline drawing as well as a visibility representation with $O(n \log n)$ area. [26] shows that G admits a planar polyline drawing with $O(n)$ area, if G has degree 4. The technique of [26] can be easily extended to construct a planar polyline drawing of G with $O(d^2 n)$ area, if G has degree d [1].

The paper based on this Chapter will appear in [20].

5.3 Preliminaries

We assume a 2-dimensional Cartesian space. We assume that this space is covered by an infinite rectangular grid, consisting of horizontal and vertical channels.

We denote by $|G|$ the number of nodes (vertices) in a graph (tree) G.

A *rooted* tree is one with a pre-specified root. An *ordered* tree is a rooted tree with a pre-specified left-to-right order of the children for each node. Let T be an ordered binary tree with n nodes. Let p and δ be two constants such that $p = 0.48$ and $0 < \delta \leq 0.0004$. A *spine* S of T is a path $v_0 v_1 v_2 \ldots v_m$, where $v_0, v_1, v_2, \ldots, v_m$ are nodes of T, that is defined recursively as follows (as defined in the proof of Lemma A.1 in [4]):

- v_0 is the same as the root of T, and v_m is a leaf of T;
- let α_i and β_i be the the left and right subtrees with the maximum number of nodes among the subtrees that are rooted at any of the nodes in the path $v_0 v_1 \ldots v_i$; let L_i and R_i be the subtrees rooted at the left and right children of v_i respectively. Then,
  - if $|\alpha_i|^p + |R_i|^p \leq (1-\delta)n^p$ and $|L_i|^p + |\beta_i|^p > (1-\delta)n^p$, set v_{i+1} to be the left child of v_i,
  - if $|\alpha_i|^p + |R_i|^p > (1-\delta)n^p$ and $|L_i|^p + |\beta_i|^p \leq (1-\delta)n^p$, set v_{i+1} to be the right child of v_i,
  - if $|\alpha_i|^p + |R_i|^p \leq (1-\delta)n^p$ and $|L_i|^p + |\beta_i|^p \leq (1-\delta)n^p$, we terminate the construction as follows:

    - if $|L_i| \leq |R_i|$, set the spine to be the concatenation of path $v_0v_1 \ldots v_i$ and the leftmost path from v_i to a leaf v_m,
    - otherwise (i.e. $|L_i| > |R_i|$), set the spine to be the concatenation of the path $v_0v_1 \ldots v_i$ and the rightmost path from v_i to a leaf v_m.
  - in [4] it is shown that the case $|\alpha_i|^p + |R_i|^p > (1-\delta)n^p$ and $|L_i|^p + |\beta_i|^p > (1-\delta)n^p$ is not possible.

$v_0, v_1, \ldots, v_m$ are called *spine nodes*. A *subtree of S* is a subtree of T rooted at the non-spine child of a spine node. A *left (right) subtree of S* is a subtree of T rooted at a left (right) non-spine child of a spine node.

We will use Lemma A.1 of [4], which is given below:

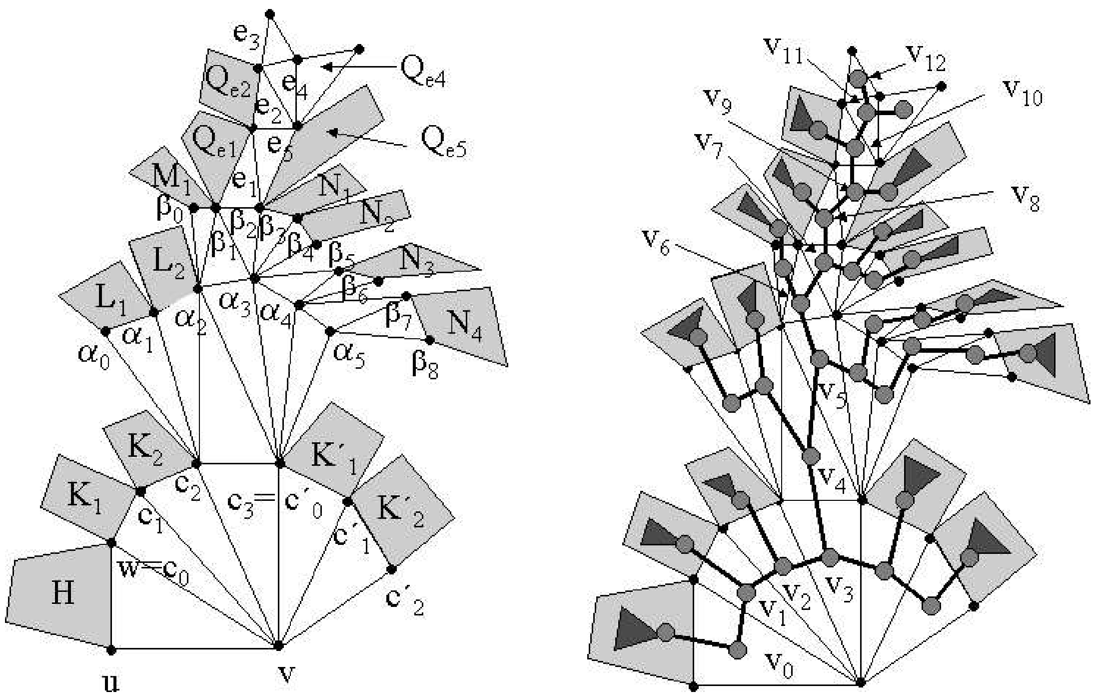
Lemma 5.3.1 (Lemma A.1 of [4]) *Let $p = 0.48$. For any left subtree α and right subtree β of a spine, $|\alpha|^p + |\beta|^p \leq (1-\delta)n^p$, for any constant δ, $0 < \delta \leq 0.0004$.*

