8

Tree Drawing Algorithms

Adrian Rusu
Rowan University

8.1 Introduction

Tree drawing is concerned with the automatic generation of geometric representations of relational information, often for visualization purposes. The typical data structure for modeling hierarchical information is a tree whose vertices represent entities and whose edges correspond to relationships between entities. Visualizations of hierarchical 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.

The automatic generation of drawings of trees finds many applications, such as Software Engineering (program nesting trees, object-oriented class hierarchies), Business Administration (organization charts), Decision Support Systems (activity trees), Artificial Intelligence (knowledge-representation isa hierarchies), Logic Programming (SLD-trees), Web-site Design and Browsing (structure of a Web-site), Biology (evolutionary trees), and Chemistry (molecular drawings).

Algorithms for drawing trees are typically based on some graph-theoretic insight into the structure of the tree. The input to a tree drawing algorithm is a tree T that needs to be drawn. The output is a *drawing* Γ which 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.

T is an *ordered* tree if the children of each node are assigned a fixed left-to-right order. For any node u in T, its *leftmost child* (*rightmost child*) is the one that comes first (last) in the left-to-right ordering of the children of u in T. 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

0-8493-8597-0/01/$0.00+$1.50

© 2004 by CRC Press, LLC

they have the same parent. The *subtree* of T rooted at a node v consists of v and all the descendants of v. T is the *empty tree* if it has zero nodes in it.

Let v be a node of an ordered tree. Then $n(v)$, $p(v)$, $l(v)$, $r(v)$, and $s_1(v), \ldots, s_i(v)$, are the number of nodes in the subtree rooted at v, parent, leftmost child, rightmost child, and siblings of v, respectively.

The rest of the chapter is organized as follows. After motivating the need for tree drawing algorithms and providing drawing conventions and aesthetics in this section, we describe the main approaches for tree drawing algorithms in subsequent sections. We then present some of the most representative algorithms for drawing binary and general trees.

8.1.1 Drawing Conventions

A *drawing convention* is a basic rule that a drawing must satisfy to be admissible [DETT99]. A list of the most used drawing conventions for drawing trees and their significance is given below (see Figure 8.1):

Polyline Drawings

A *polyline drawing* is a drawing in which 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* (see Figure 8.1(a)).

Orthogonal Drawings

An *orthogonal drawing* is one in which each edge is drawn as a chain of alternating horizontal and vertical segments (see Figure 8.1(b)).

Upward and Non-Upward Drawings

An *upward drawing* is defined as a drawing where no child is placed higher in the y-direction than its parent (see Figure 8.1(a),(c)). A *non-upward drawing* is a drawing which is not upward (see Figure 8.1(b),(d)).

Grid Drawings

A *grid drawing* is one in which each vertex is placed at integer coordinates. Assuming 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 computer screen can be viewed as a grid of pixels placed at integer coordinates. 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 rendered in a (large-enough) grid-based display surface, such as a computer screen, without any distortions due to truncation and round-off errors. The smallest rectangle with horizontal and vertical sides parallel to the axes, that covers the entire grid drawing, is called the *enclosing rectangle*.

Planar Drawings

A *planar drawing* is a drawing in which edges do not intersect each other in the drawing (for example, the drawings (a), (b), and (c) in Figure 8.1 are planar drawings, and the drawing (d) is a non-planar drawing). Planar drawings are normally easier to understand than non-planar drawings, i.e. drawings with edge-crossings. Since any tree admits a planar drawing, it is desirable to obtain planar drawings for trees.

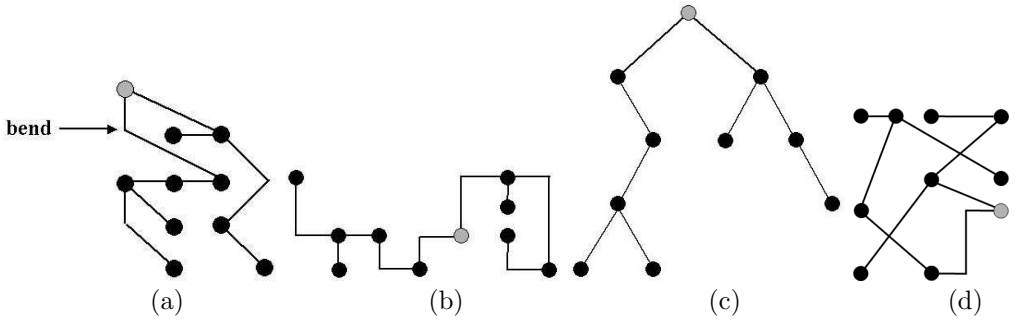


Figure 8.1 Various kinds of drawings of the same tree: (a) polyline, (b) orthogonal, (c) straight-line, (d) non-planar. Also note that the drawings shown in Figures (a) and (c) are upward drawings, whereas the drawings shown in Figures (b) and (d) are not. The root of the tree is shown as a shaded circle, whereas other nodes are shown as black circles.

Straight-line Drawings

The so called *straight-line* tree drawings have each edge drawn as a straight line segment (see Figure 8.1(c)). It is natural to draw each edge of a tree as a straight line between its end-nodes. Straight-line drawings are easier to understand than polyline drawings.

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 [TDB88, Pur97, PCJ97, Pur00]. 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.

8.1.2 Aesthetics

Aesthetics specify graphic properties of the drawing that we would like to apply as much as possible. Most of the tree drawing algorithms have concentrated on drawing trees in as small as possible area with user-controlled aspect ratio. A list of the most important aesthetics of drawings of trees is given below:

- **Area**: 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. Note that we cannot discuss the area of non-grid drawings (i.e. drawings that have the nodes placed at real coordinates), since, by placing the nodes closer or farther, such a drawing can be scaled down or up by any value.
- **Aspect Ratio**: The *aspect ratio* of a grid drawing is defined as the ratio of the length of the shortest side to the length of the longest side of its enclosing rectangle. An aspect ratio is considered *optimal* if it is equal to 1. 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.
- **Subtree Separation**: Let $T[v]$ be the subtree rooted at node v of tree T. $T[v]$ consists of v and all the descendants of v. A drawing of T has the *subtree-separation* property [CGKT97] 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. Focus+context [SB94] is a style in which part of the information is presented in detail (the focus) while the rest is still available, but at a smaller size (the context). The subtree-separation property allows for a focus+context style rendering of a 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.

- **Closest Leaf**: The *closest leaf* is defined as the smallest euclidean distance between the root of the tree and a leaf in the drawing [RS08].
- **Farthest Leaf**: The *farthest leaf* is defined as the largest euclidean distance between the root of the tree and a leaf in the drawing [RS08].

The aesthetics closest leaf and farthest leaf help determine whether the algorithms place leaves close or far from the root. It is important to minimize the distance between the root and the leaves of the tree, especially in the case when the user needs to visually analyze the information contained in the levels close to the root and levels close to the leaves, without the information in between. Such a case appears in particular for algorithms where a change at the top level (root) of the tree generates modifications at the bottom levels (leaves) of the tree (for example, usual operations - find, insert, remove - on binary search trees, splay trees, or B^+ trees).

Other well-known aesthetics which have been used in various tree drawing studies are as follows [DETT99]:

- **Size**: the longest side of the smallest rectangle with horizontal and vertical sides covering the drawing.
- **Total Edge Length**: the sum of the lengths of the edges in the drawing.
- **Average Edge Length**: the average of the lengths of the edges in the drawing.
- **Maximum Edge Length**: the maximum among the lengths of the edges in the drawing.
- **Uniform Edge Length**: the variance of the edge lengths in the drawing.
- **Angular Resolution**: the smallest angle formed by two edges incident on the same node.
- **Symmetry**: visual identification of symmetries in the drawing.

It is widely accepted [DETT94, DETT99, Pur97, PCJ97] that small values of the size, total edge length, average edge length, maximum edge length, and uniform edge length are related to the perceived aesthetic appeal and visual effectiveness of the drawing. High angular resolution is desirable in visualization applications and in the design of optical communication networks. For binary trees, the degree of a node is at most three, hence a trivial upper bound on the angular resolution is 120°. Given a symmetric drawing, a conceptual understanding of the entire tree can be built up from that of a smaller subtree, replicated a number of times.

8.2 Level-Based Approach

The *Level-Based Approach* can be used on both binary and general trees and it is characterized by the fact that in the drawings produced, the nodes at the same distance from the root are horizontally aligned. Algorithms based on this approach are usually simple to understand and implement and produce intuitive drawings which exhibit clear display of

symmetries. However, these algorithms have two disadvantages: the drawing has an area of $\Omega(n^2)$ and, for balanced trees with many nodes, the width is much larger than the height.

Level-based algorithms have been designed previously [Blo93, RT81, BJL02, Wal90]. The algorithms described in [BJL02, Wal90] achieve better area, but they do not exhibit the subtree separation property.

A recursive algorithm for binary trees [RT81], which exhibits the subtree separation property, uses the following steps: draw the subtree rooted at the left child, draw the subtree rooted at the right child, place the drawings of the subtrees at horizontal distance 2, and place the root one level above and halfway between the children. If there is only one child, place the root at horizontal distance 1 from the child. A drawing produced by this algorithm is provided in Figure 8.2.

Figure 8.2 Drawing of the Fibonacci tree with 88 nodes, generated by the Level-Based algorithm of [RT81].

By using a geometric transformation (cartesian $\rightarrow$ polar), level drawings yield *radial drawings*, where nodes are placed on concentric circles by level (see Figure 8.3).

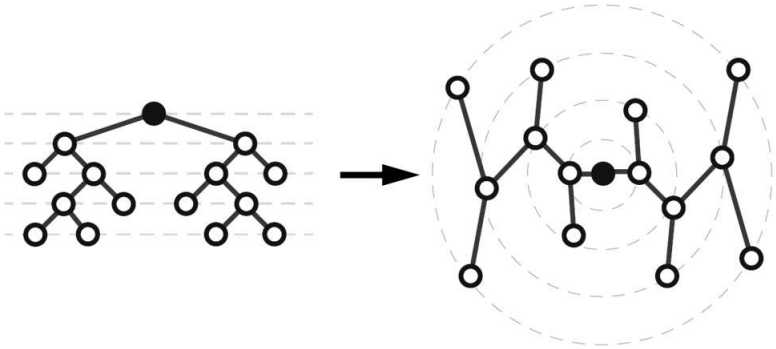


Figure 8.3 Example of a transformation from a level drawing to a radial drawing.

Radial drawings are often used in drawing graphs, even though they do not always guarantee planarity. Several algorithms for radial drawings of trees have been designed and some of them have also been used in various applications [Ber81, Ead92, CPM+98, CPP00, BM03, Bac07].

8.3 H-V Approach

The *Horizontal-Vertical Approach* can be used on both binary and general trees. In this approach, a divide-and-conquer strategy is used to recursively construct an upward, orthogonal, and straight-line drawing of a tree, by placing the root of the tree in the top-left corner, and the drawings of its left and right subtrees one next to the other (*horizontal composition*) or one below the other (*vertical composition*) (see Figure 8.3). The resulting drawing also exhibits the subtree separation property within an $O(n \log n)$ area.

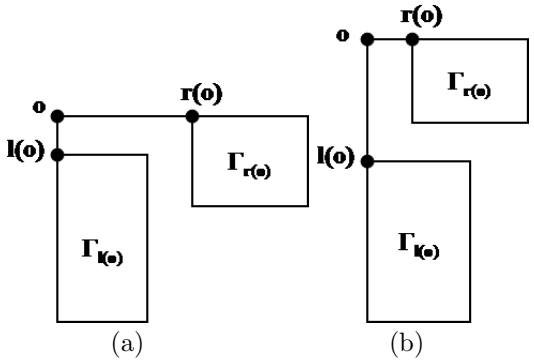


Figure 8.4 General H-V approach. (a) *Horizontal composition*: the drawings of the subtrees rooted at the children of o are placed one next to the other. (b) *Vertical composition*: the drawings of the subtrees rooted at the children of o are placed one below the other.

Various H-V algorithms can be obtained depending on which layout is used and what other conditions are imposed on the drawing. An algorithm using this approach has been developed for binary trees [CDP92]. This algorithm places the drawing of the subtree with the greater width one unit below the drawing of the subtree with the smaller width (see Figure 8.3(b)). A modification of this algorithm, in which vertical and horizontal combinations are used alternatively, produces area-efficient drawings of complete, AVL, and Fibonacci trees. The algorithm can easily be extended to general trees.

8.4 Path-Based Approach

The *Path-Based Approach* uses a recursive winding paradigm to draw a binary tree T by laying down a small chain of nodes monotonically in the x-direction leading to a distinguished node v, and then "winding" by recursively laying out the subtrees rooted at the children of v in the opposite direction.

Several path-based algorithms have been designed [CGKT02, GR03a, SKC00].

