ELSEVIER

Computational Geometry 15 (2000) 175–202

Computational Geometry
Theory and Applications

www. elsevier.nl/locate/comgeo

Area-efficient algorithms for straight-line tree drawings [☆]

Chan-Su Shin [a,*], Sung Kwon Kim [b,1], Kyung-Yong Chwa [a]

[a] *Department of Computer Science, Korea Advanced Institute of Science and Technology, Taejon 305-701, South Korea*
[b] *Department of Computer Science and Engineering, Chung-Ang University, Seoul 156-756, South Korea*

Communicated by R. Tamassia; submitted 1 October 1996; received in revised form 1 November 1999; accepted 9 November 1999

Abstract

We investigate several *straight-line* drawing problems for bounded-degree trees in the integer grid without edge crossings under various types of drawings: (1) *upward* drawings whose edges are drawn as vertically monotone chains, a sequence of line segments, from a parent to its children, (2) *order-preserving* drawings which preserve the left-to-right order of the children of each vertex, and (3) *orthogonal straight-line* drawings in which each edge is represented as a single vertical or horizontal segment.

Main contribution of this paper is a unified framework to reduce the upper bound on area for the straight-line drawing problems from $O(n\log n)$ (Crescenzi et al., 1992) to $O(n\log\log n)$. This is the first solution of an open problem stated by Garg et al. (1993). We also show that any binary tree admits a small area drawing satisfying any given aspect ratio in the orthogonal straight-line drawing type.

Our results are briefly summarized as follows. Let T be a bounded-degree tree with n vertices. Firstly, we show that T admits an upward straight-line drawing with area $O(n\log\log n)$. If T is binary, we can obtain an $O(n\log\log n)$-area upward orthogonal drawing in which each edge is drawn as a chain of at most two orthogonal segments and which has $O(n/\log n)$ bends in total. Secondly, we present $O(n\log\log n)$-area (respectively, -volume) orthogonal straight-line drawing algorithms for binary trees with arbitrary aspect ratios in 2-dimension (respectively, 3-dimension). Finally, we present some experimental results which shows the area requirements, in practice, for (order-preserving) upward drawing are much smaller than theoretical bounds obtained through analysis. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Graph drawing; Tree drawing; Layout; Drawing area; Aspect ratio

[☆] An extended abstract [24] of this paper was presented in *International Computing* and *Combinatorics Conference* (COCOON'96), held at Hong Kong, 17–19 June 1996. This work was supported in part by Korea Science and Engineering Foundation, No. 94-0100-04-01-3.

[*] Corresponding author.

E-mail addresses: cssin@jupiter.kaist.ac.kr (C.-S. Shin), skkim@cau.ac.kr (S.K. Kim), kychwa@jupiter.kaist.ac.kr (K.-Y. Chwa).

[1] Supported by the Chung-Ang University Special Research Grants in 1998.

1. Introduction

In many applications, constructing geometric representations of graphs in a readable and efficient way is crucial for understanding inherent properties of their structures. The automatic generation of such well-expressed representations is one of main motivations such that *graph drawing* has been receiving considerable attentions over many broad areas of computer science ranging from purely theoretical aspects including graph theory and computational geometry to application-oriented areas including VLSI circuit layout design and visual interface design.

1.1. Problems

A typical graph drawing problem is that given a graph G, generate a geometric representation of G according to several *graphic standards* and *optimization criteria*. Various works in the graph drawing research field are well summarized in the annotated bibliography by Di Battista et al. [10].

A general graph drawing standard is that a vertex of G is represented by a geometric object such as a point, rectangle and cube (or box), and an edge (u, v) is represented by a simple Jordan curve connecting the geometric objects associated with vertices u and v.

According to the kinds of curves to represent edges, graphic standards may have various versions. A *polyline* drawing maps an edge to a polygonal chain, a *straight-line* drawing does to a straight-line segment, and an *orthogonal* drawing does to a chain of orthogonal segments. (In fact, straight-line drawings and orthogonal drawings are special cases of polygonal drawings.) Especially, straight-line and orthogonal standards have been deeply considered by many researchers. The reasons are that it is relatively easy to investigate combinatorial properties of drawings, and the standards are suitable to some important applications such as VLSI circuit design and illustrations of text books in graph theory. Note that edges in polyline and orthogonal drawings may have *bends*, which occur when joining two contiguous line segments in an edge.

There are two other standards commonly used, namely, *grid* and *planar* standards. A drawing is said to be grid if all vertices and bends of edges have integer coordinates, and planar if no two edges intersect in the drawing. All drawings in this paper will be grid and planar, so we will omit the term “grid and planar” hereafter.

In particular, when a rooted tree is considered, it is common that for exhibiting the inherent hierarchy in the rooted tree, every edge between a parent and its child would be represented by a vertically monotone chain so that the parent has y-coordinate greater than or equal to that of the child. A drawing satisfying this condition is said to be *upward*. In addition, a *strictly upward* drawing means that the parent has y-coordinate strictly greater than that of its child.

The quality of a drawing is measured by a combination of optimization criteria such as area, volume, aspect ratio, and the number of bends. The *area* of a 2-dimensional drawing is defined as the area of the smallest axis-parallel rectangle enclosing the drawing. The *height* and *width* of the drawing are the height and width of the rectangle, respectively. The *volume* of a 3-dimensional drawing is similarly defined. In general, to avoid wasting valuable spaces on a page or computer screen, it is important to keep the area or volume of the drawing small. In VLSI industrial field, this criteria is vital to accumulate modules and wires into as little space as possible.

The *aspect ratio* of a drawing is defined to be the ratio of the longest side length to the shortest side length of the enclosing rectangle. A drawing with high aspect ratio is not desirable in the sense that the