An *outerplanar* graph is a planar graph for which there exists an embedding with all vertices on the exterior face. Throughout this chapter, by the term *outerplanar* graph we will mean a *maximal* outerplanar graph, i.e., an outerplanar graph to which no edge can be added without destroying its outerplanarity. It is easy to see that each face of a maximal outerplanar graph is a triangle. Two vertices of a graph are *neighbors*, if they are connected by an edge. The *dual tree* T_G of an outerplanar graph G is defined as follows:

- there is a one-to-one correspondence between the nodes of T_G and the internal faces of G, and

- there is an edge $e = (u, v)$ in T_G if and only if the faces of G corresponding to u and v share an edge e' on their boundaries. e and e' are *duals* of each other.

For example, Figure 5.3.1(b), shows the dual tree of the outerplanar graph of Figure 5.3.1(a).



(a) (b)

Figure 5.3.1: (a) An outerplanar graph G. Here, H, K_1, K_2, K'_1, K'_2, L_1, L_2, M_1, N_1, N_2, N_3, N_4, Q_{e1}, Q_{e2}, Q_{e4}, and Q_{e5} are subgraphs of G, and are themselves outerplanar graphs. (b) The dual tree T_G of G. The edges of T_G are shown with dark lines. Note that $v_0 v_1 \dots v_{12}$ is a spine of T_G.

Let $P = v_0 v_1 \dots v_q$ be a connected path of T_G. Let H be the subgraph of G corresponding to P. A *beam* drawing of H is shown in Figure 5.3.2, where the vertices of H are placed on two horizontal channels, and the faces of H are drawn as triangles.

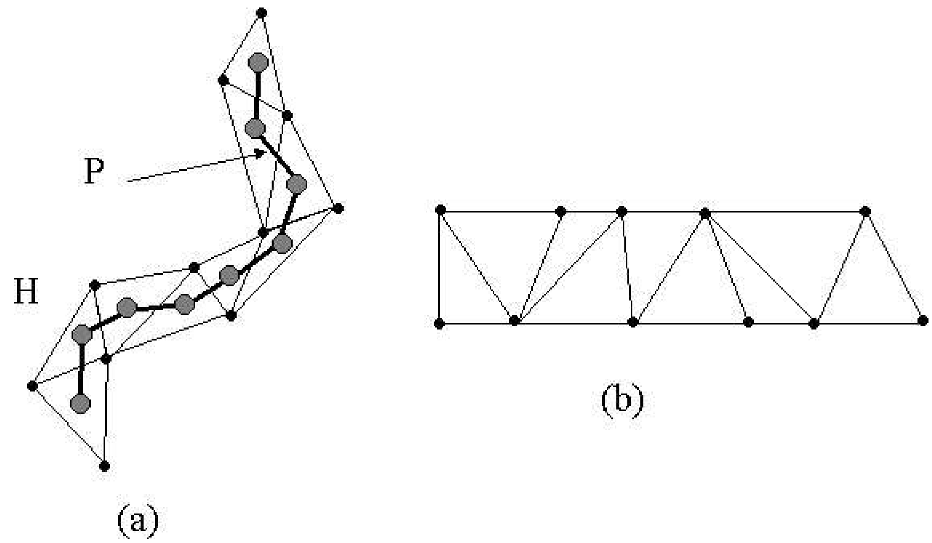


Figure 5.3.2: (a) A path P and its corresponding graph H. (b) A beam drawing of H.