Recursively, for every subtree rooted at a node v, a parameter A is fixed, so that, if $n(v) \leq A$ then the drawings of the subtrees rooted at the children of v are placed one next to the other, as in Figure 8.5 (a). Otherwise, the subtree looks like Figure 8.5 (b), where v_1 is the root of the subtree, $v_{i+1} = r(v_i)$ for $i \geq 1$, $k \geq 1$ is the first index for which $n(v_k) > n - A$ and $n(v_{k+1}) \leq n - A$, T_i is the subtree rooted at $l(v_i)$, $T' = l(v_k)$, and $T'' = r(v_k)$. In the second case, depending on whether an upward or a non-upward drawing is to be obtained, the drawings are placed as in Figures 8.6(a) and 8.6(b), respectively.

The user controls the aspect ratio by modifying parameter A.

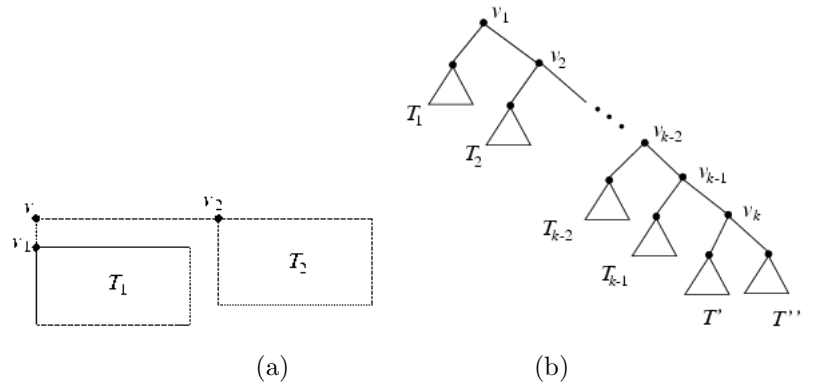


Figure 8.5 (a) When $n(v) \leq A$, the subtrees are placed one next to the other. (b) When $n(v) > A$, the tree is divided into subtrees $T_1, T_2, \ldots, T_{k-2}, T_{k-1}, T', T''$.

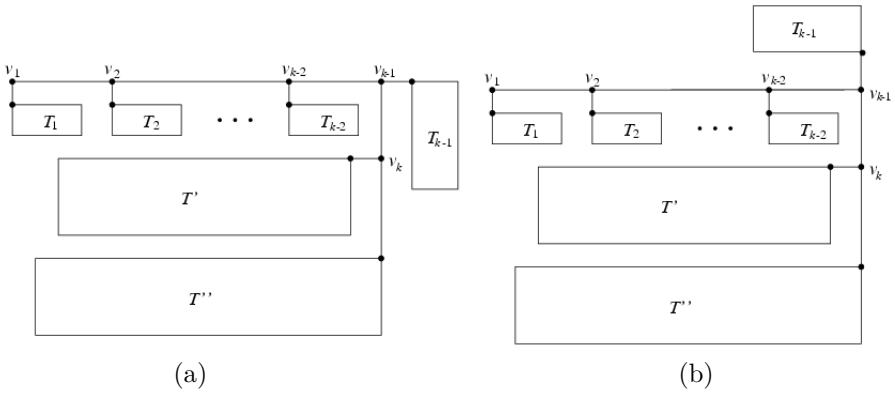


Figure 8.6 (a) Upward drawing of binary tree T. (b) Non-upward drawing of binary tree T.

A drawing of the Fibonacci tree with 88 nodes produced by the algorithm of Chan et al. [CGKT02], with the value for the parameter A at one of the extremes, is provided in Figure 8.7. This algorithm produces the best worst-case theoretical bound on area for path-based algorithms: $O(n \log \log n)$.

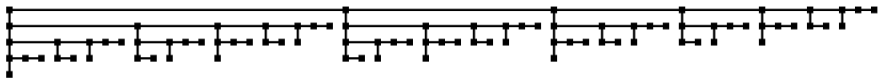


Figure 8.7 Drawing of Fibonacci tree with 88 nodes produced by the path-based algorithm [CGKT02], with parameter A at one of the extremes: $A = 88$.

8.5 Ringed Circular Layout Approach

In these algorithms, children are placed on the circumference or the interior of a circle centered at their parents [GADM04, CC99, Ead92, MH98, MMC99, TM02, RSJ07]. In general, these algorithms are used to draw high-degree trees. However, the resulting drawings are

often not planar. An example of the general idea of the approach is provided in Figure 8.5.

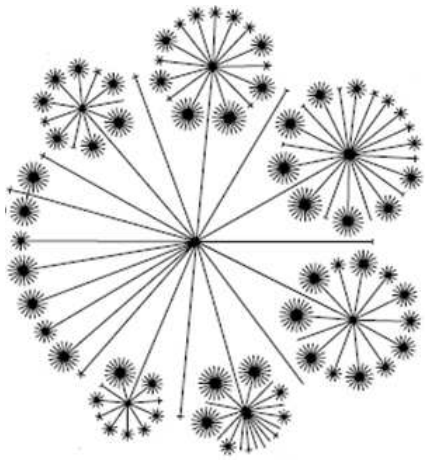


Figure 8.8 General idea for the ringed circular layout approach.

Cone trees [RMC91] are a 3D extension of the 2D ringed circular layout approach. In cone trees, the parent is located at the tip of a cone, and its children are spaced equally on the bottom circle of the cone.

8.6 Separation-Based Approach

The *Separation-Based Approach* can be used on both binary and general trees. Separation-based algorithms have been designed [GR02, GR03b, GR03c, RS07]. In this approach, a divide-and-conquer strategy is used to recursively construct a drawing of a tree, by performing the following actions at each recursive step:

- *Find a Separator Edge or a Separator Node*: A *separator edge* (*node*) of a tree T with $degree(T) = d$ is an edge (node), which, if removed, divides T into at most d smaller, partial, trees. It has been shown that every tree contains a separator edge or a separator node [GR03c, Val81].
- *Split Tree*: Split T into at most d partial trees by removing a separator edge or a separator node.
- *Assign Aspect Ratios*: Pre-assign a desirable aspect ratio to each partial tree.
- *Draw Partial Trees*: Recursively construct a drawing of each partial tree using its pre-assigned aspect ratio.
- *Compose Drawings*: Arrange the drawings of the partial trees, and draw the nodes and edges, that were removed from the tree to divide it, such that the drawing of the tree thus obtained meets certain aesthetics.

8.7 Algorithms for Drawing Binary Trees

A *binary tree* is one where each node has at most two children. Most of the research on drawing trees targets binary trees, hence, in this section, several algorithms for drawing

binary trees are presented.

Binary trees have a strong connection to real-life applications. For instance, binary trees represent programs in combinatory logic, which is under investigation as an approach to nanostructure synthesis and control [Mac03]. The idea is to use molecular processes to implement the combinatory logic tree substitution operations, so that the molecular reorganization of the trees results in the desired stucture or process. Visualization of these binary trees could improve the investigator's ability in interpreting the substitution operations involved in combinatory logic.

8.7.1 Theoretical Results

We summarize some known theoretical results on planar grid drawings of binary trees. Let T be an n-node binary tree. Garg et al. [GGT96] presents an algorithm for constructing an upward polyline drawing of T with $O(n)$ area, and any user-specified aspect ratio in the range $[n^{-\epsilon}, n^{\epsilon}]$, where ϵ is any constant, such that $0 < \epsilon < 1$. It also shows that $n \log\log n$ is a tight bound for the area of upward orthogonal polyline drawings, 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. Leiserson [Lei80] and Valiant [Val81] present algorithms for constructing a (non-upward) orthogonal polyline drawing of T with $O(n)$ area. Chan et al. [CGKT02] 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 $[1, n/\log n]$. It also shows that $n \log n$ is a tight bound for such drawings. Shin et al. [SKC00] gives an algorithm for constructing an upward straight-line drawing of T with $O(n \log\log n)$ area. Chan et al. [CGKT02] and Shin et al. [SKC00] show that T admits a non-upward planar straight-line orthogonal grid drawing with height $O(n/A) \log A$ and 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 $A = \log n$). If T is a Fibonacci tree, (AVL tree, and complete binary tree), then Crescenzi et al. [CDP92] and Trevisan [Tre96] (Crescenzi et al. [CPP98, CDP92], respectively) give algorithms for constructing an upward straight-line drawing of T with $O(n)$ area. Garg and Rusu [GR04] present an algorithm for constructing a (non-upward) straight-line drawing of T with $O(n)$ area, and any user-specified aspect ratio in the range $[n^{-\epsilon}, n^{\epsilon}]$, where ϵ is any constant, such that $0 < \epsilon < 1$. This is trivially a tight bound, as any straight-line drawing of a binary tree with n nodes requires $\Omega(n)$ area.

Table 8.1 summarizes these results.

As for order-preserving algorithms, Shin et al. [SKC00] 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 $O(n(\log\log n)^2)$. Crescenzi et al. [CDP92], Crescenzi and Penna [CP98], and Trevisan [Tre96] give order-preserving planar upward straight-line grid drawings of complete, logarithmic, and Fibonacci trees, respectively, with area $O(n)$. Dolev and Trickey [DT81] prove that binary trees admit $\Theta(n)$ area order-preserving orthogonal drawings. Kim [Kim95] shows an upper bound of $O(n \log n)$ area for upward order-preserving orthogonal drawings of ternary trees (trees whose nodes have at most three children), result that immediately extends to binary trees. This area bound is optimal, as Garg et al. [GGT96] demonstrates a lower bound of $O(n \log n)$ area for such drawings of binary trees. Crescenzi et al. [CDP92] gives an algorithm that achieves $O(n^2)$ area for upward orthogonal straight-line order-preserving drawings of binary trees. Frati [Fra07] proves that this bound is optimal. Frati [Fra07] also gives the best known upper bound of $O(n^{1.5})$ area for non-upward orthogonal straight-line order-preserving drawings of binary trees. It is unknown whether this is an optimal bound,

Drawing Type	Area	Aspect Ratio	Reference
Upward Orthogonal Polyline	$O(n \log \log n)$	$\Theta(\log^2 n/(n \log \log n))$	[GGT96]
(Non-upward) Orthogonal Polyline	$O(n)$	$\Theta(1)$	[Lei80, Val81]
Upward Orthogonal Straight-line	$O(n \log n)$	$[1, n/\log n]$	[CDP92, CGKT02]
(Non-upward) Orthogonal Straight-line	$O(n \log \log n)$	$\Theta(\log^2 n/(n \log \log n))$	[CGKT02, SKC00]
Upward Polyline	$O(n)$	$[n^{-\epsilon}, n^{\epsilon}]$	[GGT96]
Upward Straight-line	$O(n \log \log n)$	$\Theta(\log^2 n/(n \log \log n))$	[SKC00]
(Non-upward) Straight-line	$O(n)$	$[n^{-\epsilon}, n^{\epsilon}]$	[GR04]

Table 8.1 Bounds on the areas and aspect ratios of various kinds of planar grid drawings of an n-node unordered binary tree. Here, ϵ is an arbitrary constant, such that $0 < \epsilon < 1$.

as the trivial $O(n)$ is the lower bound currently known. Garg et al. [GGT96] provides an algorithm that constructs an upward polyline order-preserving drawing of a binary tree with $O(n \log n)$ area, which is the optimal bound for such drawings [CDP92]. Kim [Kim04] improves the number of bends from $O(n)$ to $O(n/\log n)$, while matching the area bound. Garg and Rusu [GR03a] show that a binary tree admits an order-preserving planar straight-line grid drawing with $O(n \log \log n)$ area. In addition, they show that a binary tree admits an order-preserving upward planar straight-line drawing with optimal $O(n \log n)$ area.

Table 8.2 summarizes these results.

A variety of results are available for other kinds of drawings. Di Battista et al. [DETT99] and Frati [Fra09] have given a survey of these results.

8.7.2 Experimental Analysis

Experimental studies provide insight into the behavior of tree drawing algorithms beyond their targetted aesthetic criteria. In a comprehensive experimental study [RS08], separation-based algorithm by Garg and Rusu [GR04], path-based algorithm by Chan et al. [CGKT02], level-based algorithm by Reingold and Tilford [RT81], and ringed circular layout algorithm by Teoh and Ma [TM02], were compared on a large suite of seven types of binary trees of various sizes, based on ten quality measures: area, aspect ratio, size, total edge length, average edge length, maximum edge length, uniform edge length, angular resolution, closest leaf, and farthest leaf. As the specific algorithms compared are intended to be representative of their respective approaches, it is expected that the results generally apply to other algorithms using the same approach, and even extend to trivial extensions to general trees.