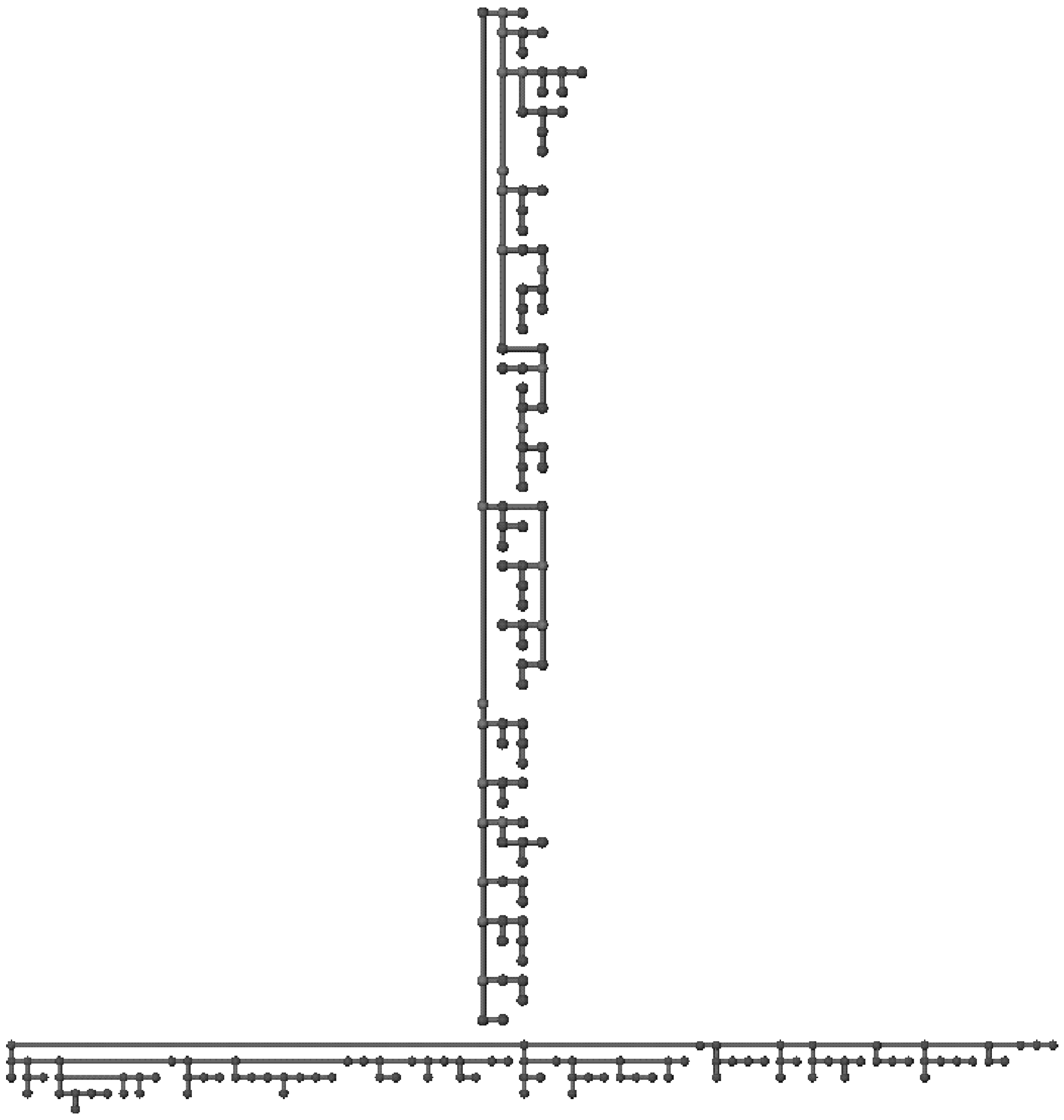

Fig. 1. Drawing examples for a binary tree of 100 vertices with two different ratios of the height to the width; the upper one with ratio of 10:1, the lower one with ratio of 1:16.

drawing may not be conveniently placed on some computer screen. Conversely, if one has only longish windows on the screen, it would be better to have drawings with high aspect ratio. Hence, for providing the flexibility of fitting drawings in arbitrarily shaped windows, it is desirable that one is able to draw graphs with any given aspect ratio. For an illustration, see Fig. 1.

In a polyline drawing (including orthogonal drawing), edges with three or more bends may be difficult to “follow” the course of the edges for the eye. In addition, many bends in a drawing may be the cause of the performance degradation of VLSI circuits. For these reasons, both the *total number of bends* and the *number of bends per edge* should be kept small.

Depending on the application where the drawing will be used, the primary drawing criterion may differ. From the standpoint of VLSI designers, the most important objective is to minimize the area needed for embedding VLSI circuits. However, visual interface designers would prefer a drawing with fewer bends per edge rather than a drawing with smaller area because the drawing with fewer bends are more readable to the user. Thus, the general goal in drawing problems will optimize the appropriately chosen one or more criteria at the same time.

This paper deals with *tree drawing problems* in 2- and 3-dimensions. Primarily, we aim at developing *planar straight-line grid* drawing algorithms for bounded-degree trees so that drawings take up as little area or volume as possible, admit any given aspect ratio, or optimize other criteria such as the number of bends per edge when the straight-line standard is not required.

A (rooted) tree is a fundamental data structure for representing hierarchies of many information structures such as family trees, organization charts, and search trees. For that reason, a lot of tree drawing algorithms [7–9,12,19,28] have been proposed by many researchers in VLSI layout, visual interface, and graph theory fields. Surprisingly, however, there exists a large gap between lower bounds and upper bounds on certain criteria, especially on area. For instance, in upward straight-line tree drawing problems, even in binary tree drawing ones, the best known upper bound on the area is $O(n \log n)$ [6], but its lower bound is $\Omega(n)$. In this paper, we will try to close the gap by lowering the upper bound to $O(n \log\log n)$ for some classes of trees.

1.2. Previous works

Several straight-line drawing algorithms for rooted trees were proposed in many literatures [4,7–9,12, 23,24]. They are summarized in Table 1.

The best known algorithm for the upward straight-line drawing problems was proposed by Crescenzi et al. [7] and Shiloach [23], independently. They showed that any rooted tree admits an upward straight-line drawing with area $O(n \log n)$.

There has been no drawing algorithm producing an upward straight-line drawing with area $o(n \log n)$. Hence, as stated in [12,26], reducing the upper bound remained until now as an open problem.

Under strictly upward straight-line drawing standard, Crescenzi et al. [7] proved that there exists a class of rooted trees requiring area $\Omega(n \log n)$, and presented an algorithm to construct an $O(n \log n)$-area drawing for any rooted tree. They also presented algorithms [7,9] producing $O(n)$-area strictly upward straight-line drawings for some classes of balanced trees. These classes include complete binary trees, Fibonacci trees, and AVL trees. Recently, Crescenzi and Penna [8] showed that trees in a wider class of balanced trees can be drawn in linear area. They called them *logarithmic trees*, which satisfy that the height of any (sufficiently high) subtree is logarithmic with respect to the number of vertices. The class contains most of balanced search trees, including k-balanced trees, red–black trees, $BB[\alpha]$-trees, and (a, b)-trees.

A drawing for an ordered tree is said to be *order-preserving* if the drawing preserves the left-to-right order of the children of each vertex. Garg et al. [12] showed that there is a family of trees requiring

Table 1
Drawing standards are represented as abbreviations: Up. (Upward), S-Up. (Strictly Upward), SL (Straight-Line), Ortho. (Orthogonal), Order (Order-preserving), and Poly. (Polyline). The term “b.p.e.” means the number of bends per edge and “A.R.” is an abbreviation of aspect ratio. “Balanced” trees means the classes of trees including k-balanced, red–black, BB $[\alpha]$-, and (a, b)-trees

		Tree drawing results			
		Previous results		Our results	
Drawing type	Tree type	Sources	Prev. (area) bound	Our (area) bound	
Up. SL	general	[7]	$\Omega(n)$, $O(n \log n)$	–	–
	deg-O(1)	[7]	$\Omega(n)$, $O(n \log n)$	$O(n \log\log n)$	Theorem 1
Up. Order SL	general	[3,7]	$\Omega(n \log n)$, $O(n^{1+\varepsilon})$	–	–
	balanced	[7–9]	$\Theta(n)$	$O(n \log\log n)$	Theorem 3
S-Up. SL	general	[7]	$\Theta(n \log n)$	–	–
	balanced	[7–9]	$\Theta(n)$	–	–
S-Up. Order SL	general	[3,7]	$\Omega(n \log n)$, $O(n^{1+\varepsilon})$	–	–
	balanced	[7–9]	$\Theta(n)$	$O(n(\log\log n)^2)$	Theorem 4
Up. Ortho.	binary	[12,16]	$\Theta(n \log\log n)$	$\Theta(n \log\log n)$	Theorem 2
			$O\left(\frac{n}{\log n}\right)$ bends	$O\left(\frac{n}{\log n}\right)$ bends	
			4 b.p.e.	1 b.p.e.	
Ortho. SL	binary (2D)	[7]	$\Omega(n)$, $O(n \log n)$	$O(n \log\log n)$	Theorem 5
			A.R. $O\left(\frac{n}{\log n}\right)$	arbitrary A.R.	
	binary (3D)	[5]	$\Omega(n)$, $O(n \log n)$	$O(n \log\log n)$	Theorem 6
			A.R. $O\left(\frac{\sqrt{n}}{\log n}\right)$	arbitrary A.R.	
Ortho.	deg-4 (2D)	[19,28]	$\Theta(n)$	$O(n)$	Section 8
			$O(\log n)$ b.p.e.	$O(\log\log n)$ b.p.e.	
			A.R. $O(1)$	arbitrary A.R.	
	deg-6 (3D)	[18]	$\Theta(n)$	$O(n)$	Section 8
			$O(1)$ b.p.e.	$O(1)$ b.p.e.	
			A.R. $O(1)$	(almost) arbitrary A.R.	
Up. Polyline	deg-$O(n^{\delta})$	[12]	$\Theta(n)$	–	–

$\Omega(n \log n)$ area under the order-preserving upward standard. They also presented an $O(n \log n)$-area upward polyline drawing algorithm.

Garg et al. [12] presented an upward orthogonal drawing algorithm for any binary tree with $O(n \log\log n)$ area, which was shown to be asymptotically optimal. If the upward requirement is relaxed, any tree with maximum degree four admits an orthogonal drawing with area $O(n)$ [19,28]. However, it has not been known whether or not non-upward straight-line drawings with area less than $O(n \log n)$ [7] are possible.

1.3. Our results

In this paper, we present solutions to the unsolved upward straight-line drawing problems and other related problems for bounded-degree trees. A *bounded-degree* tree throughout the paper indicates a tree with constant maximum degree.