A line-segment with end-points a and b is a *flat* line-segment if a and b either belong to the same horizontal channel, or belong to adjacent horizontal channels.

Let B be a flat line-segment with end-points a and b, such that b is at least two units to the right of a. Let G be an outerplanar graph with two distinguished adjacent vertices u and v, such that the edge (u,v) is on the external face of G; u and v are called the *poles* of G. Let D be a planar straight-line drawing of G. D is a *feasible* drawing of G with base B if:

- the two poles of G are mapped to a and b each,
- each non-pole vertex of G is placed at least one unit above both a and b, and is placed at least one unit to the right of a and at least one unit to the left of b.

5.4 Outerplanar Graph Drawing Algorithm

The drawing algorithm, which we call *Algorithm OpDraw*, is recursive in nature. In each recursive step, it takes as input an outerplanar graph G with pre-specified poles, and a long-enough flat line-segment B, and constructs a feasible drawing D of G with base B by constructing a drawing M of the subgraph H corresponding to the spine of G, splitting G into several smaller outerplanar graphs after removing H and some other vertices from it, constructing feasible drawings of each smaller outerplanar graph, and then combining their drawings with M to obtain D.

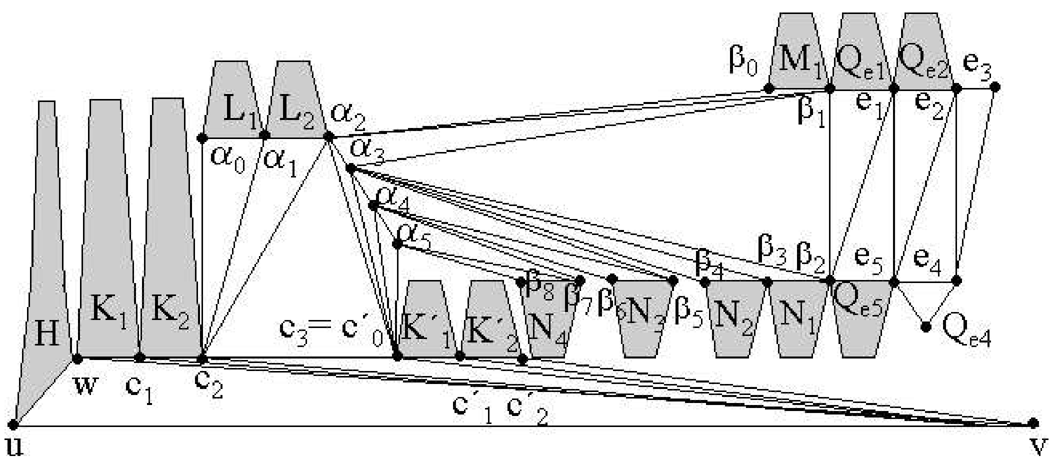


Figure 5.4.1: The drawing of the outerplanar graph of Figure 5.3.1(a) constructed by *Algorithm OpDraw*.

We now give the details of the actions performed by *Algorithm OpDraw* in each recursive step (see Figure 5.4.1):

- Let u and v be the poles of G. Let T_G be the dual tree of G. Let r be the node of T_G that corresponds to the internal face F of G that contains both u and v. Convert T_G into an ordered tree as follows:

– make T_G a rooted tree by making r its root,

– and for each node w, let w' be the parent of w in T_G (which now is a rooted tree). Let c (d) be the children of w such that the face corresponding to c immediately follows (precedes) the face corresponding to w' in the counter-clockwise order of internal faces incident on the face corresponding to w. Make c the leftmost child of w, and d the rightmost child of w. Assign the children of w the same left-to-right order as the counter-clockwise order in which the faces that correspond to them are incident on the face corresponding to w.

Note that T_G is a binary tree because each internal face of G is a triangle.

- Draw F as a triangle such that u and v coincide with the end-points of B, and the third vertex w of F is placed one unit above the higher of u and v. (We will determine later on the horizontal distances of w from u and v, when we analyze the area-requirement of the drawing.) In the rest of this section, we will assume that u is placed either at the same horizontal channel as, or at a higher horizontal channel than v (the case where v is placed higher than u is similar). (In Figure 5.4.1, u and v are shown to be on the same horizontal channel, but the construction given below will also apply if u were placed at a higher horizontal channel than v.)