This experimental analysis includes some interesting findings:

- The performance of a drawing algorithm on a tree-type is not a good predictor of the performance of the same algorithm on other tree-types: some of the algorithms perform best on a tree-type, and worst on other tree-types.
- Reingold-Tilford algorithm [RT81] scores worse in comparison to the other chosen algorithms for almost all ten aesthetics considered.
- The intuition that low average edge length and area go together is contradicted in only one case.
- The intuitions that average edge length and maximum edge length, uniform edge

Tree Type	Drawing Type	Area	Aspect Ratio	Ref.
Complete	Upward Straight-line Order-preserving	$\Theta(n)$	$O(1)$	[CDP92]
Fibonacci	Upward Straight-line Order-preserving	$\Theta(n)$	$O(1)$	[Tre96]
Special Balanced Binary Trees such as Red-black	Upward Straight-line Order-preserving	$O(n(\log\log n)^2)$	$n/\log^2 n$	[SKC00]
Logarithmic	Upward Straight-line Order-preserving	$\Theta(n)$	$O(1)$	[CP98]
Binary	Upward Orthogonal Polyline Order-preserving	$O(n\log n)$	$\Theta(\log^2 n/(n\log\log n))$	[Kim95, GGT96]
	Non-upward Orthogonal Polyline Order-preserving	$O(n)$	$(9a+8)/(9b+8)$	[DT81]
	Upward Orthogonal Straight-line Order-preserving	$\Theta(n^2)$	$O(1)$	[CDP92, Fra07]
	Non-upward Orthogonal Straight-line Order-preserving	$O(n^{1.5})$	$O(\sqrt(n)/n)$	[Fra07]
	Upward Polyline Order-preserving	$O(n\log n)$	$\log n/n$	[Kim04]
		$O(n\log n)$	$\Theta(\log^2 n/(n\log\log n))$	[GGT96, CDP92]
	Non-upward Polyline Order-preserving	$O(n\log\log n)$	$(n\log\log n)/\log^2 n$	[GR03a]
	Upward Straight-line Order-preserving	$\Theta(n\log n)$	$n/\log n$	[GR03a]
	Non-upward Straight-line Order-preserving	$O(n\log n)$	$[1, n/\log n]$	[GR03a]
		$O(n\log\log n)$	$(n\log\log n)/\log^2 n$	[GR03a]

Table 8.2 Bounds on the areas and aspect ratios of various kinds of order-preserving planar grid drawings of an n-node ordered tree. Here, $ab \leq kn$, where k is some constant.

length and total edge length, and short maximum edge length and close farthest leaf go together are contradicted for unbalanced binary trees.

- With regards to area, of the four algorithms studied, three perform best on different types of trees.
- With regards to aspect ratio, of the four algorithms studied, three perform well on trees of different types and sizes.
- Not all algorithms studied perform best on complete binary trees even though they have one of the simplest tree structures.
- The level-based algorithm of Reingold-Tilford [RT81] produces much worse aspect ratios than algorithms designed using other approaches.
- The path-based algorithm of Chan et al. [CGKT02] tends to construct drawings with better area at the expense of worse aspect ratio.

8.7.3 Unordered Trees

In this section, we present the algorithm of [GR04] in more detail. This algorithm uses a Separation-Based Approach (therefore we call it *Separation*), and achieves optimal linear area for planar straight-line grid drawings, while at the same time, giving the user control over the aspect ratio. In addition, the drawings produced by this algorithm exhibit the subtree separation property.

Let T be a tree with root o. Let n be the number of nodes in T. A *partial tree* of T is a connected subgraph of T.

For some trees, the algorithm designates a special *link* node u^*, that has at most one child.

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

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

Let A and ϵ be two numbers, where ϵ is a constant, such that $0 < \epsilon < 1$, and $n^{-\epsilon} \leq A \leq n^{\epsilon}$. A is called the *desirable aspect ratio* for T.

Theorem 8.1 *[Separator Theorem [Val81]] Every binary tree T with n nodes, where $n \geq 2$, contains an edge e, called a* separator edge*, such that removing e from T splits it into two non-empty trees with n_1 and n_2 nodes, respectively, such that for some x, where $1/3 \leq x \leq 2/3$, $n_1 = xn$, and $n_2 = (1-x)n$. Moreover, e can be found in $O(n)$ time.*

The algorithm takes ϵ, A, and T as input, and uses a divide-and-conquer strategy to recursively construct a feasible drawing Γ of T, by performing the following actions at each recursive step:

- *Split Tree*: Split T into at most five partial trees by removing at most two nodes and their incident edges from it. Each partial tree has at most $(2/3)n$ nodes. Based on whether the separator edge is on the leftmost path of T or not, there are two general cases, which are shown in Figure 8.9.
- *Assign Aspect Ratios*: Correspondingly, assign a desirable aspect ratio A_k to each partial tree T_k. The value of A_k is based on the value of A and the number of nodes in T_k.
- *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 is a feasible drawing. Note that the arrangement of these drawings is done based on the cases shown in Figure 8.9. In each case, if $A < 1$, then the drawings of the partial trees are stacked one above the other, and if $A \geq 1$, then they are placed side-by-side.

Remark: The drawing Γ constructed by the algorithm may not have aspect ratio exactly equal to A, but it fits inside a rectangle with area $O(n)$ and aspect ratio A.

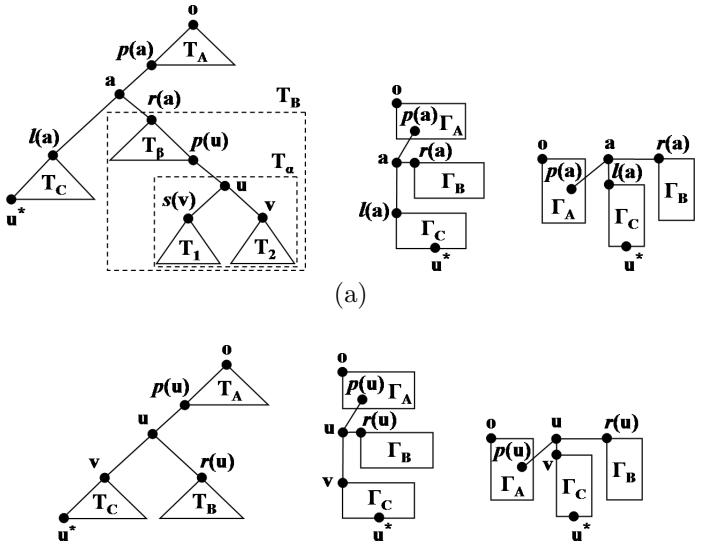


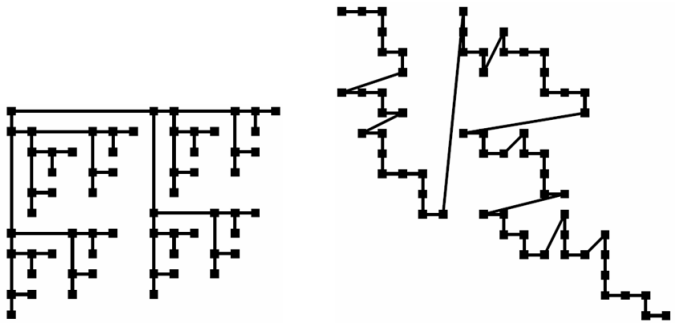
Figure 8.9 (a) Drawing T in Case 1 (when the separator (u, v) is not in the leftmost path of T). (b) Drawing T in Case 2 (when the separator (u, v) is in the leftmost path of T). For each case, first is shown the structure of T for that case, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. For simplicity, $p(a)$ and $p(u)$ are shown to be in the interior of Γ_A, but actually, either they are the same as o, or if $A < 1$ ($A \geq 1$), then they are placed at the bottom (right) boundary of Γ_A. For simplicity, Γ_A, Γ_B, and Γ_C are shown as identically sized boxes, but in actuality, they may have different sizes.

Figure 8.10 (a) shows a drawing of a complete binary tree with 63 nodes constructed by algorithm *Separation*, with $A = 1$ and $\epsilon = 0.5$. Figure 8.10 (b) shows a drawing of a tree with 63 nodes, consisting of a single path, constructed by algorithm *Separation*, with $A = 1$ and $\epsilon = 0.5$.

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 8.1, 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, there are two general cases (each with several subcases - not covered here):
  - *Case 1: The separator edge (u, v) is not in the leftmost path of T.* Let o be the root of T. Let a be the last node common to the path $o \rightsquigarrow v$, and the leftmost path of T. Let partial trees T_A, T_B, T_C, T_α, T_β, T_1 and T_2 be defined as follows (see Figure 8.9 (a)):



(a) (b)

Figure 8.10 (a) Drawing of the complete binary tree with 63 nodes constructed by Algorithm *Separation*, with $A = 1$ and $\epsilon = 0.5$. (b) Drawing of a tree with 63 nodes, consisting of a single path, constructed by Algorithm *Separation*, with $A = 1$ and $\epsilon = 0.5$.

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

Nodes a and u and their incident edges are being removed to split T into at most five partial trees T_A, T_C, T_β, T_1, and T_2. $p(a)$ is designated as the link node of T_A, $p(u)$ as the link node of T_β, and u^* as the link node of T_C. Arbitrarily select a leaf of T_1, and a leaf of T_2, and designate them as the link nodes of T_1 and T_2, respectively.

– *Case 2: The separator edge (u, v) is in the leftmost path of T.* Let o be the root of T. Let partial trees T_A, T_B, and T_C be defined as follows (see Figure 8.9 (b)):

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

Node u and its incident edges are being removed to split T into at most three partial trees T_A, T_B, and T_C. $p(u)$ is designated as the link node of T_A, and u^* as the link node of T_C. Arbitrarily select a leaf of T_B and designate it as the link node of T_B.

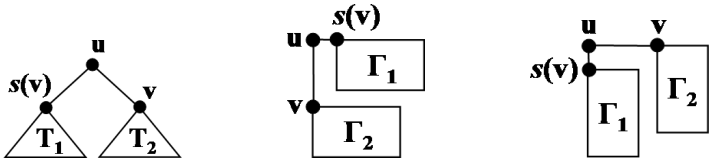


Figure 8.11 Drawing T_α in the general case ($u \neq a$ and $T_1 \neq \emptyset$). First is shown the structure of T_α, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. For simplicity, Γ_1 and Γ_2 are shown as identically sized boxes, but in actuality, their sizes may be different.

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 the number of nodes in T_k.
Definition: T_k is a *large* partial tree of T if:

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

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

In Step *Assign Aspect Ratios*, a desirable aspect ratio A_k is assigned to each non-empty T_k as follows: Let $x_k = n_k/n$.

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

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

Draw Partial Trees

If $A \geq 1$, then the values of A_A and A_β (A_A and A_β are the desirable aspect ratios for T_A and T_β, respectively) are being changed to $1/A_A$ and $1/A_\beta$, respectively. This is done so because later in Step *Compose Drawings*, when constructing Γ, if $A \geq 1$, then the drawings of T_A and T_β are rotated by 90°. Drawing T_A and T_β with desirable aspect ratios $1/A_A$ and $1/A_\beta$, respectively, compensates for the rotation, and ensures that the drawings of T_A and T_β that eventually get placed within Γ are those with desirable aspect ratios A_A and A_β, respectively.

Next, each non-empty partial tree T_k, $k \in \{A, B, C, \alpha, \beta, 1, 2\}$, is drawn recursively with A_k as its desirable aspect ratio. The base case for the recursion happens when T_k contains exactly one node, in which case, the drawing of T_k is simply the one consisting of exactly one node.

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 Case 1.

In Case 1, first a drawing Γ_α of the partial tree T_α is constructed by composing Γ_1 and Γ_2 as shown in Figure 8.11, then a drawing Γ_B of T_B is constructed by composing Γ_α and Γ_β as shown in Figure 8.12, and finally Γ is constructed by composing Γ_A, Γ_B and Γ_C as shown in Figure 8.9 (a).

In the general case ($u \neq a$ and $T_1 \neq \emptyset$), Γ_α is constructed as follows (see Figure 8.11):

- If $A < 1$, then Γ_1 is placed above Γ_2 such that the left boundary of Γ_1 is one unit to the right of the left boundary of Γ_2; u is placed in the same vertical channel as v and in the same horizontal channel as $s(v)$.
- If $A \geq 1$, then Γ_1 is placed 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; u is placed in the same vertical channel as $s(v)$ and in the same horizontal channel as v.

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

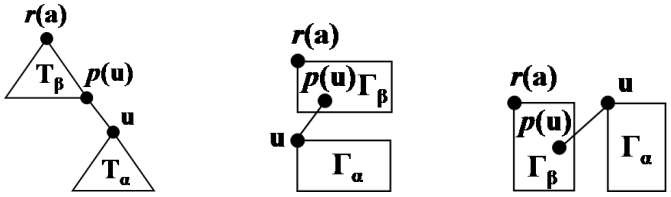


Figure 8.12 Drawing T_B in the general case ($T_\beta \neq \emptyset$). First is shown the structure of T_B, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. For simplicity, $p(u)$ is shown to be in the interior of Γ_β, but actually, it is either same as $r(a)$, or if $A < 1$ ($A \geq 1$), then is placed on the bottom (right) boundary of Γ_β. For simplicity, Γ_β and Γ_α are shown as identically sized boxes, but in actuality, their sizes may be different.