All drawing algorithms presented in this paper are based on a unified drawing framework, so-called "partition-and-merge" strategy, which was used for producing small area drawings of a variety of families of graphs including trees [2,19,28]. The strategy for a tree is to partition a tree into several pieces by deleting some edges of the tree, draw each piece independently, and then merge the drawings of the pieces by inserting (drawing) the deleted edges.

The effectiveness of the strategy heavily depends on how to partition a tree. A well-known partitioning method is based on *planar separator theorem* due to Lipton and Tarjan [20], which partitions a tree into two pieces of size almost half. However, the method requires too much additional area when the deleted edges are inserted to merge the drawings of pieces. Thus, we propose a new partitioning method that one can merge the drawing pieces with a little additional area, which is a variant of [13,16].

- Results presented in this paper are summarized in Table 1. We first present an $O(n \log\log n)$-area *upward straight-line* drawing algorithm for any bounded-degree tree. This is the first result to reduce the upper bound from $O(n \log n)$ [7] to $O(n \log\log n)$ (Section 4). If the constant value hidden in $O(n \log\log n)$ is quite large, then our algorithm may become worse than the $O(n \log n)$-area algorithm [7]. The experiment performed in Section 7, however, shows the hidden constant value is sufficiently small.
- Through a minor modification of the upward straight-line drawing algorithm, we can obtain an $O(n \log\log n)$-area *upward orthogonal drawing* with at most one bend per edge and $O(n/\log n)$ bends in total. The best known upward orthogonal drawing algorithms were proposed by Garg et al. [12] and Kim [16]. Their algorithms produce drawings with $O(n \log\log n)$ area, $O(n/\log n)$ bends, and at most four bends per edge. Our algorithm is superior to their algorithms in the sense that the number of bends per is at most one and the other criteria remain the same.
- We present area-efficient *order-preserving* (*strictly*) *upward straight-line* drawing algorithms for some classes of balanced search trees (Section 5). The classes cover most of balanced search trees widely used in computer science including k-balanced trees, red–black trees [20], BB[α]-trees [20] and (a, b)-trees [21], where k, a and b are fixed constants, and $2 \leqslant a \leqslant b$. It is worthwhile to draw search trees in order-preserving fashion because the values stored at the vertices of a tree should be kept sorted. With upward standard, balanced trees in the classes can be drawn in area $O(n \log\log n)$. If we consider a strictly upward standard, they can be drawn in area $O(n(\log\log n)^2)$. As stated previously, the optimal linear-area algorithm [8,9] for drawing trees in those classes has already been known. We wish to show that our drawing framework is also suitable to draw some balanced trees under order-

preserving standard even though the theoretical bound on area is not good as those in [8,9]. The authors in [9] showed that AVL trees can be drawn in the area no greater than $36n$, but they did not analyzed how large the multiplicative constant in the area function is for the other classes of balanced trees, and said that the constant value is generally in the order of thousands, which is clearly infeasible. In this paper, we show that the constant value in our algorithm for all those classes of balanced trees is sufficiently small; for example, 46.1 for AVL trees, 64 for red–black trees, and 48 for $(2,3)$-trees; this is also complemented by the experimentation in Section 7. We believe that our algorithm will perform well in many practical situations.

- Applying our drawing framework to non-upward tree drawing problems, we can obtain a *non-upward orthogonal straight-line* drawing for any binary tree such that (1) the area is $\mathrm{O}(n\log\log n)$, (2) the aspect ratio can be an arbitrary value in the range of $[\mathrm{O}(1), \mathrm{O}(n\log\log n/\log^2 n)]$, and (3) every edge is drawn either as a single horizontal or vertical segment, which will be called an *orthogonal straight-line* drawing. This is the first result to guarantee a drawing with $\mathrm{o}(n\log n)$ area and permit an arbitrary aspect ratio. In addition, we can directly extend the 2-dimensional $\mathrm{O}(n\log\log n)$-area drawing to a 3-dimensional $\mathrm{O}(n\log\log n)$-volume one with arbitrary aspect ratio. This is also the first result to guarantee a drawing with $\mathrm{o}(n\log n)$ volume and any given aspect ratio. The previously best known result was presented by Cohen et al. [5]. They showed that any binary tree can be drawn in volume of $\mathrm{O}(n\log n)$ and with any aspect ratio in the range of $[\mathrm{O}(\sqrt{n}/\log n), \mathrm{O}(n)]$. However, edges in the drawing may be drawn as non-orthogonal segments. Furthermore, their algorithm cannot produce a drawing with aspect ratio $\mathrm{o}(\sqrt{n}/\log n)$.
- We, finally, introduce other related problems and show that the unified framework can be applicable to solve them efficiently in Section 8.

All drawing algorithms presented in this paper run in linear time. Since there are no intricate parts in implementing our algorithms in linear time, we will omit the time analysis.

Remark. After the preliminary version [24] of this paper was published, Chan et al. [4] independently proposed a tree drawing strategy. Although they called it "recursive winding" strategy, two strategies (including the partitioning methods) stem from the same idea; in fact, they are essentially identical.

2. Preliminaries

We begin with basic notations and definitions that will be used throughout the paper.

The *degree* of a vertex v in a tree T is the number of edges incident to v. The *height* of v in T is defined as the maximum of the lengths[2] of paths from v to leaves in T. The *tree-height* of T is the height of its root. A subtree rooted at v in T is denoted by T_v. The *size* of T, *size*(T), is the number of vertices in T.

An *ordered tree* is a rooted tree in which the children of each vertex are ordered from left to right. Let T be an ordered tree of n vertices with maximum degree $d = \mathrm{O}(1)$. The *leftmost* (respectively, *rightmost*) *path* of T is a maximal path consisting of the leftmost (respectively, rightmost) edges only starting at the root. An ordered tree T is *left-heavy* if, for each vertex of T, its subtrees are ordered from left to right by non-increasing order of their sizes. A *right-heavy* tree is defined in a symmetric way. We denote by $\mathrm{NL}(T)$ (respectively, $\mathrm{NR}(T)$) the maximum number of non-leftmost (respectively, non-rightmost) edges

[2] The *length* of a path is defined to be the number of vertices on the path.

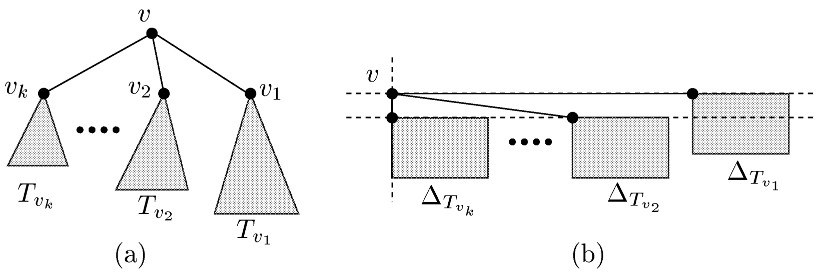

Fig. 2. An upward straight-line drawing Δ_T for a right-heavy tree T.

of a path from the root to a leave in T. It is easy to see that if T is left-heavy, then $\mathrm{NL}(T) \leqslant \lfloor \log n \rfloor$ and if T is right-heavy, then $\mathrm{NR}(T) \leqslant \lfloor \log n \rfloor$.

Let Δ_T be a drawing for an ordered tree T produced by some drawing algorithm. If a 2-dimensional drawing Δ_T can be enclosed by a rectangle of height H_T (with respect to Y-axis) and width W_T (with respect to X-axis), we call it a drawing with area $H_T \times W_T$. Similarly, a 3-dimensional drawing Δ_T with height H_T, width W_T and depth D_T (with respect to Z-axis) is called a drawing with volume $H_T \times W_T \times D_T$.

We say that a vertex v of Δ_T is *south-open* if a ray emanating from v in Δ_T with south direction does not intersect Δ_T except at v. Similarly, we can define a vertex in Δ_T to be east-open, southwest-open, etc.