- Let $P = v_0 v_1 \dots v_q$ be the spine of G, where $v_0 = r$. Assume that the edge (v_0, v_1) is the dual of edge (v, w) (the case where (v_0, v_1) is the dual of edge (u, w) is symmetrical). Let (v_0, v') be the dual of edge (u, w). Let H be the subgraph of G corresponding to the subtree of T_G rooted at v'. Recursively construct a feasible drawing D_H of H with

$\overline{uw}$ as the base.

- Let $c_0 = w, c_1, \ldots, c_m (= c'_0), c'_1, c'_2, \ldots, c'_s$ be the counter-clockwise order of the neighbors of v different from u, where, for each i ($1 \leq i \leq m$), the face $c_{i-1}c_iv$ corresponds to the spine node v_i, and for each i ($1 \leq i \leq s$), the face $c'_{i-1}c'_iv$ corresponds to a non-spine node v'_i of T_G. (In Figure 5.4.1, $m = 3$.) Place the vertices $c_1, \ldots, c_m (= c'_0), c'_1, c'_2, \ldots, c'_s$ at the same horizontal channel as w. (We will determine later on the horizontal distances between these vertices.)

- Let (v_i, x_i) be the dual of edge (c_{i-1}, c_i). Let K_i be the subgraph of G corresponding to the subtree of T_G rooted at x_i. For each i, where $1 \leq i \leq m-1$, recursively construct a feasible drawing of K_i with $\overline{c_{i-1}c_i}$ as the base.

- Let (v'_i, x'_i) be the dual of edge (c'_{i-1}, c'_i). Let K'_i be the subgraph of G corresponding to the subtree of T_G rooted at x_i. For each i, where $1 \leq i \leq s$, recursively construct a feasible drawing D'_i of K'_i with $\overline{c'_{i-1}c'_i}$ as the base.

- Let $\alpha_0, \alpha_1, \ldots, \alpha_t$ be the vertices of K_m, such that $\alpha_0, \alpha_1, \ldots, \alpha_h$ ($0 \leq h \leq t$) is the clockwise order of the neighbors of c_{m-1} in K_m, and $\alpha_h, \alpha_{h+1}, \ldots, \alpha_t$ is the clockwise order of the neighbors of c_m in K_m. Let j be the index such that the dual of edge (c_{j-1}, c_j) belongs to P (if no such j exists, then set $j = t$). (In Figure 5.4.1, $j = 3$.) Place $\alpha_0, \alpha_1, \ldots, \alpha_{j-1}$ in the same horizontal channel, and $\alpha_{j-1}, \alpha_j, \ldots, \alpha_t$ along a line making 45° angle with the horizontal channels, such that

  - α_t is in the same vertical channel as c_m, and at least one unit above the horizontal channel X occupied by $c_0 = w, c_1, \ldots, c_m (= c'_0), c'_1, c'_2, \ldots, c'_s$ (we will give the exact value of the vertical distance between α_t and X a little while later),

- for each k, where $j-1 \leq k \leq t-1$, α_k is one unit above and one unit to the left of α_{k+1}, and
- α_0 is in the same vertical channel as c_{m-1}.

(We will determine later on the horizontal distances between $\alpha_0, \alpha_1, \ldots, \alpha_{j-1}$.)

- For each i, where $0 \leq i \leq j-1$, removing α_{i-1} and α_i, splits K_m into two subgraphs, one containing c_{m-1} and c_m, and another subgraph L'_i. Let L_i be the subgraph of K_m consisting of the vertices of L'_i, α_{i-1} and α_i, and the edges between them. Recursively construct a feasible drawing of L_i with $\overline{\alpha_{i-1}\alpha_i}$ as the base.

- Let $S = \beta_0, \beta_1, \ldots, \beta_\mu$ be an ordered sequence of the neighbors of $\alpha_{j-1}, \alpha_j, \ldots, \alpha_t$ that are not equal to c_{m-1} and c_m, and do not belong to L_{j-1}. In S, we first place the neighbors of α_{j-1}, then of α_j, and so on, finally placing the neighbors of α_t. For each k, where $j-1 \leq k \leq t$, we place the neighbors of α_k into S in the same order as their clockwise order around α_k. Let ε be the index such that the edge $(\beta_{\varepsilon-1}, \beta_\varepsilon)$ is the dual of an edge in P (if there is no such ε, then set $\varepsilon = \mu$). (In Figure 5.4.1, $\varepsilon = 2$.)