In the general case ($T_\beta \neq \emptyset$), Γ_B is constructed as follows (see Figure 8.12):

- if $A < 1$, then Γ_β is placed one unit above Γ_α such that the left boundaries of Γ_β and Γ_α are aligned.
- If $A \geq 1$, then first Γ_β is rotated clockwise by 90° and then flipped right-to-left, then Γ_β is placed one unit to the left of Γ_α such that the top boundaries of Γ_β and Γ_α are aligned.

Draw edge $(p(u), u)$.

In general Case 1, Γ is constructed from Γ_A, Γ_B, and Γ_C as follows (see Figure 8.9 (a)):

- If $A < 1$, then Γ_A, Γ_B, and Γ_C are stacked one above the other, such that they are separated by unit distance from each other, and the left boundaries of Γ_A and Γ_C are aligned with each other and are placed one unit to the left of the left boundary of Γ_B; a is placed in the same vertical channel as o and $l(a)$, and in the same horizontal channel as $r(a)$.
- If $A \geq 1$, then first Γ_A is rotated clockwise by 90° and flipped right-to-left. Then, Γ_A, Γ_C, and Γ_B are placed from left-to-right in that order, separated by unit distances, such that the top boundaries of Γ_A and Γ_B are aligned with each other, and are one unit above the top boundary of Γ_C. Then, Γ_C is moved down until u^* becomes the lowest node of Γ; a is placed in the same vertical channel as $l(a)$ and in the same horizontal channel as o and $r(a)$.

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

In general Case 2, Γ is constructed by composing Γ_A, Γ_B, and Γ_C using a procedure similar to the one used in Case 1 (see Figure 8.9(b)).

Theorem 8.2 *Let T be a binary tree with n nodes. Given two numbers A and ϵ, where ϵ is a constant, such that $0 < \epsilon < 1$, and $n^{-\epsilon} \leq A \leq n^{\epsilon}$, a planar straight-line grid drawing of T with $O(n)$ area and aspect ratio A, can be constructed in $O(n \log n)$ time. Moreover, Γ has the subtree-separation property.*

Proof: Designate any leaf of T as its link node. Construct a drawing Γ of T by invoking Algorithm *Separation* with T, A, and ϵ as input. Γ will be a planar straight-line grid drawing contained entirely within a rectangle with $O(n)$ area and aspect ratio A, and which exhibits the subtree separation property. □

8.7.4 Ordered Trees

In this Section, we present two algorithms of [GR03a] in detail. The first algorithm (we call it *Fixed Spine*) shows that a binary tree admits an order-preserving *upward* planar straight-line grid drawing with *optimal* $O(n \log n)$ area. The second algorithm (we call it *Arbitrary Spine*), shows that a binary tree admits an order-preserving planar straight-line grid drawing with width $O(A + \log n)$, height $O((n/A) \log A)$, and area $O(n \log n)$, for any given $2 \leq A \leq n$. Setting $A = \log n$, it results in an area of $O(n \log \log n)$. Both algorithms take $O(n)$ time to construct the drawings.

Let T be an ordered tree. Each node of T has at most two children, called its *left* and *right* children, respectively.

Let α be a positive integer. An order-preserving planar straight-line grid drawing of T is an *α-drawing* of T, if it has the following two properties:

- **Property 1**: No node is placed to the left of, or above the root of T.
- **Property 2**: The vertical and horizontal separations between the root and its rightmost child are equal to α and one units, respectively.

A *left-corner* drawing of an ordered tree is an order-preserving planar straight-line grid drawing, where no node of the tree is placed to the left of, or above its root. Note that an α-drawing is also a left-corner drawing.

The *mirror-image* of T is the ordered tree obtained by reversing the counterclockwise order of edges incident on each node.

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 8.13):

- v_0 is the same as the root of T;
- v_{i+1} is a 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 *non-spine* node of T is one that does not belong to its spine.

Algorithm Fixed Spine

For simplicity, throughout this section, it is assumed that each non-leaf node has exactly two children. The algorithm can be simply extended to cover the case where a non-leaf node has only one child.

The *Fixed Spine* drawing algorithm uses a Path-based approach to obtain an order-preserving upward planar straight-line grid drawing with optimal ($O(n \log n)$) area of 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 α-drawing $D(T)$ of T, where α is a positive integer given as a parameter to the algorithm.

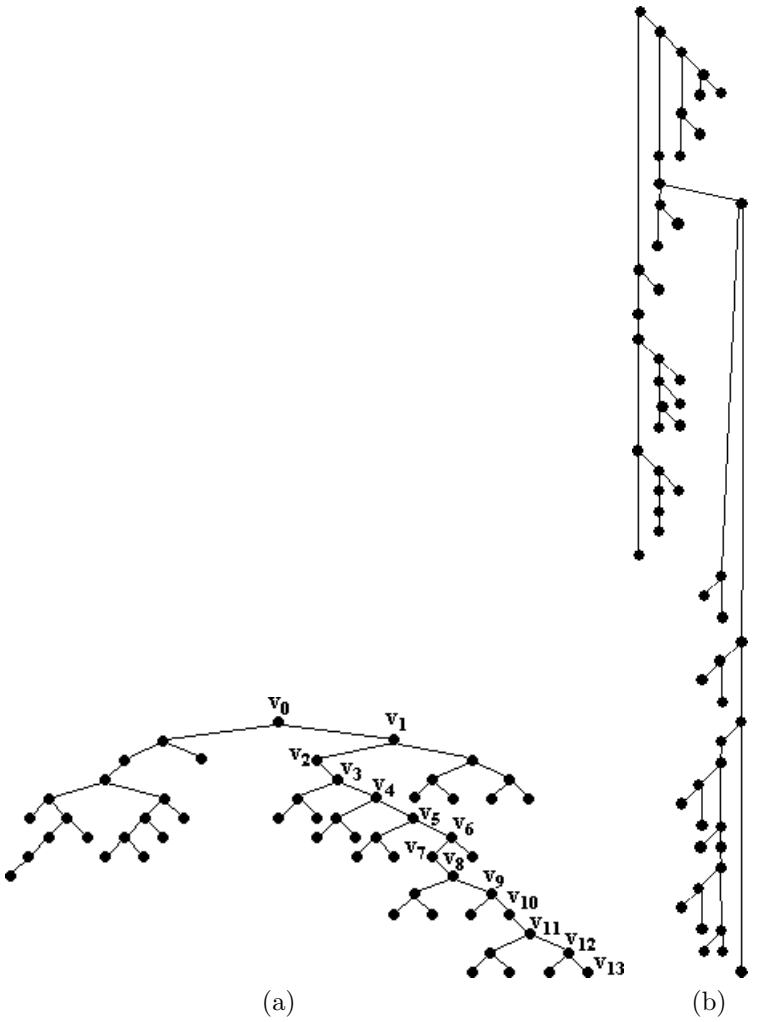


Figure 8.13 (a) A binary tree T with spine $v_0v_1 \dots v_{13}$. (b) The order-preserving planar upward straight-line grid drawing of T constructed by *Fixed Spine* algorithm.

Let $P = v_0v_1v_2 \dots v_m$ be a spine of T.
There are two cases (see Figures 8.14 and 8.15):

- *Case 1: v_1 is the left child of v_0* (see Figure 8.14(a)).
  Let L be the subtree rooted at v_1, s be the non-spine child of v_0, and R be the subtree rooted at s. v_0 is placed at the origin. 1-drawings $D(L)$ and $D(R)$ of L and R are recursively constructed. $D(R)$ is placed such that s is one unit to the right of, and α units below v_0. $D(L)$ is placed such that v_1 is in the same vertical channel as v_0, and is one unit below $D(R)$ (see Figure 8.14(b)).
- *Case 2: v_1 is the right child of v_0* (see Figure 8.15(a)).
  Let $k \geq 1$ be the smallest integer, such that v_k is either a leaf, or has a non-spine node as its left child.
  There are two subcases:

– *v_k has a non-spine node as its left child*: Let $s_0, s_1, \ldots, s_k$ be the non-spine children of $v_0, v_1, \ldots, v_k$, respectively. Let L, A, and B be the subtrees rooted at s_0, s_k, and v_{k+1}, respectively. Let $R_1, R_2, \ldots, R_{k-1}$ be the subtrees rooted at $s_1, s_2, \ldots, s_{k-1}$, respectively. T is drawn as shown in Figure 8.15(b). v_0 is placed at the origin. v_1 is placed one unit to the right of, and α units below v_0. 1-drawings $D(L), D(A), D(R_1), D(R_2), \ldots, D(R_{k-1})$ of $L, A, R_1, R_2, \ldots, R_{k-1}$, respectively, are recursively constructed. $D(R_1)$ is placed one unit to the right of, and one unit below v_1. For each i, where $2 \leq i \leq k-1$, v_i and $D(R_i)$ are placed such that v_i is in the same horizontal channel as the bottom of $D(R_{i-1})$ and is in the same vertical channel as v_{i-1}, and $D(R_i)$ is one unit to the right of, and one unit below v_i. v_k is placed in the same vertical channel as v_{k-1}, and in the same horizontal channel as the bottom of $D(R_{k-1})$. $D(A)$ is placed one unit below v_k, such that s_k is in the same vertical channel as v_k. $D(L)$ is placed one unit below $D(A)$, such that s_0 is in the same vertical channel as v_0. Let $\beta = h(D(A)) + h(D(L)) + 2$, where $h(D(A))$ and $h(D(L))$ denote the heights of $D(A)$ and $D(L)$, respectively. Let G be the drawing with the maximum width among $D(L), D(A), D(R_1), D(R_2), \ldots, D(R_{k-1})$. Let W be the width of G. A β-drawing of the mirror-image of B is recursively constructed, and then flipped right-to-left to obtain a drawing $D(B)$ of B. $D(B)$ is placed such that v_{k+1} is one unit below v_k, and $\max\{W+3$, width of $D(B)\}$ units to the right of v_0.

– *v_k is a leaf*: T is drawn in a similar fashion as in the previous subcase, except that $D(A)$ and $D(B)$ do not exist.

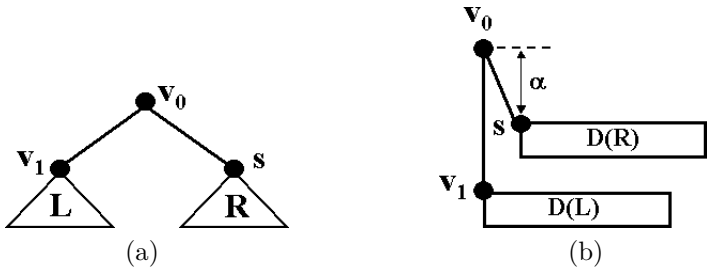


Figure 8.14 (a) The structure of a binary tree T in *Case 1*, where v_1 is the left child of v_0. (b) The drawing of T in *Case 1*. For simplicity, $D(L)$ and $D(R)$ are shown as identically sized boxes, but in actuality they may have different sizes.

Theorem 8.3 *An ordered 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.*

Proof: Let T be an n-node ordered binary tree. Using the above algorithm, construct a 1-drawing $D(T)$ of T in $O(n)$ time. As discussed above, $D(T)$ will be an order-preserving upward planar straight-line grid drawing of T with height at most n, width $O(\log n)$, and

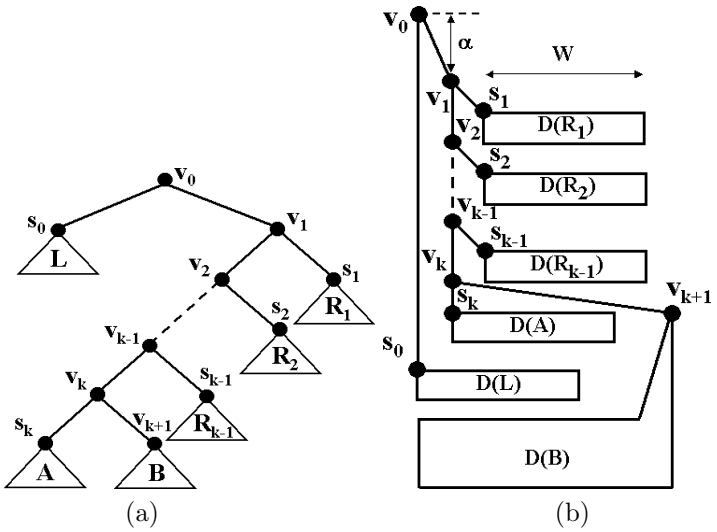


Figure 8.15 The structure of a binary tree T in *Case 2*, where v_1 is the right child of v_0: (a) v_k has a non-spine node as its left child; (b) the drawing of T, when v_k has a non-spine node as its left child. For simplicity, $D(A), D(L), D(R_1), \ldots, D(R_{k-1})$ are shown as identically sized boxes, but in actuality they may have different sizes.

optimal $O(n \log n)$ area. □

LEMMA 8.1 A left-corner drawing of an n-node ordered binary tree with area $O(n \log n)$, height $O(\log n)$, and width at most n, can be constructed in $O(n)$ time.