We now introduce a notion of *enlargement* of a straight-line drawing. Let Δ_T be a straight-line drawing with height H_T and width W_T in 2-dimension. An enlargement of Δ_T is defined to be a transformation of Δ_T to Δ'_T that is a topologically equivalent[3] drawing with height $H_T + d_h$ and width $W_T + d_w$, where d_h and d_w are non-negative integers. The drawing Δ'_T can be obtained from Δ_T as follows. First, transform Δ_T to a topologically equivalent drawing with $H_T + d_h$ and W_T by moving all vertices and edges below $y = y'$ in Δ_T by d_h units downward, and then by enlarging all edges that formerly passed through the space between $y = y'$ and $y = y' - 1$. Second, transform the drawing to Δ'_T with $H_T + d_h$ and $W_T + d_w$ in a symmetric way. Such enlargement of Δ_T does not violate any drawing standards of Δ_T. An enlargement of a 3-dimensional drawing can be similarly defined. Accordingly, whenever it is necessary, one can enlarge Δ_T to a topologically equivalent drawing of arbitrary size larger than Δ_T.

Finally, we review an $\mathrm{O}(n \log n)$-area upward straight-line drawing algorithm [7] for any ordered tree that will be used as a subroutine in our drawing algorithms. Let T be an ordered tree of n vertices with maximum degree d.

1. Transform T into a right-heavy tree if it is not right-heavy.
2. Suppose that the root v of T has k children, $v_1, v_2, \ldots, v_k$, in right-to-left order. Recursively draw T_{v_i} for each i. Horizontally arrange their drawings so that $\Delta_{T_{v_{i+1}}}$ is placed one unit away to the left of $\Delta_{T_{v_i}}$ for $1 \leqslant i \leqslant k-1$ and only $\Delta_{T_{v_1}}$ is again shifted up by one unit (see Fig. 2). Next, place the root v at the intersection point of the horizontal line containing the top side of $\Delta_{T_{v_1}}$ and the vertical line containing the left side of $\Delta_{T_{v_k}}$.
3. Draw an edge (v, v_i) as a straight line for each i. Then the leftmost edge (v, v_k) is drawn as a vertical segment and the rightmost edge (v, v_1) is drawn as a horizontal segment.

[3] Two drawings are *topologically equivalent* if adjacencies of vertices and edges in two drawings are perfectly identical.

We can easily observe that the height of Δ_T is proportional to the length of the leftmost path of T. Using the fact that T is right-heavy, the height of Δ_T has at most $\lfloor \log n \rfloor$. Clearly, the width of Δ_T is at most $n-1$. Hence we can get the following lemma.

Lemma 1 [7]. *For any n-vertex tree T with a constant degree, we can have an upward straight-line drawing Δ_T such that $H_T \leqslant \lfloor \log n \rfloor$, $W_T \leqslant n-1$. In addition, the root of T is placed at the upper-left corner of Δ_T.*

Notice that we can draw a tree T of n vertices in height $H_T \leqslant n-1$ and width $W_T \leqslant \lfloor \log n \rfloor$ through an analogous way to the above algorithm.

Lemma 2 [7]. *For any n-vertex tree T with a constant degree, we can have an upward straight-line drawing Δ_T such that $H_T \leqslant n-1$ and $W_T \leqslant \lfloor \log n \rfloor$. In addition, the root of T is placed at the upper-left corner of Δ_T.*

We shall refer to both algorithms of Lemmas 1 and 2 as algorithms $\mathcal{A}_h$ and $\mathcal{A}_v$, respectively, which are named according to the recursive step where subtree drawings are arranged horizontally or vertically.

Remark. If T is a binary tree, then each edge in Δ_T produced either by Lemma 1 or by Lemma 2 is drawn either as a horizontal segment or a vertical segment. Thus Δ_T becomes an upward orthogonal straight-line drawing of T.

3. Partition-and-merge drawing strategy

There is a general-purpose drawing strategy that produces small area drawings for a wide variety of families of graphs, including trees and planar graphs. That is the *partition-and-merge* strategy: (i) partition a tree T into pieces by deleting some edges, (ii) draw these pieces, and (iii) merge the drawings of the pieces by inserting and drawing the deleted edges among the drawings. An algorithm of drawing each of the pieces will be called the *base drawing algorithm*, and an algorithm of merging the drawings of the pieces will be called the *merging algorithm*. Then a drawing algorithm based on the partition-and-merge strategy is fully described by three elements: a partitioning method $\mathcal{P}$, a base drawing algorithm $\mathcal{B}$, and a merging algorithm $\mathcal{M}$. Several tree drawing algorithms [2,12,16,19,27,28] including our drawing algorithms adopt this strategy.

Formally, an n-vertex rooted tree T is partitioned by $\mathcal{P}$ into $\mathrm{O}(n/m)$ partial trees[4] each of which has size at most $\mathrm{O}(m)$, where m is a *partition parameter* $\leqslant n$. In particular, we shall call the resulting partial trees *fragments*. This partitioning is done by deleting $\mathrm{O}(n/m)$ edges, called *separators*. Each of the fragments is drawn by $\mathcal{B}$, and then their drawings are placed appropriately and the deleted edges are restored and drawn around the fragment drawings by $\mathcal{M}$.

[4] A *partial tree* of T simply means a connected subgraph of T. Note that a subtree of T rooted at a vertex v is a partial tree of T rooted at v that includes all descendants of v.

3.1. Partitioning method

A well-known partitioning method used in [2,12,19,28] is based on the planar separator theorem due to Lipton and Tarjan [20]. The separators are determined by recursively finding an edge whose removal divides T into two fragments, each with at least $n/3$ vertices and at most $2n/3$ vertices [15].

In [16], the separators are determined by introducing a notion of the *critical vertex*, which was originally used for parallel tree contractions by Gazit et al. [13]. But, this method is applied to only the bounded-degree trees. Our partitioning method is a variation of this method.

We now present a partitioning method which will be used in our drawing algorithms in this paper. Unlike other methods, our method has a property, which is desirable to produce small area tree drawings. The property will be explained at the end of the subsection.

Let T be a n-vertex tree with maximum degree of d. For a partition parameter m $(\leqslant n)$, a vertex v of T is *m-critical* if v is not a leaf and $\lceil \text{size}(T_v)/m \rceil > \lceil \text{size}(T_{v'})/m \rceil$ for all children v' of v. Let v be an m-critical vertex with a child w. Since $\lceil \text{size}(T_v)/m \rceil > \lceil \text{size}(T_w)/m \rceil \geqslant 1$, $\text{size}(T_v)$ must be no less than $m+1$. In other words, if $\text{size}(T_v) \geqslant 2m+1$, then T_v must contain at least one m-critical vertex. From the definition of the m-critical vertex, we can further observe that the least common ancestor of any two m-critical vertices is also m-critical.

Our partitioning method is summarized below.

```
Procedure PartitionTree($T, m$)
1. find all $m$-critical vertices of $T$.
2. for every $m$-critical vertex $v$ in $T$ do
       define the edges incident to $v$ as separators of $T$.
3. delete the separators from $T$.
4. return the fragments and the separators of $T$.
```

Lemma 3. *For any tree T with maximum degree d and a given integer m $(\leqslant n)$, the procedure PartitionTree produces at most $2dn/m$ fragments, each of which has at most m vertices.*

Proof. The number of fragments is bounded by the number of separators of T, which is the number of m-critical vertices times the maximum degree d of T. Gazit et al. [13] proved that the number of m-critical vertices is at most $2n/m - 1$. Thus there are at most $2dn/m$ fragments. From the fact that any subtree with vertices $\geqslant 2m+1$ contains at least one m-critical vertex, we can easily conclude that the size of a fragment is at most m. □

The procedure PartitionTree naturally defines a rooted tree, called a *fragment tree FT* of T, in which each vertex corresponds to a fragment and there is an edge between two fragments F_1 and F_2 in *FT* if there exists a separator (v, w) in T such that $v \in F_1$, $w \in F_2$ and v is the parent of w. Then F_1 is the parent of F_2 in *FT* and w becomes the root of F_2. We call v a *connection vertex* of F_1 and say the separator (v, w) to be *incident* to F_1 and F_2. Clearly, a fragment tree *FT* of T consists of $O(n/m)$ vertices whose maximum degree may be at most d.

A fragment is *trivial* if it has only one vertex. Since all edges incident to an m-critical vertex are defined as separators, every m-critical vertex itself is a trivial fragment in *FT*. The fact also implies that every non-trivial fragment in *FT* has no m-critical vertices.