- Place $\beta_0, \beta_1, \ldots, \beta_{\varepsilon-1}$ in the same horizontal channel from left-to-right, and place $\beta_\varepsilon, \beta_{\varepsilon+1}, \ldots, \beta_\mu$ in another horizontal channel from right-to-left, such that:

  - $\beta_0, \beta_1, \ldots, \beta_{\varepsilon-1}$ are placed one unit above α_{j-1},
  - $\beta_\varepsilon, \beta_{\varepsilon+1}, \ldots, \beta_\mu$ are placed one unit below α_t,
  - β_0 and β_μ are at either to the right of, or on the same vertical channel as c'_s,

  - $\beta_{\varepsilon-1}$ and β_{ε} are on the same vertical channel, and
  - the distance between $\beta_{\varepsilon-1}$ and β_{ε} is equal to 2 plus the vertical distance between α_{j-1} and α_t.

- For each i, where $0 \leq i \leq \varepsilon - 1$, if there is an edge $e = (\beta_{i-1}, \beta_i)$ in G, then do the following: Notice that removing e from G, split it into two subgraphs, one that contains $\alpha_{j-1}, \alpha_j, \ldots, \alpha_t$, and another subgraph M'_i that does not contain any of them. Let M_i be the subgraph of G consisting of β_{i-1}, β_i, the vertices of M'_i, and the edges between them. Recursively construct a feasible drawing of M_i with $\overline{\beta_{i-1}\beta_i}$ as its base.

- For each i, where $\varepsilon \leq i \leq \mu$, if there is an edge $e = (\beta_{i-1}, \beta_i)$ in G, then do the following: Notice that removing e from G, splits it into two subgraphs, one that contains $\alpha_{j-1}, \alpha_j, \ldots, \alpha_t$, and another subgraph N'_i that does not contain any of them. Let N_i be the subgraph of G consisting of β_{i-1}, β_i, the vertices of N'_i, and the edges between them. Recursively construct a feasible drawing D''_i of N_i with $\overline{\beta_{i-1}\beta_i}$ as its base, and then flip D''_i upside-down.

- Let $(v_{\rho-1}, v_{\rho})$ be the edge of P that is the dual of the edge $(\beta_{\varepsilon-1}, \beta_{\varepsilon})$. Let R be the subgraph of G that corresponds to the subpath $v_{\rho}v_{\rho+1}\ldots v_q$. Construct a beam drawing E of R. For each edge e on the external face of R, do the following: Let $e = (\gamma_1, \gamma_2)$. Removing γ_1 and γ_2 from G splits it into two subgraphs, one containing $\beta_0, \beta_1, \ldots, \beta_{\mu}$, and the other subgraph Q'_e not containing them. Let Q_e be the subgraph of G containing γ_1, γ_2, and the vertices of Q'_e, and the edges between them. If e is on the top or bottom boundary of E, then recursively construct a feasible drawing D_e

of Q_e with $\overline{\gamma_1\gamma_2}$ as its base. If e is on the bottom boundary of E, then flip Q_e upside down. (Note that if e is on the right boundary of E, then Q_e will contain just the edge e because v_q is a leaf of T_G.)

- We are now ready to give the vertical distance between α_t and X: it is equal to $1+\theta$, where θ is maximum height of any of D'_i, D''_i, and D_e, where e is on the bottom boundary of E. Note that this will guarantee that the vertices of each D''_i and D_e will occupy horizontal channels that are either above or the same as the horizontal channel that contains $c_0 = w, c_1, \ldots, c_m (= c'_0), c'_1, c'_2, \ldots, c'_s$. This ensures that there are no crossings between the edges of any D''_i or D_e, and any edge of the form (v, c'_j).

Let $h(n)$ and $w(n)$ be the height and width, respectively, of a feasible drawing D of G with base B, constructed by the Algorithm *OpDraw*. Here, n is the number of vertices in G. Let d be the degree of G. Note that, by the definition of feasible drawings, $w(n)$ will be equal to the horizontal separation between the end-points of B.

It is easy to prove using induction that $w(n) = n-1$ is sufficient. As for the horizontal distances between u and w, between c_{i-1} and c_i (for $1 \le i \le m-1$), between c'_{i-1} and c'_i (for $1 \le i \le s$), between α_{i-1} and α_i (for $1 \le i \le j-1$), between β_{i-1} and β_i (for $1 \le i \le \varepsilon-1$), and between β_{i-1} and β_i (for $\varepsilon+1 \le i \le \mu$), it is sufficient to set them to be equal to $|H|-1$, $|K_i|-1$, $|K'_i|-1$, $|L_i|-1$, $|M_i|-1$, and $|N_i|-1$, respectively. It is also sufficient to set the distance between the end-points of each edge e on the top or bottom boundary of E, to be equal to $|Q_e|-1$.