Proof: First a 1-drawing of the mirror image of T is constructed using Theorem 8.3, then it is rotated clockwise by 90°, and then it is flipped right-to-left. □

Algorithm Arbitrary Spine

For any user-defined number A, where $2 \leq A \leq n$, algorithm *Arbitrary Spine* uses a Path-based approach to 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 implies that, by setting $A = \log n$, such a drawing can be constructed with area $O(n \log \log n)$.

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.

Let n be the number of nodes in T. Let $2 \leq A \leq n$ be any number given as a parameter to the algorithm.

Figure 8.16 shows the drawing of the tree of Figure 8.13(a) constructed by algorithm *Arbitrary Spine* with $A = \sqrt{n}$, using Lemma 8.1.

In each recursive step, the algorithm 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 Lemma 8.1 such that the drawing has width at most m and height $O(\log m)$, where m is the number of

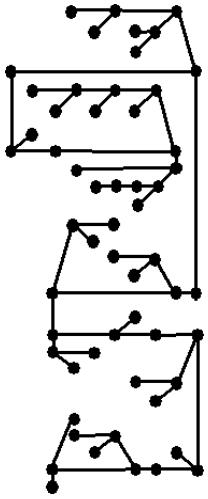


Figure 8.16 Drawing of the tree with $n = 57$ nodes of Figure 8.13(a) constructed by the Algorithm *Arbitrary Spine* with $A = \sqrt{n} = \sqrt{57} = 7.55$, using Lemma 8.1.

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 $P = v_0 v_1 v_2 \dots v_q$ be a spine of T'.
2. Let m_i denote the number of nodes in the subtree of T' rooted at v_i, where $0 \leq i \leq q$. Let v_k be the node of P with the value for k such that $m_k > m - A$ and $m_{k+1} \leq m - A$ (since T' has more than A nodes in it, and $m_0, m_1, \dots, m_q$ is a strictly decreasing sequence of numbers, such a k exists).
3. See Figures 8.17 and 8.18. Let T_i denote the subtree rooted at the non-spine child of v_i, where $0 \leq i \leq k-1$. Assume, for simplicity, that v_k and v_{k+1} are not leaves (the algorithm can be easily extended to handle the case, where v_k or v_{k+1} is a leaf). Let T^* and T^+ denote the subtrees rooted at the non-spine children of v_k and v_{k+1}, respectively. Let T'' denote the subtree rooted at v_{k+1}. Let T''' denote the subtree rooted at v_{k+2}.
4. Place v_0 at the origin.
5. There are 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 8.17. If s_0 is the left child of v_0, then D^* is placed one unit below v_0, with its left boundary aligned with v_0 (see Figure 8.17(a,c)). If s_0 is the right child of v_0, then D^* is placed one unit above, and one unit to the right of v_0 (see Figure 8.17(b,d)). Let W^*, W^+, and W''' be the widths of D^*, D^+, and D''', respectively. Place v_1 in the same horizontal channel as v_0 to its right at the distance $\max\{W^* + 2, W^+ + 2, W'''\}$ 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 D^+ is flipped right-to-left, and placed one

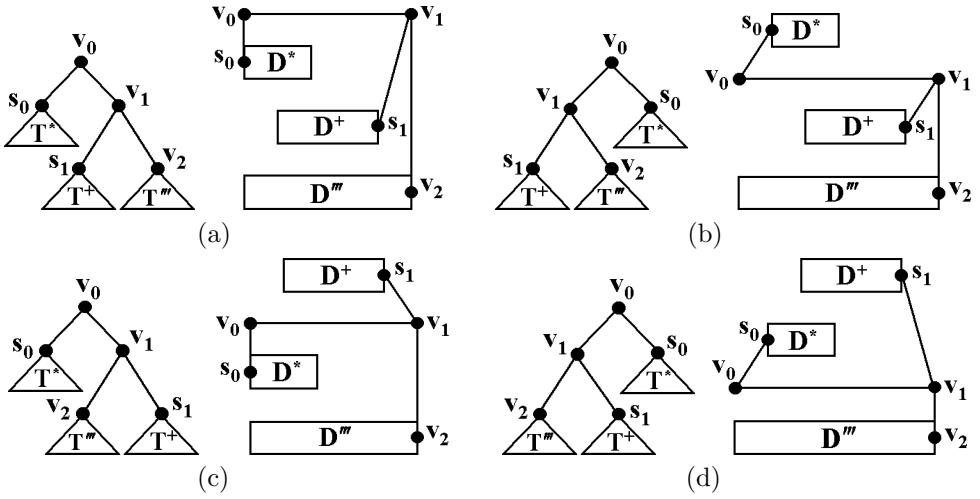


Figure 8.17 Case $k = 0$: (a) s_0 is the left child of v_0, and s_1 is the left child of v_1; (b) s_0 is the right child of v_0, and s_1 is the left child of v_1; (c) s_0 is the left child of v_0, and s_1 is the right child of v_1; (d) s_0 is the right child of v_0, and s_1 is the right child of v_1.

unit below B_0, and one unit to the left of v_1 (see Figure 8.17(a,b)). If s_1 is the right child of v_1, then D^+ is flipped right-to-left, and placed one unit above C_0, and one unit to the left of v_1 (see Figure 8.17(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''' right-to-left, and place it one unit below B_1, such that its right boundary is aligned with v_1 (see Figure 8.17).

- $k > 0$: For each T_i, where $0 \leq i \leq k-1$, construct a left-corner drawing D_i of T_i using Lemma 8.1.

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

  T' is drawn as shown in Figure 8.18. If T_0 is rooted at the left child of v_0, then D_0 is placed one unit below v_0, with its 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. Place each D_i and v_i, where $1 \leq i \leq k-1$, 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 width 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.

  Place v_k to the right of and in the same horizontal channel as v_{k-1}, such that the horizontal distance between v_k and v_0 is equal to $\max\{d+1, W^*+2, W''\}$.

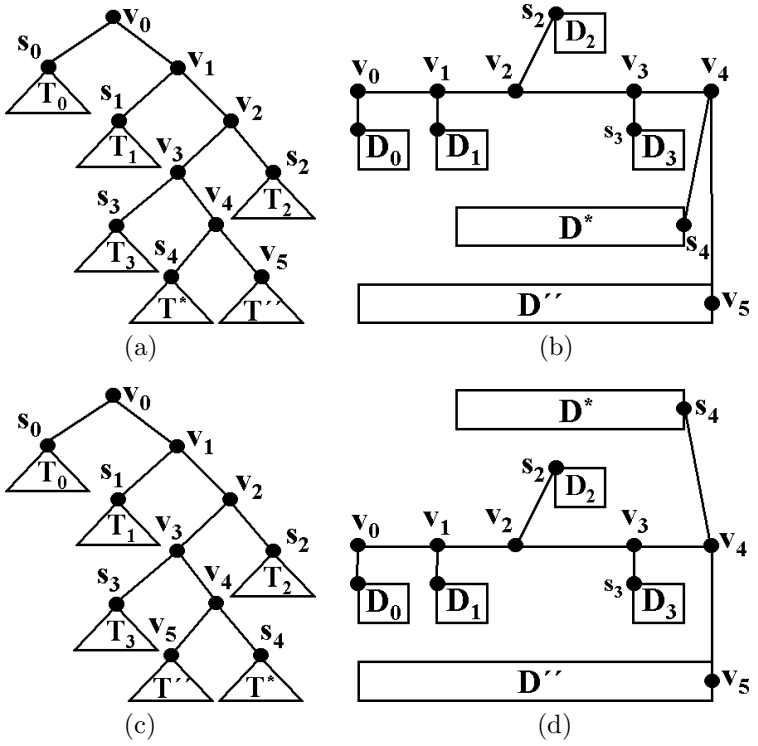


Figure 8.18 Case $k > 0$: Here $k = 4$, s_0, s_1, and s_3 are the left children of v_0, v_1, and v_3 respectively, s_2 is the right child of v_2, T_0, T_1, T_2, T_3, and T'' are the subtrees rooted at v_0, v_1, v_2, v_3, and v_5, respectively, s_4 is the non-spine child of v_4, and T^* is the subtree rooted at s_4; (a) s_4 is the left child of v_4; (c) s_4 is the right child of v_4. For simplicity, D_0, D_1, D_2, D_3 are shown as identically sized boxes, but in actuality they may have different sizes.

> If T^* is rooted at the left-child of v_k, then D^* is flipped right-to-left, and placed one unit below B_{k-1}, and one unit left of v_k (see Figure 8.18(b)). If T^* is rooted at the right-child of v_k, then D^* is flipped right-to-left, and placed one unit above C_{k-1}, and one unit to the left of v_k (see Figure 8.18(d)). Let B_k be the lowest horizontal channel occupied by the subdrawing consisting of $v_1, v_2, \ldots, v_k$, and $D_0, D_1, D_2, \ldots, D_{k-1}, D^*$. Flip D'' right-to-left, and place it one unit below B_k, such that its right boundary is aligned with v_k (see Figure 8.18(b,d)).

Theorem 8.4 *Let T be an ordered 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 $A = \log n$, it is obtained that:

COROLLARY 8.1 An n-node ordered 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.

8.8 Algorithms for 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 than a binary tree. The *degree* of a tree is equal to the maximum number of edges incident on a node.

8.8.1 Theoretical Results

We summarize known theoretical results on planar grid drawings of general trees. Chan [Cha02] has shown an upper bound of $O(n^{1+\epsilon})$, where $\epsilon > 0$ is any user-defined constant, on the area of an order-preserving planar upward straight-line grid drawing of a general tree. Garg et al. [GGT96] has given an upper bound of $O(n \log n)$ on order-preserving planar upward polyline grid drawings. As for the lower bound on the area-requirement of order-preserving drawings, Garg et. al [GGT96] 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. Garg et al. [GGT96] 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 $\max\{0, 2\beta - 1\} < \gamma < 1$. Garg and Rusu [GR03c] show that any tree with degree $O(n^{\delta})$, where $0 \leq \delta < 1/2$ is any constant, 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.

Table 8.3 summarizes these results.

Tree Type	Drawing Type	Area	Aspect Ratio	Ref.
Tree with degree $O(n^{\delta})$, for any constant $0 \leq \delta < 1/2$	Non-upward Straight-line Non-order-preserving	$\Theta(n)$	$[1, n^{\alpha}]$	[GR03c]
Tree with degree $O(n^{\beta})$, for any constant $0 \leq \beta < 1$	Upward Polyline Non-order-preserving	$\Theta(n)$	$[1, n^{\gamma}]$	[GGT96]
General	Upward Polyline Order-Preserving	$\Theta(n \log n)$	$n/\log n$	[GGT96]
	Upward Straight-line Order-Preserving	$O(n^{1+\epsilon})$	n	[Cha02]
	Non-upward Straight-line Order-preserving	$O(n^{1+\epsilon})$	n	[Cha02]
		$O(n \log n)$	$n/\log n$	[GR03a]

Table 8.3 Bounds on the areas and aspect ratios of various kinds of planar straight-line grid drawings of an n-node tree. Here, α, γ, and ϵ are arbitrary user-defined constants, such that $0 \leq \alpha < 1$, $0 \leq \gamma < 1$, and $0 < \epsilon < 1$.

A variety of results are available for other kinds of drawings. Di Battista et al. [DETT99]

and Frati [Fra09] have given a survey of these results.

8.8.2 Unordered Trees

In this section, we briefly sketch a bottom-up algorithm developed using the ringed circular layout approach [TM02]. This algorithm (called *Rings*) is space-efficient for high-degree trees, however, the resulting drawing is straight-line but not planar.

The subtrees rooted at the children of the root of the tree are drawn recursively as circles placed in concentric rings around the center of the circle to ensure efficient use of space. The children of the root are divided into multiple categories according to their size. One ring is assigned to each category, so the outer rings consist of the largest trees, while the inner rings consist of the smallest ones (see Figure 8.19). In this way, a tree containing more information is allocated more space, thus showing more distinguishable edges and allowing more structural information to be shown in context.