Our partitioning method has a good property that each non-trivial fragment has at most one connection vertex. This is, in general, not true for other partitioning methods [12,19,20,27,28], in which a non-trivial fragment may have two or more connection vertices (possibly $O(\log n)$ ones). Let F be a non-trivial fragment with a child fragment F'. There is a separator $s = (v, w)$ such that v is a connection vertex in F and w is the root of F'. To draw s in the merging step, we should make enough room to connect v and w without edge-crossings in Δ_F and $\Delta_{F'}$. As a consequence, the additional area needed to draw separators incident to F is proportional to the number of connection vertices of F. Accordingly, to perform the merging step with a little cost, it is desirable that each non-trivial fragment has as less connection vertices as possible. The following lemma, hence, plays a crucial role in breaking the upper bound, $O(n \log n)$, on area.

Lemma 4. *Let FT be a fragment tree of T produced by the procedure PartitionTree. Then each non-trivial fragment in FT has at most one child fragment, that is, has at most one connection vertex. Moreover, if exists, the unique child fragment is trivial.*

Proof. For contradiction, suppose that F has two or more children $F_1, F_2, \ldots, F_k$ in FT. Let (v_i, w_i) be a separator between F and F_i, where v_i is a connection vertex of F and w_i is the root of F_i. By the definition of the separator, w_i for each i must be m-critical, thereby the least common ancestor u of w_i and w_j is also m-critical, so u itself constitutes a trivial fragment.

Since, however, F is connected, u must belong to F. Consequently, F consists of two or more fragments, which is a contradiction. Now we will show that every non-trivial fragment F has a trivial fragment as its unique child (if exists) in FT. Let (v, w) be a separator connecting F and one of its child fragments, F', where v is a connection vertex of F and w is the root of F'. By the definition of the separator, at least one of v and w must be m-critical. However, since v belongs to a non-trivial fragment F, v is not m-critical. Hence w must be m-critical. This means that w itself is F', i.e., F' is a trivial fragment. □

Let F be a fragment of FT. We shall denote by c_F the connection vertex of F (if it exists), and denote by r_F the root of F. If F is trivial, then $r_F = c_F$. If F is non-trivial, then c_F is uniquely defined by the above lemma.

3.2. Two merging methods

Once a fragment tree FT is generated by $\mathcal{P}$ and the fragments F are drawn by a base drawing algorithm $\mathcal{B}$, their drawings Δ_F's will be arranged by a merging algorithm $\mathcal{M}$. We shall explain two merging algorithms $\mathcal{M}_v$ and $\mathcal{M}_h$ which will be used in the drawing algorithms later. The algorithm $\mathcal{M}_h$ vertically stacks Δ_F's from top to bottom, and the other algorithm $\mathcal{M}_h$ horizontally arranges Δ_F's from left to right. Thus $\mathcal{M}_v$ produces a placement whose height relatively increases more than its width does, whereas $\mathcal{M}_h$ produces a placement whose width relatively increases more than its height does.

The algorithm $\mathcal{M}_v$ places Δ_F's vertically according to the reverse preorder [5] of FT. To do it, we need two assumptions about Δ_F's: (i) r_F lies on the left side of Δ_F and is north-open, and (ii) c_F lies on the right side of Δ_F and is south-open.

[5] To traverse T in the reverse preorder, first visit the root of T, and then recursively visit the subtrees of v in the left-to-right order.

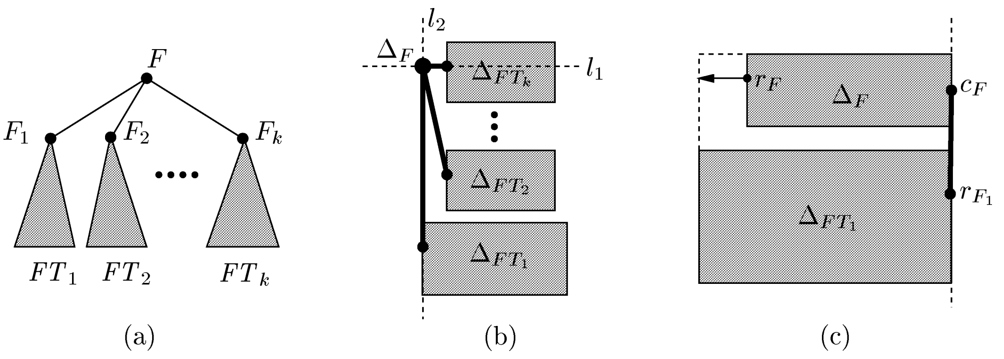

Fig. 3. Recursive construction of $\varDelta_{FT}$ by $\mathcal{M}_v$.

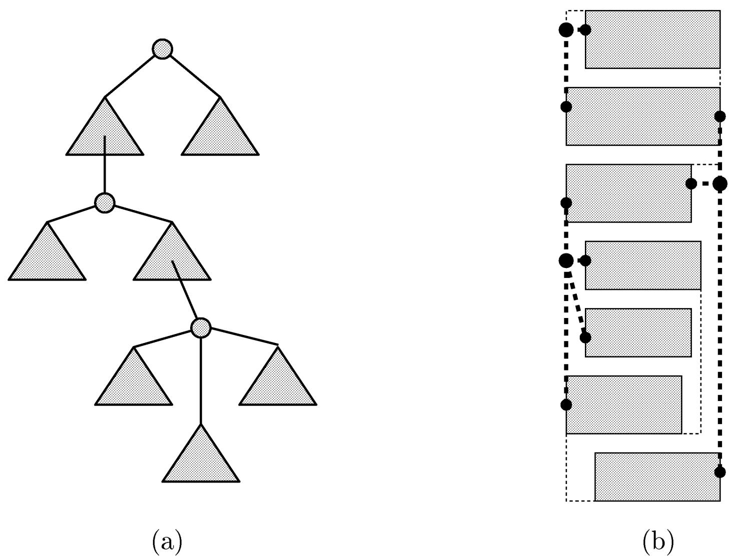

Fig. 4. (a) A fragment tree. Circles represent trivial fragments. (b) Placements of $\varDelta_F$'s by $\mathcal{M}_v$.

Suppose that FT has the root fragment F whose child fragments are $F_1, F_2, \ldots, F_k$ from left to right. Let FT_i be the subtree rooted at F_i in FT (see Fig. 3(a)). Recursively construct the drawing for FT_i so that the following invariant is satisfied: r_{F_i} is placed on the left side of $\varDelta_{FT_i}$ and is north-open.

When F is trivial, vertically stack the drawings $\varDelta_{FT_i}$'s so that $\varDelta_{FT_i}$ is directly below $\varDelta_{FT_{i+1}}$ (see Fig. 3(b)). Place F at the intersection of a horizontal line l_1 passing through r_{F_k} and a vertical line l_2 containing the left side of $\varDelta_{FT_1}$. Then draw each separator $(c_F; r_{F_i})$ for i as a straight line; this is always possible because of the invariant about r_{F_i}. The invariant clearly holds for $\varDelta_{FT}$.

When F is a non-trivial fragment with the unique child fragment F_1, horizontally flip[6] $\varDelta_{FT_1}$ and place $\varDelta_{FT_1}$ directly below $\varDelta_F$ so that the right sides of $\varDelta_F$ and $\varDelta_{FT_1}$ are aligned at a vertical line (see

[6] A horizontally flipped drawing of a drawing $\varDelta_T$ is a topologically equivalent drawing of $\varDelta_T$ that is obtained by mirroring $\varDelta_T$ with respect to y-axis.

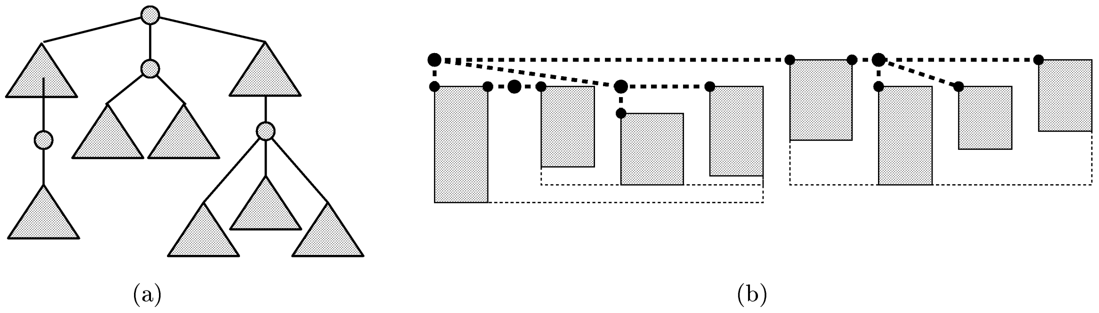

Fig. 5. Placements of Δ_F's by $\mathcal{M}_h$.