As for $h(n)$, first notice that, because G has degree d, $t-(j-1)$ is less than $2d$, and hence, the distance between $\beta_{\varepsilon-1}$ and β_{ε} is less than $2d+2$.

Let h' be a function, such that $h'(f) = h(n)$, where f is the number of internal faces in G, i.e., the number of nodes in the dual tree T_G of G.

From the construction of D, we have that:

$$\begin{aligned} h'(f) \leq \; & \max\{\max_{1\leq i\leq s}\{h'(|T_{K'_i}|)\}, \max_{\varepsilon+1\leq i\leq \mu}\{h'(|T_{N_i}|)\}, \max_{edge\ e\ on\ bottom\ boundary\ of\ E}\{h'(|T_{Q_e}|)\}\} \\ & + \max\{h'(|T_H|), \max_{1\leq i\leq m-1}\{h'(|T_{K_i}|)\}, \max_{1\leq i\leq j-1}\{h'(|T_{L_i}|)\}, \max_{1\leq i\leq \varepsilon-1}\{h'(|T_{M_i}|)\}, \\ & \max_{edge\ e\ on\ top\ boundary\ of\ E}\{h'(|T_{Q_e}|)\}\} + O(d), \end{aligned}$$

Since P is a spine of T_G, and

- the dual trees of H, K_i, L_i, M_i, and Q_e (in the case when edge e is on top boundary of E), are either left subtrees of P, or belong to the left subtrees of P, and
- the dual trees of K'_i, N_i, and Q_e (in the case when edge e is on bottom boundary of E), are either right subtrees of P, or belong to the right subtrees of P,

from Lemma 5.3.1, it follows that:

$$h'(f) \leq \max_{f_1^p + f_2^p \leq (1-\delta) f^p}\{h'(f_1) + h'(f_2) + O(d)\}.$$

Using induction, we can show that $h'(f) = O(df^{0.48})$ (see also [4]). Since $f = O(n)$, $h(n) = h'(f) = O(df^{0.48}) = O(dn^{0.48})$.

Theorem 5.4.1 *Let G be an outerplanar graph with degree d and n vertices. We can construct a planar straight-line grid drawing of G with area $O(dn^{1+0.48}) = O(dn^{1.48})$ in $O(n)$ time.*

Proof: Arbitrarily select any edge $e = (u, v)$ on the external face of G, and designate u and v as the poles of G. Let B be any horizontal line-segment with length $n-1$. Let δ be any user-defined constant in the range $(0, 0.0004]$. Construct a feasible drawing of G with base B using Algorithm *OpDraw*. From the discussion given above, it follows immediately that the area of the drawing is $O(dn^{1+0.48}) = O(dn^{1.48})$. It is easy to see the algorithm runs in $O(n)$ time. □

Corollary 5.4.1 *Let G be an outerplanar graph with n vertices and degree $d = O(n^{\delta})$, where $0 \leq \delta < 0.52$ is a constant. We can construct a planar straight-line grid drawing of G with $o(n^2)$ area in $O(n)$ time.*

Chapter 6

Conclusion and Open Problems

The visualization of relational information is concerned with the presentation of abstract information about relationships between various entities. It has many applications in diverse domains such as software engineering, biology, civil engineering, and cartography. Relational information is typically modeled by an abstract graph, where vertices are entities and edges represent relationships between entities. The aim of graph drawing is to automatically produce drawings of graphs which clearly reflect the inherent relational information.

In this thesis, we have investigated problems related to the automatic generation of area-efficient grid drawings of trees and outerplanar graphs, which are important categories of graphs.

In this thesis, we have obtained the following results:

1. An algorithm for producing planar straight-line grid drawings of binary trees with optimal linear area and with user-defined arbitrary aspect ratio,

2. An algorithm for producing planar straight-line grid drawings of degree-d trees with n nodes, where $d = O(n^{\delta})$ and $0 \leq \delta < 1/2$ is a constant, with optimal linear area and with user-defined arbitrary aspect ratio,

3. An algorithm which establishes the currently best known upper bound, namely $O(n \log n)$, on the area of order-preserving planar straight-line grid drawings of ordered trees,

4. An algorithm which establishes the currently best known upper bound, namely $O(n \log \log n)$, on the area of order-preserving planar straight-line grid drawings of ordered binary trees,

5. An algorithm for producing order-preserving upward planar straight-line grid drawings of ordered binary trees with optimal $O(n \log n)$ area,

6. An algorithm which establishes the trade-off between the area and aspect ratio of order-preserving planar straight-line grid drawings of ordered binary trees, in the case when the aspect ratio is arbitrarily defined by the user, and

7. An algorithm for producing planar straight-line grid drawings of outerplanar graphs with n vertices and degree d in $O(dn^{1.48})$ area. This result shows for the first time that a large category of outerplanar graphs, namely those with degree $d = O(n^{\delta})$, where $0 \leq \delta < 0.52$ is a constant, can be drawn in sub-quadratic area.