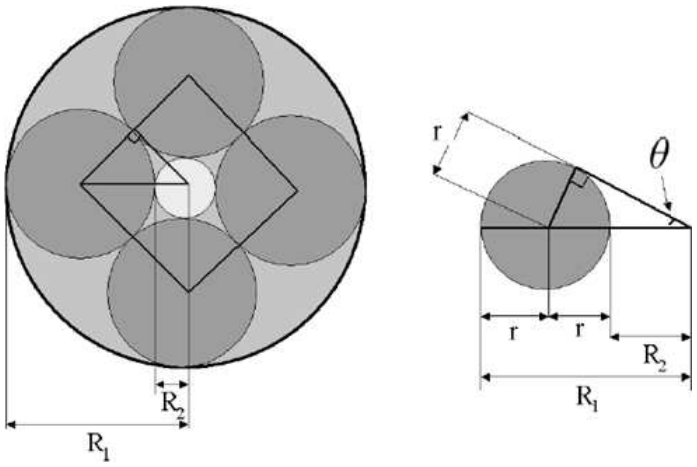


Figure 8.19 Layout of the ringed circular layout algorithm of [TM02]. The four larger rings represent the largest children of the parent node, and the inner ring represents the area left for the rest of the children.

The relationship below can be established between the number of children circles in the outermost ring and the percentage of area taken up by the ring.

$$f(n) = \frac{(R_2)^2}{(R_1)^2} = \frac{(1-\sin(\theta))^2}{(1+\sin(\theta))^2} = \frac{(1-\sin(\frac{\pi}{n}))^2}{(1+\sin(\frac{\pi}{n}))^2} \tag{8.1}$$

here, $f(n)$ is the fraction of the area left after n circles have been placed in the ring. The basic steps of the algorithm are presented below:

Algorithm *Rings*

Sort the children by their number of children;

Find the smallest k for which the sum of the number of children of the first k children expressed as a fraction of the total number of grandchildren is greater or equal to $f(k)$;

Place first k children in the outermost ring;

Place the rest of the children in the same way in the inner rings;

end Algorithm.

Visual cues like color and transparency are also used to enhance structural information, as well as to highlight specific information (such as information importance or relevance). Adjacent concentric rings are rotated in opposite directions to decrease the occlusion of a particular branch (see Figure 8.20.

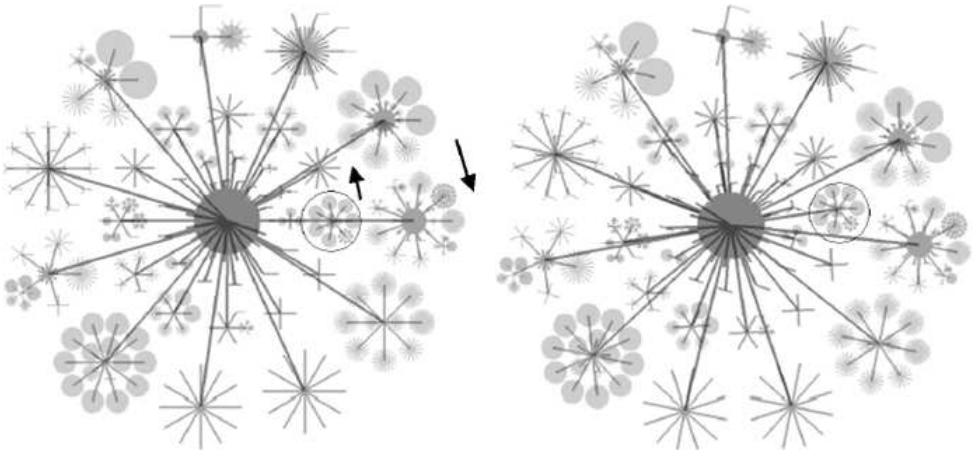


Figure 8.20 Rotation strategy to decrease occlusion.

A binary tree adaptation of the *Rings* algorithm [RS08] places the children of a node in either the same vertical or horizontal channel, starting with the same horizontal channel at the root (depth 0), and alternates between vertical and horizontal channel placement for every following depth in the tree. In addition, the length of the edge connecting a subtree to its parent is set to $depth(subtree(v)) + 1$, where $depth(subtree(v))$ is the depth of the subtree rooted at node v. This ensures that enough space is made available to draw the rest of the subtree, which is consistent with other rings-based algorithms. A drawing produced by the binary tree adaptation of the *Rings* algorithm is provided in Figure 8.21.

In order to allow for real-time interaction, a top-down variation of the *Rings* algorithm, called *FastRings* [RSJ07], trades space for time. In *FastRings*, all nodes of the tree are considered to be equivalent and assigned same size circles. This allows the algorithm to start drawing the tree much sooner, when only the first level of children is available. The drawing can be refined later by filling up the circles from the first level once new information becomes available. Experiments show that *FastRings* increases the speed of constructing entire drawings by 51%, and is twelve times faster in producing first drawings.

8.8.3 Ordered Trees

In this section, we briefly sketch an algorithm for constructing a (non-upward) order-preserving planar straight-line grid drawing of a general ordered tree with n nodes with $O(n \log n)$ area in $O(n)$ time [GR03a]. This algorithm uses a Path-based approach.

Let T be an ordered tree with n nodes. In each recursive step, the algorithm breaks T into several subtrees, draws each subtree recursively, and then combines their drawings to obtain an α-drawing $D(T)$ of T, where α is a positive integer given as a parameter to the

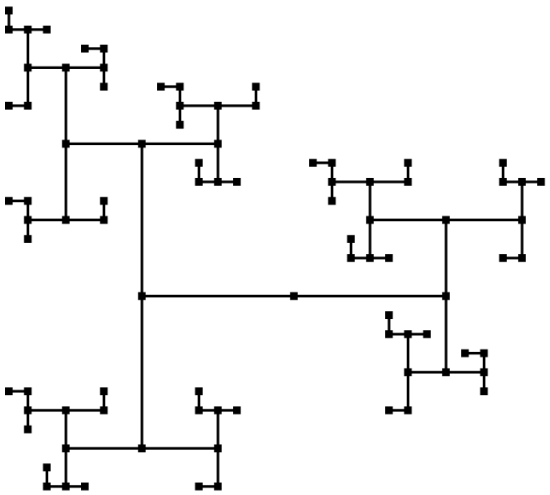


Figure 8.21 Drawing of the Fibonacci tree with 88 nodes, generated by the binary tree adaptation of the *Rings* algorithm.

algorithm.

Let $P = v_0 v_1 v_2 \dots v_m$ be a spine of T (see Section 8.7.4 for the definition of spine). The general structure of T is shown in Figure 8.22(a). Let $s_0, s_1, \dots, s_i, v_1, s_{i+1}, s_{i+2}, \dots, s_p$ be the left-to-right order of the children of v_0, where the list $s_0, s_1, \dots, s_i$ is empty if v_1 is the leftmost child of v_0, and the list $s_{i+1}, s_{i+2}, \dots, s_p$ is empty if v_1 is the rightmost child of v_0. Let A_k denote the subtree rooted at the node s_k, where $0 \leq k \leq p$. Let $t_0, t_1, \dots, t_j, v_2, t_{j+1}, t_{j+2}, \dots, t_r$ be the left-to-right order of the children of v_1, where the list $t_0, t_1, \dots, t_j$ is empty if v_2 is the leftmost child of v_1, and the list $t_{j+1}, t_{j+2}, \dots, t_r$ is empty if v_2 is the rightmost child of v_1. Let B_k denote the subtree rooted at the node t_k, where $0 \leq k \leq r$. Let C denote the subtree rooted at v_2.

T is drawn as follows (see Figure 8.22(b)):

1. Recursively construct 1-drawings $D(A_0), \dots, D(A_p)$ of $A_0, \dots, A_p$, respectively, and $D(B_0), \dots, D(B_r)$ of $B_0, \dots, B_r$, respectively.
2. Place v_0 at the origin.
3. Place $D(A_{i+1}), \dots, D(A_p)$ one above the other at unit vertical separations from each other, such that $D(A_p)$ is at the top, $D(A_{i+1})$ is at the bottom, $s_{i+1}, \dots, s_p$ are in the same vertical channel, and s_p is α units below, and one unit to the right of v_0.
4. Place $D(B_{j+1}), \dots, D(B_r)$ one above the other at unit vertical separations from each other, such that $D(B_r)$ is at the top, $D(B_{j+1})$ is at the bottom, $t_{j+1}, \dots, t_r$ are in the same vertical channel, and t_r is one unit below $D(A_{i+1})$, and one unit to the right of s_{i+1}.
5. Place v_1 in the same horizontal channel as the bottom of $D(B_{j+1})$, and one unit to the right of v_0.
6. Place $D(B_0), \dots, D(B_j)$ one above the other at unit vertical separations from each other, such that $D(B_j)$ is at the top, $D(B_0)$ is at the bottom, $t_0, \dots, t_j$ are in the same vertical channel, and t_j is one unit below, and one unit to the right of v_1.

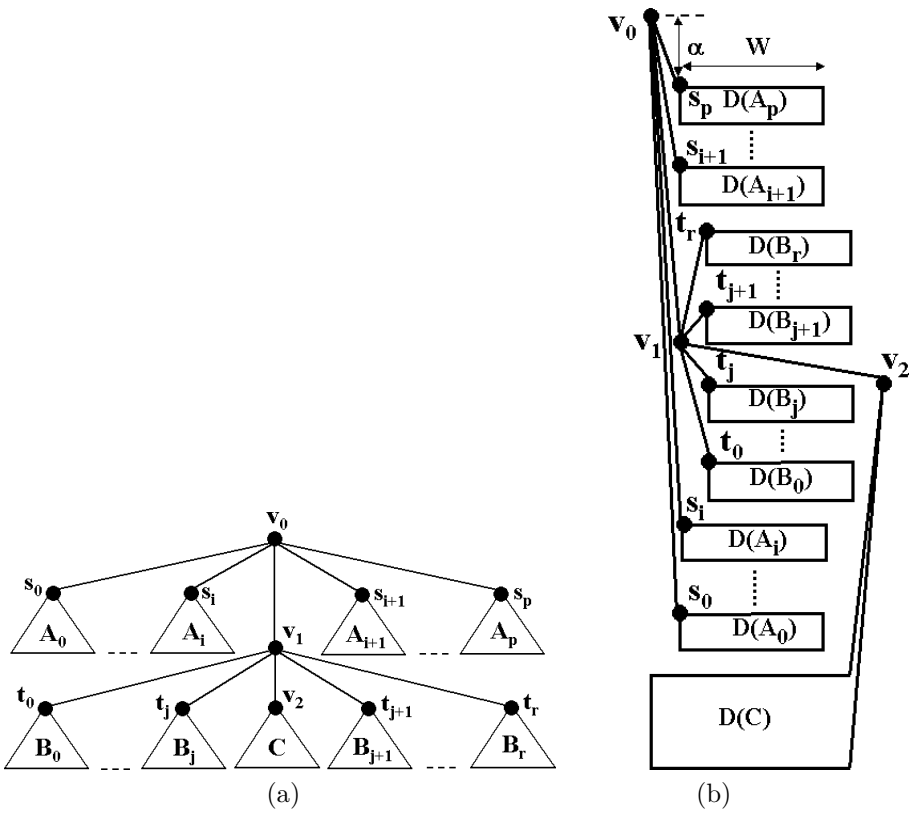


Figure 8.22 (a) The structure of a general tree T. (b) The drawing of T constructed by the algorithm of Section 8.8.3. For simplicity, $D(A_0), \ldots, D(A_p), D(B_0), \ldots, D(B_r)$ are shown as identically sized boxes, but in actuality they may have different sizes.

7. Place $D(A_0), \ldots, D(A_i)$ one above the other at unit vertical separations from each other, such that $D(A_i)$ is at the top, $D(A_0)$ is at the bottom, $s_0, \ldots, s_i$ are in the same vertical channel, and s_i is one unit below $D(B_0)$, and in the same vertical channel as v_1.
8. Let $\beta = h(D(B_0)) + \ldots + h(D(B_j)) + h(D(A_0)) + \ldots + h(D(A_i)) + i + j + 2$, where $h(D(B_0)), \ldots, h(D(B_j)), h(D(A_0)), \ldots, h(D(A_i))$ denote the heights of $D(B_0), \ldots, D(B_j), D(A_0), \ldots, D(A_i)$, respectively. Recursively construct a β-drawing of the mirror-image of C, and flip it right-to-left to obtain a drawing $D(C)$ of C. Let G be the drawing with the maximum width among $D(A_0), \ldots, D(A_p), D(B_0), \ldots, D(B_r)$. Let W be the width of G. Place $D(C)$ such that v_2 is one unit below v_1, and $\max\{W + 3$, width of $D(C)\}$ units to the right of v_0.

Theorem 8.5 *An ordered tree with n nodes admits a (non-upward) order-preserving planar straight-line grid drawing with $O(n \log n)$ area, $O(\log n)$ width, and height at most n, which can be constructed in $O(n)$ time.*

Proof: Let T be an n-node ordered tree. Using the above algorithm, construct a 1-drawing $D(T)$ of T in $O(n)$ time. As discussed above, $D(T)$ will be an order-preserving planar straight-line grid drawing of T with height at most n, width $O(\log n)$, and $O(n \log n)$

area. □

LEMMA 8.2 A left-corner drawing (see Section 8.7.4 for the definition of a left-corner drawing) of an n-node ordered tree with area $O(n \log n)$, height $O(\log n)$, and width at most n, can be constructed in $O(n)$ time.