Fig. 3(c)). Then we can draw a separator (c_F, r_{F_1}) as a straight line without intersecting Δ_F and Δ_{FT_1} except at c_F and r_{F_1}, by the second assumption about c_F in Δ_F and the invariant about r_{F_1} in Δ_{FT_1}. But r_F is not necessarily placed on the left side of Δ_{FT}. To satisfy the invariant for Δ_{FT}, we enlarge Δ_F so that its width is equal to that of Δ_{FT_1}. Since r_F is still located on the left side of the enlarged drawing and is north-open, the invariant for Δ_{FT} holds. Fig. 4 shows an example of a final drawing.

Unlike $\mathcal{M}_v$, the algorithm $\mathcal{M}_h$ arranges Δ_F's from left to right according to the preorder of FT. We need assumptions about Δ_F's which are slightly different from those in $\mathcal{M}_v$: (i) r_F lies on the upper-left corner of Δ_F, and (ii) c_F lies on the upper-right corner of Δ_F. Then $\mathcal{M}_h$ arranges Δ_F's from left to right in a similar way to $\mathcal{M}_v$ except that no flipping operations are needed. The detail of $\mathcal{M}_h$ is straightforward; for an example, refer to Fig. 5.

Lemma 5. *Suppose that a binary tree T is partitioned by* PartitionTree(T, m). *Let FT be the fragment tree of T. Suppose that each fragment F of FT is drawn by some base drawing algorithm $\mathcal{B}$ and its drawing Δ_F has height $\leqslant H_F$ and width $\leqslant W_F$. Then the merging algorithm $\mathcal{M}_v$ produces Δ_T with height no more than $H_F \times \mathrm{size}(FT)$ and width at most $W_F + \mathrm{NL}(FT)$, and $\mathcal{M}_h$ produces Δ_T with height at most $H_F + \mathrm{NR}(FT)$ and width at most $W_F \times \mathrm{size}(FT)$.*

Proof. Consider only $\mathcal{M}_v$; the proof for $\mathcal{M}_h$ can be similarly done. Since $\mathcal{M}_v$ stacks Δ_F's vertically, each drawing ΔF of $F \in FT$ contributes to the height of Δ_T at most by H_F. Thus the height is at most $H_F \times \mathrm{size}(FT)$. Next, consider the width W_T of Δ_T. We now show by induction on the tree-height of FT that $W_{FT} \leqslant W_F + \mathrm{NL}(FT)$. Note that $W_T = W_{FT}$. When FT consists of only one fragment F, it obviously holds because $\mathrm{NL}(FT) = 0$. Inductively, we consider FT with root fragment F whose subtrees are $FT_1, \ldots, FT_k$. If F is trivial, by our drawing algorithm (see Fig. 3(b)),

$$W_{FT} = \max\{W_{FT_1}; W_{FT_2} + 1, \ldots, W_{FT_k} + 1\}.$$

By induction hypothesis,

$$\begin{aligned} W_{FT} &= \max\{W_F + \mathrm{NL}(FT_1), W_F + \mathrm{NL}(FT_2) + 1, \ldots, W_F + \mathrm{NL}(FT_k) + 1\} \\ &\leqslant W_F + \max\{\mathrm{NL}(FT_1), \mathrm{NL}(FT_2) + 1, \ldots, \mathrm{NL}(FT_k) + 1\} \\ &= W_F + \mathrm{NL}(FT). \end{aligned}$$

Note that

$$\mathrm{NL}(FT) = \max\{\mathrm{NL}(FT_1), \mathrm{NL}(FT_2) + 1, \ldots, \mathrm{NL}(FT_k) + 1\}$$

by definition. If F is non-trivial (see Fig. 3(c)),

$$W_{FT} = \max\{W_{FT_1}, W_F\} \leqslant \max\{W_F + \mathrm{NL}(FT_1), W_F\} = W_F + \mathrm{NL}(FT).$$

Note that $\mathrm{NL}(FT) = \mathrm{NL}(FT_1)$. For both cases, $W_T = W_{FT} \leqslant W_F + \mathrm{NL}(FT)$. $\square$

Remark. Consider how $\mathcal{M}_v$ and $\mathcal{M}_h$ work when T is a binary tree. Since every fragment in FT has at most two children, each separator is drawn either as a horizontal or vertical segment. So, if all edges in Δ_F are drawn as horizontal or vertical segments, then Δ_T merged by $\mathcal{M}_v$ and $\mathcal{M}_h$ becomes an orthogonal straight-line drawing.

4. Upward straight-line drawing algorithm

Let T be a rooted tree of n vertices with maximum degree of d. Applying the partition-and-merge strategy, we obtain an upward straight-line drawing Δ_T with height $\mathrm{O}(n \log\log n/\log n)$, width $\mathrm{O}(\log n)$ and area $\mathrm{O}(n \log\log n)$. We will use PartitionTree as $\mathcal{P}$, a variant of algorithm $\mathcal{A}_h$ of Lemma 1 as $\mathcal{B}$, and $\mathcal{M}_v$ as $\mathcal{M}$.

First, partition T by PartitionTree(T, m) with partition parameter $m = \log n$. To avoid degenerate cases, we modify the partition so that $r_F \neq c_F$ for every non-trivial fragment $F \in FT$. If $r_F = c_F$ for a non-trivial fragment F, then, by deleting the edges between r_F and its children in F, we make r_F itself a new trivial fragment and make each subtree of r_F in F a new non-trivial fragment. Note that none of these new non-trivial fragments has a connection vertex. This modification increases the number of fragments at most $(d-2)$ times. Thus there are at most $2d(d-2)n/m$ in the modified FT.

Next, draw each fragment by a variant of algorithm $\mathcal{A}_h$ of Lemma 1. Its detail is as follows. See Fig. 6(a). Let $P = (r_F = v_1, v_2, \ldots, v_h = c_F)$ be a path from r_F to c_F in F; if c_F does not exist for F, then we pick any leaf in F as c_F. Let $v_{i1}, v_{i2}, \ldots, v_{ik_i}$ be the children of v_i (except v_{i+1}). Let F_{ij} be the subtree rooted at v_{ij} $(1 \leqslant j \leqslant k_i)$ in F.

1. Draw F_{ij} for all i and j by $\mathcal{A}_h$. Then $\Delta_{F_{ij}}$ has height $H_{F_{ij}} \leqslant \log(\mathrm{size}(F_{ij}))$ and width $W_{F_{ij}} \leqslant \mathrm{size}(F_{ij}) - 1$. Note that the root of F_{ij} is placed at the upper-left corner of Δ_{ij}.
2. Place $\Delta_{F_{ij}}$ as follows. To assist the following description, we prepare two horizontal lines l_1 and l_2 and one vertical line l_3 as shown in Fig. 6(b). Horizontal line l_2 is one unit below l_1. Place all $\Delta_{F_{ij}}$'s

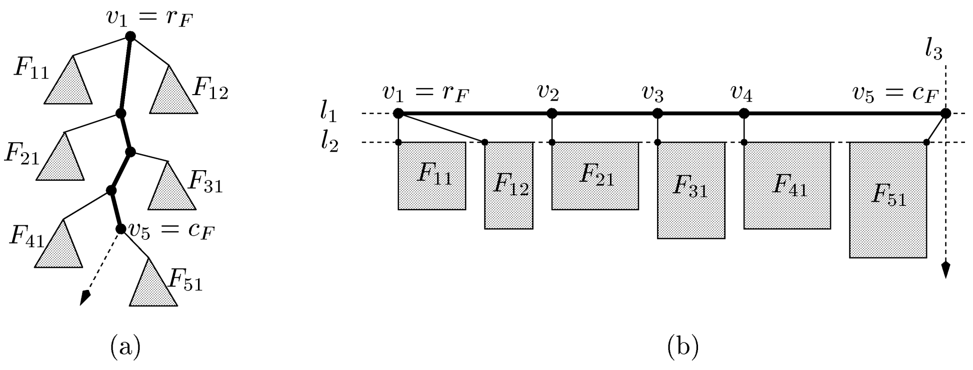

Fig. 6. A drawing Δ_F of a non-trivial fragment F.