All our algorithms are time-efficient. More specifically, algorithms 1 and 2 run in $O(n\log n)$ time each, and algorithms 3, 4, 5, 6, and 7 run in $O(n)$ time each.

We have also identified the following open problems, regarding planar straight-line grid drawings of trees and outerplanar graphs:

- Drawing general trees in optimal linear area
  - In this thesis, we have proved that a degree-d tree with n nodes, where $d = O(n^{\delta})$ and $0 \leq \delta < 1/2$ is a constant, can be drawn in optimal linear area and with user-defined arbitrary aspect ratio. So, a natural question is whether this result can be extended to trees with even higher degree.
- Drawing ordered-trees
  - Can we prove lower bounds on non-upward order-preserving drawings, other than the trivial $\Omega(n)$ bound?
  - Does every ordered binary tree admit a non-upward order-preserving drawing in better than $O(n\log\log n)$ area?
  - Does every ordered tree admit a non-upward order-preserving drawing in better than $O(n\log n)$ area?
  - In this thesis, we have studied the trade-off between the area and aspect ratio of order-preserving drawings of ordered binary trees. Can this result be extended to even higher degree trees?
- Drawing outerplanar graphs

– In this thesis, we have proved that an outerplanar graph with n vertices and degree d, can be drawn in $O(dn^{1.48})$ area. If $d = O(n^{\delta})$, where $0 \leq \delta < 0.52$ is a constant, then the graph can be drawn in sub-quadratic area. Can we prove a sub-quadratic area bound on outerplanar graphs with even higher degree?

Bibliography

[1] T. Biedl. Drawing outer-planar graphs in $o(n \log n)$ area. In *Proc. 10th International Symposium on Graph Drawing (GD 2002)*, volume 2528 of *Lecture Notes in Computer Science*, pages 54–65. Springer-Verlag, 2002.

[2] K. Booth and G. Lueker. Testing for the consecutive ones property interval graphs and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13:335–379, 1976.

[3] T. Chan, M. Goodrich, S. Rao Kosaraju, and R. Tamassia. Optimizing area and aspect ratio in straight-line orthogonal tree drawings. *Computational Geometry: Theory and Applications*, 23:153–162, 2002.

[4] T. M. Chan. A near-linear area bound for drawing binary trees. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 161–168, 1999.

[5] Marek Chrobak and Shin ichi Nakano. Minimum-width grid drawings of plane graphs. *Comput. Geom. Theory Appl.*, 11:29–54, 1998.

[6] P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Comput. Geom. Theory Appl.*, 2:187–200, 1992.

[7] P. Crescenzi and P. Penna. Strictly-upward drawings of ordered search trees. *Theoretical Computer Science*, 203(1):51–67, 1998.

[8] P. Crescenzi, P. Penna, and A. Piperno. Linear-area upward drawings of AVL trees. *Comput. Geom. Theory Appl.*, 9:25–42, 1998. (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia).

[9] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[10] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.

[11] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.

[12] G. Di Battista, G. Liotta, and F. Vargiu. Spirality and optimal orthogonal drawings. *SIAM J. Comput.*, 27(6):1764–1811, 1998.

[13] G. Di Battista, R. Tamassia, and I. G. Tollis. Area requirement and symmetry display of planar upward drawings. *Discrete Comput. Geom.*, 7(4):381–401, 1992.

[14] V. Dujmovic and D.R. Wood. Tree-partitions of k-trees with applications in graph layout. Technical Report TR-02-03, School of Computer Science, Carleton University, Ottawa, Canada, 2002.

[15] I. Fary. On straight lines representation of planar graphs. *Acta Sci. Math. Szeged*, 11:229–233, 1948.

[16] S. Felsner, G. Liotta, and S. Wismath. Straight-line drawings on restricted integer grids in two and three dimensions. In *Proc. 9th International Symposium on Graph Drawing (GD 2001)*, volume 2265 of *Lecture Notes in Computer Science*, pages 328–342. Springer-Verlag, 2001.

[17] A. Garg, M. T. Goodrich, and R. Tamassia. Planar upward tree drawings with optimal area. *Internat. J. Comput. Geom. Appl.*, 6:333–356, 1996.