Proof: First a 1-drawing of the mirror-image of T is constructed using Theorem 8.5, then it is rotated clockwise by 90°, and then it is flipped right-to-left. □

8.9 Conclusion

Drawing trees is one of the best studied areas in graph drawing, initiated over forty years ago [Knu68]. Any tree accepts a planar drawing, hence most tree drawing algorithms achieve this aesthetic. Several tree drawing strategies exist that allow one to create drawings with small area, user-controlled aspect ratio, relatively high angular resolution, a small number of bends, and in efficient time.

We conclude by briefly introducing several algorithms and techniques which do not necessarily fit the general approaches described in this Chapter.

Hyperbolic tree [LRP95] (see Figure 8.23) simulates the distortion effect of fisheye lens (enlarge the focus and shrink the rest).

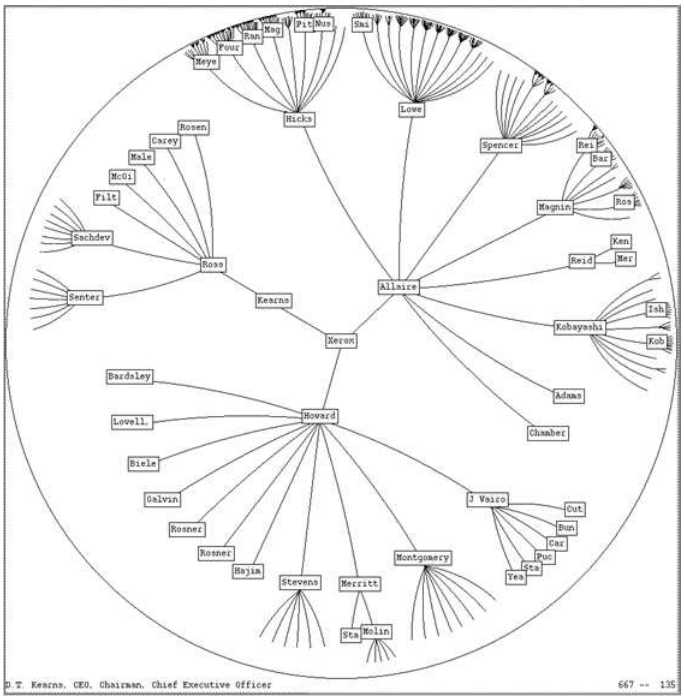


Figure 8.23 Screenshot of Hyperbolic tree.

Pad++ [BHP+96] (See Figure 8.24) displays the nodes as thumbnails of pages of infor-

mation. It institutes a focus+context style by enlarging the focus node and allowing other nodes to be in view.

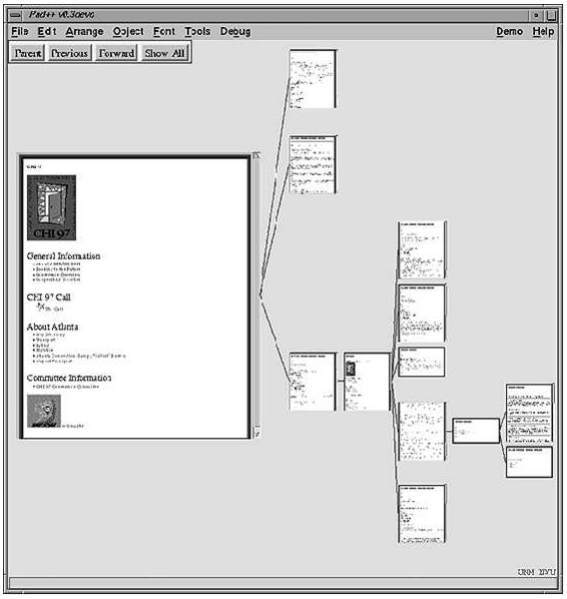


Figure 8.24 Screenshot of Pad++.

Botanical tree [KvdWW01] (See Figure 8.25) is based on the observation that people can easily see the branches, leaves and their arrangement in a botanical tree, despite of the large number of elements. Non-leaf nodes are mapped to branches and child nodes to sub-branches. Continuing branches are emphasized, long branches are contracted, and sets of leaves are shown as fruit.

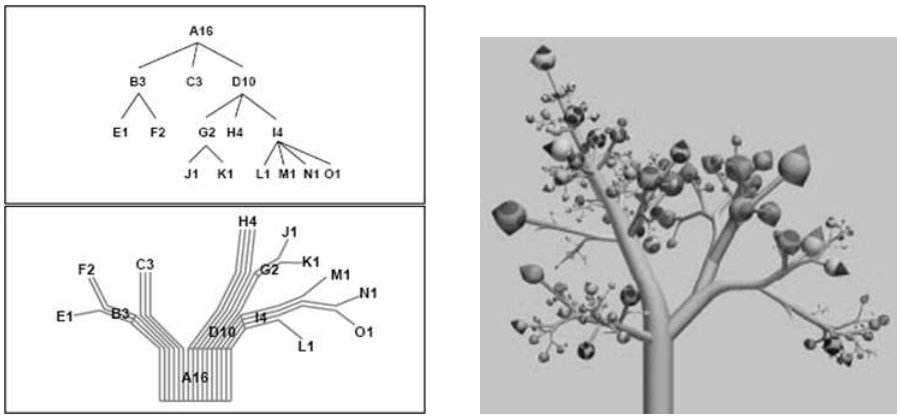
Space tree [PGB02] (See Figure 8.26) allows dynamic rescaling of branches of the tree to best fit the available screen space. Branches that do not fit on the screen are summarized by a triangular preview.

Quad [RYC08] (See Figure 8.27) allows the user to specify a preferred angular resolution, and then employs a best-effort-delivery to generate a planar straight-line drawing in which all angles between edges are above the specified angular coefficient. When a node has too many children, resulting in an impossibility of achieving angles above the specified angular coefficient, the algorithm distributes all remaining children evenly among the three quads of the Cartesian plane.

Adaptive tree drawing [RCJ06] is a system which first analyzes the input tree to classify it as a specific type and then selects an algorithm to draw it with respect to user-specified quality measures. The algorithm that is selected to draw a given tree is based upon an experimental comparison [RJSC06], which orders the performance of the algorithms for each quality measure.

Hexagonal tree drawing [BBB$^+$09] (See Figure 8.28) allows drawings of degree-6 trees on the hexagonal grid, which consists of equilateral triangles.

Most of the tree drawing algorithms draw trees on unbounded planes, and few of them draw trees on regions which are bounded by rectangles. However, certain applications, such as a graphics software by which one would like to draw a tree inside a star-shaped polygon,



(a) (b)

Figure 8.25 (a) Node and link diagram (top) and corresponding strands model (bottom). (b) Screenshot of Botanical tree.

require trees to be drawn on regions which are bounded by general polygons [BR04] (See Figure 8.29).

A *layered drawing* of a tree T is a planar straight-line drawing of T such that the vertices are drawn on a set of layers. Some applications such as phylogenetic evolutions and programming language parsing benefit from layered upward drawings of trees. Alam et al. [ASRR08] (See Figure 8.30) provide algorithms for minimum-layer upward drawings of both ordered and unordered trees.

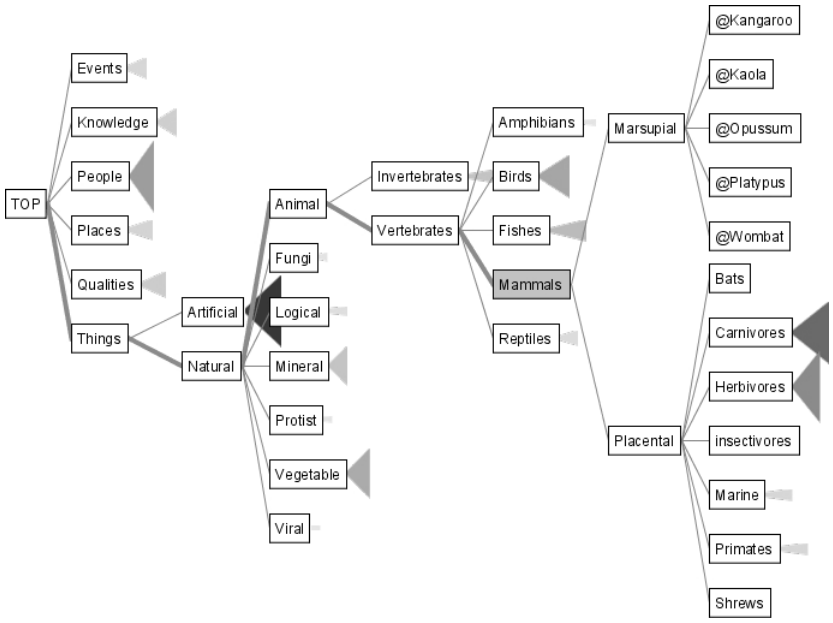


Figure 8.26 Screenshot of Space tree.

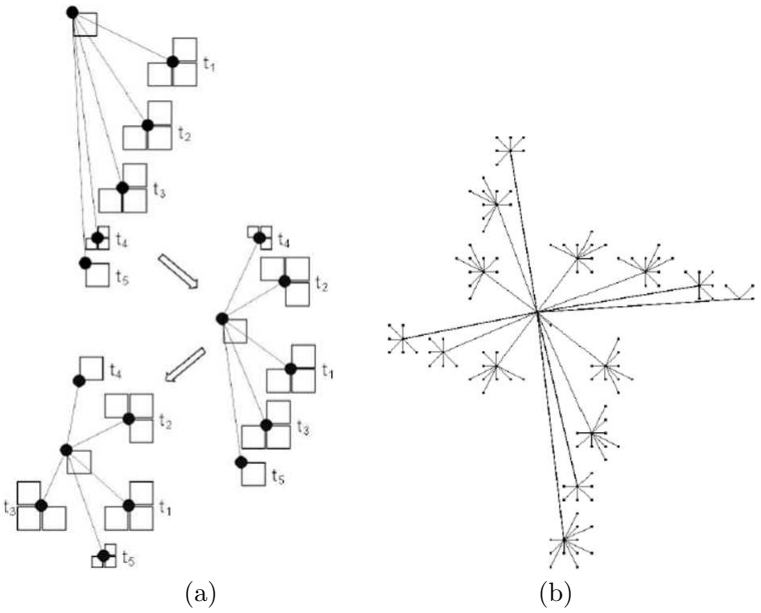


Figure 8.27 (a) Subtrees are distributed into three quads of the Cartesian plane when the angular coefficient cannot be met by using only one or two quads. (b) Screenshot of a drawing generated using Quad algorithm, with user-specified angular resolution of 45°.

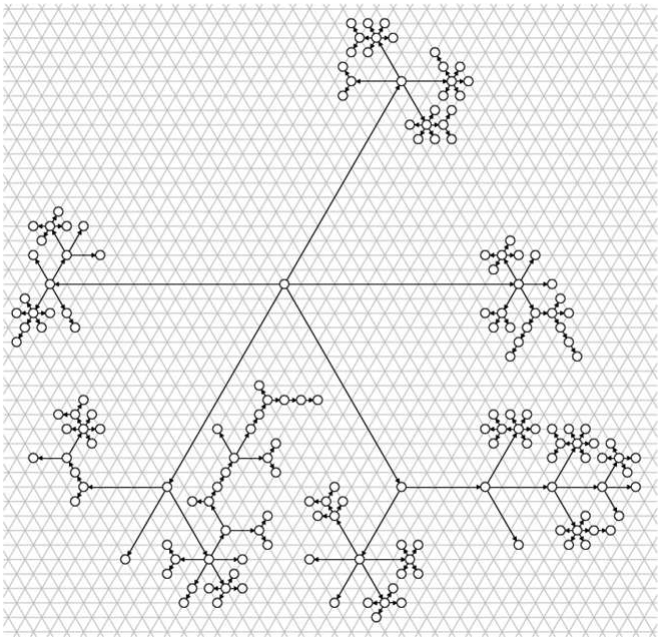


Figure 8.28 A planar straight-line drawing of a tree with outdegree five on the hexagonal grid.