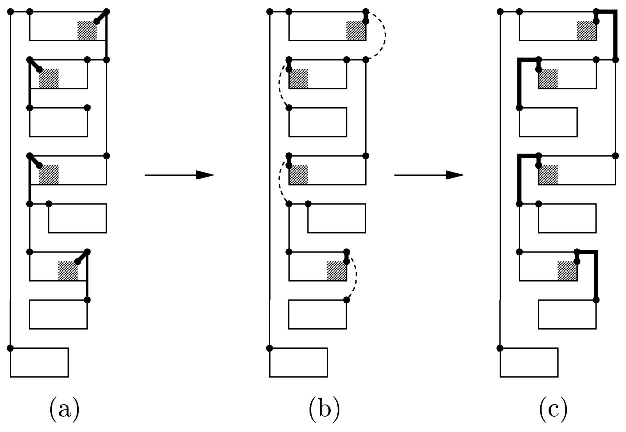

Fig. 7. An orthogonal drawing admitting one bend per edge when T is binary.

for $1 \leqslant i \leqslant h-1$ left-to-right with their roots on l_2. Put v_i for $1 \leqslant i \leqslant h-1$ on l_1 directly above the root of $\Delta_{F_{i1}}$. Next, horizontally flip $\Delta_{F_{hj}}$'s for $1 \leqslant j \leqslant k_h$ and place them left-to-right with their roots on l_2. Here, the vertical line l_3 is assumed to be placed one unit away to the right of $\Delta_{F_h k_h}$. Then, put $v_h (= c_F)$ at the intersection of l_1 and l_3. Finally, draw the edges on P and incident to P as straight line segments. Notice here that if we deal with binary trees, then the edges (v_i, v_{i1}) for $1 \leqslant i \leqslant h-1$ (except for (v_h, v_{h1})) are drawn as vertical segments.

Let us now calculate H_F and W_F. Since $H_{F_{ij}} \leqslant \log(\text{size}(F_{ij}))$ and $W_{F_{ij}} \leqslant \text{size}(F_{ij}) - 1$, $H_F \leqslant \max_{i,j}\{H_{F_{ij}}\} + 1 \leqslant \max_{i,j}\{\log(\text{size}(F_{ij}))\} + 1 \leqslant \log\log n + c$ for some constant c, and $W_F \leqslant \sum_{i,j}(W_{F_{ij}} + 1) \leqslant \sum_{i,j} \text{size}(F_{ij}) \leqslant \text{size}(F) \leqslant \log n$.

To apply $\mathcal{M}_v$ as a merging algorithm, we have to check conditions about r_F and c_F in a non-trivial fragment drawing Δ_F. In Δ_F, r_F is on the upper-left corner and is north-open, and c_F is on the upper-right corner and is south-open. Thus we can apply $\mathcal{M}_v$ to merge Δ_F's. From Lemma 5, a final drawing Δ_T has $H_T \leqslant H_F \times \text{size}(FT)$ and $W_T \leqslant W_F + \text{NL}(FT)$. If we transform FT into a left-heavy tree before applying $\mathcal{M}_v$, then $\text{NL}(FT) \leqslant \log(\text{size}(FT)) = \text{O}(\log(n/m)) = \text{O}(\log n)$. Thus $H_T = \text{O}(\log\log n \times (n/m)) = \text{O}(n \log\log n/\log n)$ and $W_T = \text{O}(\log n)$. Since all edges in Δ_T are drawn according to the upward standard, we have the following theorem.

Theorem 1. *Any bounded-degree n-vertex tree T admits an upward straight-line drawing with height* $\text{O}(n \log\log n/\log n)$*, width* $\text{O}(\log n)$ *and area* $\text{O}(n \log\log n)$.

Consider the case when T is a binary tree. When we are drawing a non-trivial fragment F, the edges (v_i, v_{i1}) for $1 \leqslant i \leqslant h-1$ are drawn as vertical segments, and only the edge (v_h, v_{h1}) is drawn as a non-vertical segment. Note here that $v_h = c_F$. In other words, at most one edge in F, which is incident to c_F, is drawn as a non-orthogonal segment in Δ_F. As a result, $\text{O}(n/\log n)$ edges of T which are incident to connection vertices are only drawn as non-orthogonal segments (see Fig. 7(a)). We will now explain how to convert each of these non-orthogonal segments to an orthogonal chain with at most one bend. First, when drawing a non-trivial fragment F, we place c_F directly above $\Delta_{F_h k_h}$ not at the intersection of l_1 and l_3, and thus the non-orthogonal edge between c_F and its child of F is drawn as a vertical segment (see Fig. 7(b)). Next, we draw each separator as a chain of one horizontal segment and one vertical segment as shown in Fig. 7(c). Clearly, the chains do not intersect the drawing. The height is still $\mathrm{O}(n \log\log n/\log n)$ and the width is $\mathrm{O}(\log n)$. Of course, the drawing has at most one bend per edge and $\mathrm{O}(n/\log n)$ bends in total.

Theorem 2. *A binary tree T has an $\mathrm{O}(n \log\log n)$-area upward orthogonal drawing with height $\mathrm{O}(n \log\log n/\log n)$ and width $\mathrm{O}(\log n)$. In addition, the drawing has at most one bend per edge and $\mathrm{O}(n/\log n)$ bends in total.*

Remark. The constant values in the area obtained by Theorems 1 and 2 are not greater than $4d(d-2)$. In particular, the value for binary trees is 12 because $d = 3$. However, as seen in Section 7, our experiment for binary trees with 10^5 vertices or less shows the value does not exceed 1.2.

5. Order-preserving upward drawing for balanced trees

In this section, we show that our drawing strategy works well for drawing some classes of bounded-degree balanced search trees under order-preserving (strictly) upward drawing standard. The classes include k-balanced trees, red–black trees [20], BB[α]-trees [20] and (a,b)-trees [21], where k, α, a and b are fixed constants, and $2 \leqslant a \leqslant b$.

Our algorithm produces an order-preserving upward drawing with $\mathrm{O}(n \log\log n)$ area; if the strictly upward standard is required, then the drawing area increases to $\mathrm{O}(n(\log\log n)^2)$. Actually, a linear-area order-preserving strictly upward drawing algorithm for such trees has been already known in [8,9]. However, as stated in [8], the constant hidden in the area function is quite large (in order of thousands) except AVL trees which can be drawn with a constant not greater than 36 [9]. In that sense, our algorithm produces a better drawing with a relatively small constant (not greater than 64 in most practical cases) for all those classes of balanced trees.

In general, according to how to maintain the balance condition, balanced search trees are classified into three categories [22]: *height-balanced, weight-balanced* and *degree-balanced*. k-balanced trees and red–black trees are height-balanced since the tree-heights of the subtrees of each individual vertex may differ at most by a constant. In BB[α]-trees, the sizes of the subtrees of each vertex may differ at most by a constant, thus called weight-balanced. The (a,b)-tree is degree-balanced since its balance is maintained by varying the degree of internal vertices.

Let T be a tree with n vertices in the classes. Our partition of T into $\mathrm{O}(n/m)$ fragments (refer to Section 3) has two important properties:

(P1) the tree-height of each non-trivial fragment is $\mathrm{O}(\log m)$, and
(P2) for any path from the root to a leaf in FT the number of non-trivial fragments on the path is also $\mathrm{O}(\log m)$.

These two properties will be crucial for achieving area less than $\mathrm{O}(n \log n)$. To show (P1) and (P2), it suffices to prove that all vertices of height greater than $\mathrm{O}(\log m)$ in T are m-critical, that is, any non-critical vertex is of height at most $\mathrm{O}(\log m)$ in T.