[18] A. Garg and A. Rusu. Straight-line drawings of binary trees with linear area and arbitrary aspect ratio. In *Proc. 10th International Symposium on Graph Drawing (GD 2002)*, volume 2528 of *Lecture Notes in Computer Science*, pages 320–331. Springer-Verlag, 2002.

[19] A. Garg and A. Rusu. Area-efficient order-preserving planar straight-line drawings of ordered trees. In *The Ninth International Computing and Combinatorics Conference (COCOON 2003)*, volume 2697 of *Lecture Notes in Computer Science*, pages 475–486. Springer, 2003.

[20] A. Garg and A. Rusu. Area-efficient planar straight-line grid drawings of outerplanar graphs. In *Proc. 11th International Symposium on Graph Drawing (GD 2003)*, Lecture Notes in Computer Science, 2003. To appear.

[21] A. Garg and A. Rusu. A more practical algorithm for drawing binary trees in linear area with arbitrary aspect ratio. In *Proc. 11th International Symposium on Graph Drawing (GD 2003)*, Lecture Notes in Computer Science, 2003. To appear.

[22] A. Garg and A. Rusu. Straight-line drawings of general trees with linear area and arbitrary aspect ratio. In *Proc. The 2003 International Conference on Computational Science and Its Applications (ICCSA 2003)*, volume 2669 of *Lecture Notes in Computer Science*, pages 876–885. Springer, 2003.

[23] J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[24] D. E. Knuth. Computer drawn flowcharts. *Commun. ACM*, 6, 1963.

[25] K. Kuratowski. Sur le probleme des courbes gauches en topologie. *Fund. Math.*, 16:271–283, 1930.

[26] C. E. Leiserson. Area-efficient graph layouts (for VLSI). In *Proc. 21st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 270–281, 1980.

[27] W. Lenhart and G. Liotta. The drawability problem for minimum weight triangulations. *Theoretical Computer Science*, 270(1):261–286, 2002.

[28] Achilleas Papakostas and Ioannis G. Tollis. Algorithms for area-efficient orthogonal drawings. *Comput. Geom. Theory Appl.*, 9(1–2):83–110, 1998. Special Issue on Geometric Representations of Graphs, G. Di Battista and R. Tamassia, editors.

[29] H. C. Purchase, R. F. Cohen, and M. I. James. An experimental study of the basis for graph drawing algorithms. *ACM J. Experim. Algorithmics*, 2(4), 1997.

[30] Helen Purchase. Which aesthetic has the greatest effect on human understanding? In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 248–261. Springer-Verlag, 1997.

[31] Md. Saidur Rahman, Shin ichi Nakano, and Takao Nishizeki. Rectangular grid drawings of plane graphs. *Comput. Geom. Theory Appl.*, 10(3):203–220, 1998.

[32] M. Sarkar and M. H. Brown. Graphical fisheye views. *Commun. ACM*, 37(12):73–84, 1994.

[33] W. Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Sympos. Discrete Algorithms*, pages 138–148, 1990.

[34] C.-S. Shin, S. K. Kim, and K.-Y. Chwa. Area-efficient algorithms for upward straight-line tree drawings. In *Proc. 2nd Ann. Internat. Conf. Computing and Combinatorics*, volume 1090 of *Lecture Notes Comput. Sci.*, pages 106–116. Springer-Verlag, 1996.

[35] C.-S. Shin, S.K. Kim, S.-H. Kim, and K.-Y. Chwa. Area-efficient algorithms for straight-line tree drawings. *Comput. Geom. Theory Appl.*, 15:175–202, 2000.

[36] S. K. Stein. Convex maps. *Proc. Amer. Math. Soc.*, 2(3):464–466, 1951.

[37] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.

[38] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, SMC-18(1):61–79, 1988.

[39] R. Tamassia and I. G. Tollis. Efficient embedding of planar graphs in linear time. In *Proc. IEEE Internat. Sympos. on Circuits and Systems*, pages 495–498, 1987.

[40] R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. Circuits Syst.*, CAS-36(9):1230–1234, 1989.

[41] R. Tamassia and I. G. Tollis. Tessellation representations of planar graphs. In *Proc. 27th Allerton Conf. Commun. Control Comput.*, pages 48–57, 1989.

[42] L. Trevisan. A note on minimum-area upward drawing of complete and Fibonacci trees. *Inform. Process. Lett.*, 57(5):231–236, 1996.

[43] L. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Comput.*, C-30(2):135–140, 1981.

[44] K. Wagner. Bemerkungen zum Vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.

[45] K. Wagner. Uber eine eigenschaft der ebene komplexe. *Math. Ann.*, 114:570–590, 1937.