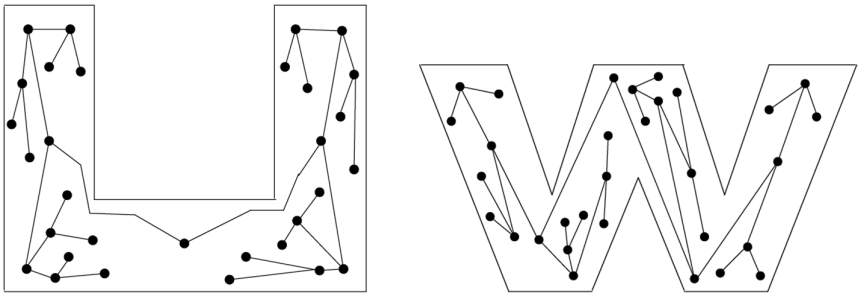


Figure 8.29 (a) Drawing of a 31-node complete binary tree inside a U-shaped rectiliniar polygon. (b) Drawing of a 31-node complete binary tree inside a W-shaped polygon.

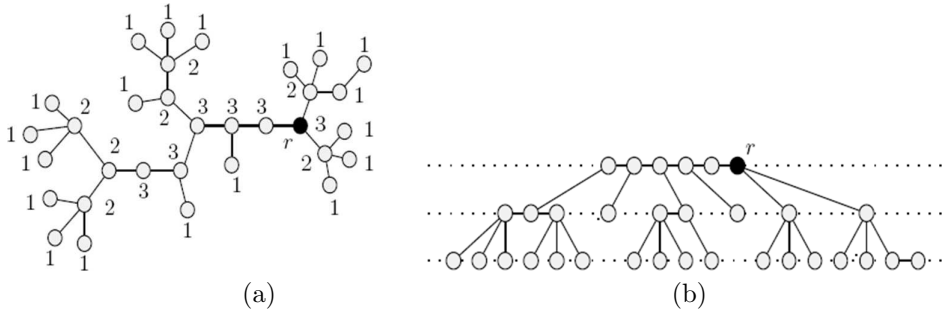


Figure 8.30 (a) A tree with root r and layer-labelings. (b) A minimum-layer upward drawing of the tree in (a).

Bibliography

[ASRR08] M.J. Alam, M.A.H. Samee, M.M. Rabbi, and M.S. Rahman. Upward drawing of trees on the minimum number of layers. In *Proceedings of the 2nd Workshop on Algortihms and Computation*, volume 4921 of *Lecture Notes in Computer Science*, pages 88–99, 2008.

[Bac07] C. Bachmaier. A radial adaptation of the Sugiyama framework for visualizing hierarchical information. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):583–594, 2007.

[BBB+09] C. Bachmaier, F.J. Brandenburg, W. Brunner, A. Hofmeier, M. Matzeder, and T. Unfried. Tree drawings on the hexagonal grid. In *Proceedings 16th International Symposium on Graph Drawing*, pages 372–383. Springer-Verlag, 2009.

[Ber81] M. A. Bernard. On the automated drawing of graphs. In *Proc. 3rd Caribbean Conf. on Combinatorics and Computing*, pages 43–55, 1981.

[BHP+96] B.B. Bederson, J.D. Hollan, K. Perlin, J. Meyer, D. Bacon, and G. Furnas. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. *Journal of Visual Languages and Computing*, 7:3–31, 1996.

[BJL02] C. Buchheim, M. Jünger, and S. Leipert. Improving Walker's algorithm to run in linear time. In Michael T. Goodrich and Stephen G. Kobourov, editors, *Graph Drawing (Proceedings of* 10^{th} *International Sympsium on Graph Drawing, 2002)*, volume 2528 of *Lecture Notes in Computer Science*, pages 344–353. Springer, 2002.

[Blo93] A. Bloesch. Aesthetic layout of generalized trees. *Software Practice and Experience*, 23(8):817–827, 1993.

[BM03] M. Bernard and S. Mohammed. Labeled radial drawing of data structures. In *Proceedings 7th International Conference on Information Visualisation*, pages 479–555. IEEE Computers Society, 2003.

[BR04] A. Bagheri and M. Razzazi. How to draw free trees inside bounded rectilinear polygons. *International Journal of Computer Mathematics*, 81(11):1329–1339, 2004.

[CC99] E. A. Chi and S. K. Card. Sensemaking of evolving web sites using visualization spreadsheets. In *Proceedings of the Symposium on Information Visualization (InfoViz '99)*, volume 142, pages 18–25. IEEE Press, 1999.

[CDP92] 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.

[CGKT97] T. M. Chan, M. T. Goodrich, S. R. Kosaraju, and R. Tamassia. Optimizing area and aspect ratio in straight-line orthogonal tree drawings. In S. North, editor, *Graph Drawing (Proc. GD '96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 63–75. Springer-Verlag, 1997.

[CGKT02] 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.

[Cha02] T. M. Chan. A near-linear area bound for drawing binary trees. *Algorithmica*, 34(1):1–13, 2002.

0-8493-8597-0/01/$0.00+$1.50
© 2004 by CRC Press, LLC

[CP98] P. Crescenzi and P. Penna. Strictly-upward drawings of ordered search trees. *Theoretical Computer Science*, 203(1):51–67, 1998.

[CPM+98] E. H. Chi, J. Pitkow, J. Mackinlay, P. Pirolli, and R. Gossweiler. Visualizing the evolution of Web ecologies. In *Proceedings of the Human Factors in Computing Systems*, pages 400–407, 1998.

[CPP98] 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).

[CPP00] E.H. Chi, P. Pirolli, and J. Pitkow. The Scent of a Site: A system for analyzing and predicting information scent, usage, and usability of a Web site. In *Proceedings of the Human Factors in Computing Systems*, pages 161–168, 2000.

[DETT94] 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.

[DETT99] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.

[DT81] D. Dolev and H.W. Trickey. On linear area embedding of planar graphs. Technical report, Stanford University, Stanford, USA, 1981.

[Ead92] P. D. Eades. Drawing free trees. *Bulletin of the Institute for Combinatorics and its Applications*, 5:10–36, 1992.

[Fra07] F. Frati. Straight-line orthogonal drawings of binary and ternary trees. In Seok-Hee Hong and Takao Nishizeki, editors, *15th International Symposium on Graph Drawing*, volume 4875 of *Lecture Notes in Computer Science*, pages 76–87, 2007.

[Fra09] F. Frati. *Small Screens and Large Graphs: Area-Efficient Drawings of Planar Combinatorial Structures*. PhD thesis, Computer Science and Engineering, Roma Tre University, 2009.

[GADM04] S. Grivet, D. Auber, J.-P. Domenger, and G. Melancon. Bubble tree drawing algorithm. In *International Conference on Computer Vision and Graphics*, pages 633–641. Springer Verlag, 2004.

[GGT96] A. Garg, M. T. Goodrich, and R. Tamassia. Planar upward tree drawings with optimal area. *Internat. J. Comput. Geom. Appl.*, 6:333–356, 1996.

[GR02] A. Garg and A. Rusu. Straight-line drawings of binary trees with linear area and arbitrary aspect ratio. In Michael T. Goodrich and Stephen G. Kobourov, editors, *Graph Drawing (Proceedings of* 10^{th} *International Sympsium on Graph Drawing, 2002)*, volume 2528 of *Lecture Notes in Computer Science*, pages 320–331. Springer, 2002.

[GR03a] A. Garg and A. Rusu. Area-efficient order-preserving planar straight-line drawings of ordered trees. *International Journal of Computational Geometry and Applications*, 13(6):487–505, 2003.

[GR03b] A. Garg and A. Rusu. A more practical algorithm for drawing binary trees in linear area with arbitrary aspect ratio. In Giuseppe Liotta, editor, *Graph Drawing (Proceedings of* 11^{th} *International Sympsium on Graph Drawing, 2003)*, volume 2912 of *Lecture Notes in Computer Science*, pages 159–165. Springer, 2003.

[GR03c] A. Garg and A. Rusu. Straight-line drawings of general trees with linear area and arbitrary aspect ratio. In *Proceedings 2003 International Conference on Computational Science and Its Applications*, volume 2669 of *Lecture Notes in Computer Science*, pages 876–885. Springer, 2003.

[GR04] A. Garg and A. Rusu. Straight-line drawings of binary trees with linear area and arbitrary aspect ratio. *Journal of Graph Algorithms and Applications*, 8(2):135–160, 2004.

[Kim95] Sung Kwon Kim. Simple algorithms for orthogonal upward drawings of binary and ternary trees. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 115–120, 1995.

[Kim04] S.K. Kim. Order-preserving, upward drawing of binary trees using fewer bends. *Discrete Applied Mathematics Journal*, 143(1–3):318–323, 2004.

[Knu68] D. E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1st edition, 1968.

[KvdWW01] E. Kleiberg, H. van de Wetering, and J.J. Van Wijk. Botanical visualization of huge hierarchies. In *Proceedings of the IEEE Symposium on Information Visualization*, 2001.

[Lei80] C. E. Leiserson. Area-efficient graph layouts (for VLSI). In *Proc. 21st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 270–281, 1980.

[LRP95] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proc. ACM Conf. on Human Factors in Computing Systems (CHI)*, 1995.

[Mac03] B. MacLennan. Molecular combinatory computing for nanostructure synthesis and control. In *Proceedings 3rd IEEE Conference on Nanotechnology*, volume 2 of *IEEE Press*, pages 179–182, 2003.

[MH98] G. Melacon and I. Herman. Circular drawing of rooted trees. Technical Report INS-9817, Netherland National Research Institute for Mathematics and Computer Sciences, 1998.

[MMC99] G. Melacon, J.D. Mackinlay, and S. K. Card. Cone trees: animated 3D visualization of hierarchical information. In *Human Factors in Computing Systems, CHI'99 Conference Proceedings*, pages 189–194. ACM Press, 1999.

[PCJ97] 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.

[PGB02] C. Plaisant, J. Grosjean, and B.B. Bederson. Spacetree: supporting exploration in large node link tree, design evolution and empirical evaluation. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 57–64, 2002.

[Pur97] 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.

[Pur00] Helen C. Purchase. Effective information visualisation: A study of graph drawing aesthetics and algorithms. *Interact. Comput.*, 13(2):147–162, 2000.

[RCJ06] A. Rusu, C. Clement, and R. Jianu. Adaptive binary trees visualization with respect to user-specified quality measures. In *Proceedings 10th International Conference on Information Visualisation*, pages 469–474. IEEE Computers Society, 2006.

[RJSC06] A. Rusu, R. Jianu, C. Santiago, and C. Clement. An experimental study on algorithms for drawing binary trees. In *Proceedings 5th Asia Pacific Symposium on Information Visualization*, volume 60 of *Conference in Research and Practice in Information Technology*, pages 85–88. Australian Computer Society Inc., 2006.

[RMC91] G. G. Robertson, J. D. Mackinlay, and S. K. Card. Cone trees: Animated 3D visualizations of hierarchical information. In *Proc. ACM Conf. on Human*

Factors in Computing Systems, pages 189–193, 1991.

[RS07] A. Rusu and C. Santiago. A practical algorithm for planar straight-line grid drawings of general trees with linear area and arbitrary aspect ratio. In *Proceedings 11th International Conference on Information Visualisation*, pages 743–750. IEEE Computers Society, 2007.

[RS08] A. Rusu and C. Santiago. Grid drawings of binary trees: An experimental study. *Journal of Graph Algorithms and Applications*, 12(2):131–195, 2008.

[RSJ07] A. Rusu, C. Santiago, and R. Jianu. Real-time interactive visualization of information hierarchies. In *Proceedings 11th International Conference on Information Visualisation*, pages 117–123. IEEE Computers Society, 2007.

[RT81] E. Reingold and J. Tilford. Tidier drawing of trees. *IEEE Trans. Softw. Eng.*, SE-7(2):223–228, 1981.

[RYC08] A. Rusu, C. Yao, and A. Crowell. A planar straight-line grid drawing algorithm for high degree general trees with user-specified angular coefficient. In *Proceedings 12th International Conference on Information Visualisation*, pages 600–609. IEEE Computers Society, 2008.

[SB94] M. Sarkar and M. H. Brown. Graphical fisheye views. *Commun. ACM*, 37(12):73–84, 1994.

[SKC00] C.-S. Shin, S. K. Kim, and K.-Y. Chwa. Area-efficient algorithms for straight-line tree drawings. *Comput. Geom. Theory Appl.*, 15:175–202, 2000.

[TDB88] 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.

[TM02] S. T. Teoh and K. L. Ma. Rings: A technique for visualizing large hierarchies. In Michael T. Goodrich and Stephen G. Kobourov, editors, *Graph Drawing (Proceedings of* 10^{th} *International Sympsium on Graph Drawing, 2002)*, volume 2528 of *Lecture Notes in Computer Science*, pages 268–275. Springer, 2002.

[Tre96] L. Trevisan. A note on minimum-area upward drawing of complete and Fibonacci trees. *Inform. Process. Lett.*, 57(5):231–236, 1996.

[Val81] L. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Comput.*, C-30(2):135–140, 1981.

[Wal90] J. Q. Walker II. A node-positioning algorithm for general trees. *Softw. – Pract. Exp.*, 20(7):685–705, 1990.

Index