A binary tree T is *k-balanced* if, for each vertex v of T, the tree-height of its left subtree and the tree-height of its right subtree differ by at most k, where k ($\geqslant 1$) is a fixed constant. A k-balanced tree is a natural generalization of AVL trees, i.e., 1-balanced trees. Let F_h^k be the minimum number of vertices of a k-balanced tree with height h. For $k = 1$, $F_h^1 = F_{h-1}^1 + F_{h-2}^1 + 1$, where $F_0^1 = 0$ and $F_1^1 = 1$. Note that $F_h^1 \geqslant \phi\sqrt{5}$, where $\phi = (1+\sqrt{5})/2$ [14]. By generalizing the above recurrence to k-balanced trees, $F_h^k = F_{h-1}^k + F_{h-k-1}^k + 1$, where $F_i^k = i$ for $0 \leqslant i \leqslant h$. Then we can easily show that $F_h^k \geqslant F_{\lfloor h/k \rfloor}^1$ by induction on h.

Lemma 6. *In the partition of a k-balanced binary tree T by* PartitionTree(T, m)*, every non-critical vertex has height at most* $\mathrm{O}(\log m)$ *in T.*

Proof. Consider a vertex v in T with height $k\log_\phi(5m) + k + 1$. We know that v must have two children because of the k-balanced condition. Let u and w be two children of v. Then both of T_u and T_w must have height at least $k\log_\phi(5m)$, so they have size at least

$$F_{k\log_\phi(5m)}^k \geqslant F_{\log_\phi(5m)}^1 \geqslant \frac{\phi^{(\log_\phi(5m))}}{\sqrt{5}} = \frac{5}{\sqrt{5}}m > 2m.$$

This implies that each of them contains at least one m-critical vertex. Since v is the least common ancestor of u and w, v becomes an m-critical vertex. It is also easy to see that any vertex with height $> k\log_\phi(5m) + k + 1$ is m-critical. Therefore, every non-critical vertex has $\mathrm{O}(\log m)$ height in T. □

For red–black trees, BB$[\alpha]$-trees and (a, b)-trees, we can prove that (P1) and (P2) also hold using a similar argument to that in Lemma 6. Hence we have the following result.

Lemma 7. *In the partition for trees in the classes of k-balanced trees, red–black trees, BB$[\alpha]$-trees and (a, b)-trees,* (P1) *each fragment has* $\mathrm{O}(\log m)$ *tree-height, and* (P2) *there are at most* $\mathrm{O}(\log m)$ *non-trivial fragments on any path from the root to a leaf in FT.*

After calling PartitionTree(T, m) with $m = \log n$, draw each non-trivial fragment $F \in FT$ by the following base drawing algorithm.

1. Draw F by A_h of Lemma 1. To preserve the left-to-right order in Δ_F, we skip step 1 of $\mathcal{A}_h$, in which F is transformed into a right-heavy tree. Since F has height $\mathrm{O}(\log\log n)$ by property (P1), it still holds that $H_F = \mathrm{O}(\log\log n)$ and $W_F = \mathrm{O}(\log n)$.
2. Let F' be the unique child fragment of F if exists. Let $s = (c_F, r_{F'})$ be the separator between F and F'. We assume that c_F has k children $b_1, b_2, \ldots, b_k$, left-to-right in F. Note that $r_{F'}$ is a child of c_F in T, but it is not b_i for any i. We denote by $e_i = (c_F, b_i)$ the ith child edge of c_F. For the completeness of the description, we assume that there exist two virtual edges e_0 and e_{k+1} to the left of e_1 and to the right of e_k, respectively. Suppose that the separator s is an edge between e_t and e_{t+1} for some t $(0 \leqslant t \leqslant k)$. We modify Δ_F to make enough room to draw s between e_t and e_{t+1} as shown in Fig. 8. Let d_h be the distance between c_F and the right side of Δ_F and d_v be the distance between c_F and the bottom side of Δ_F. To assist a description of the modification, we define two horizontal lines l_1 and l_2 and two vertical lines l_3 and l_4 as illustrated in Fig. 8. Let Δ_{b_i} be a drawing of the subtree rooted at b_i. First, we horizontally translate drawings $\Delta_{b_{t+1}}, \ldots, \Delta_{b_k}$ to the rightward direction so that the drawings entirely lie between $x = x'$ and $x = x' + d_h$. The translation does not violate drawing standards of Δ_F. Second, we translate drawings $\Delta_{b_1}, \ldots, \Delta_{b_t}$ to the downward direction so that the top sides of the drawings are aligned at l_1. Then the translated drawings lie entirely between $y = y'$ and $y = y' - d_v$. We suppose hereafter that the modified drawing is defined in a rectangle with height $H_F + d_v$ and width $W_F + d_h + 1$. That is, the modified drawing has height $\leqslant 2H_F = \mathrm{O}(\log\log n)$ and width $\leqslant 2W_F + 1 = \mathrm{O}(\log n)$.

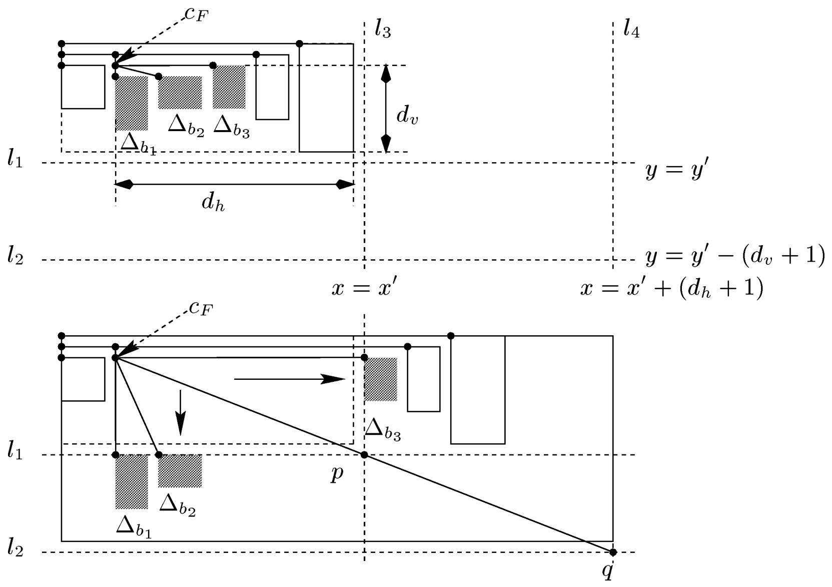

Fig. 8. An illustration of step 2. Suppose that a separator $(c_F, r_{F'})$ is an edge between (c_F, b_2) and (c_F, b_3). The above is a drawing of a non-trivial fragment F by $\mathcal{A}_h$. The below is a drawing modified so as to draw the separator $(c_F, r_{F'})$.

Let p be an intersection point of l_1 and l_3, and q be an intersection point of l_2 and l_4. Then a line passing through c_F and p passes q, and the line does not intersect other vertices and edges in Δ_F. Using this fact, we, in the merging step, place the root $r_{F'}$ at the point q, and then draw the separator $s = (c_F, r_{F'})$ as a straight-line from c_F to $q (= r_{F'})$ without crossings.

To merge Δ_F's, we use a slightly modified version of $\mathcal{M}_v$. We need two types of drawings for F, denoted by Δ_F^A and Δ_F^B. Drawing Δ_F^A is identical to Δ_F produced by the above base drawing algorithm. In Δ_F^A, r_F lies at the upper-left corner. Symmetrically, we can obtain an order-preserving drawing Δ_F^B such that r_F lies at the upper-right corner of the drawing. Note that Δ_F^B is obtained by horizontally flipping Δ_F^A so that the left-to-right order is preserved. With Δ_F^A's and Δ_F^B's in our hand, we apply $\mathcal{M}_v$ as maintaining the order-preserving property. The detail is straightforward.

We know that $\mathrm{NL}(FT) = \mathrm{O}(\log n)$ because T is balanced. Using this fact and Lemma 5, we can show the following theorem.

Theorem 3. *For a balanced tree T in the classes of k-balanced trees, red–black trees, $BB[\alpha]$-trees and (a, b)-trees, the above algorithm produces an* $\mathrm{O}(n \log\log n)$*-area order-preserving upward straight-line drawing with height* $\mathrm{O}(n \log\log n / \log n)$ *and width* $\mathrm{O}(\log n)$.

In the remainder of the section, we will show that the order-preserving upward drawing described above can be easily modified into an order-preserving strictly upward drawing. First, we construct an $\mathrm{O}(n(\log\log n)^2)$-area order-preserving upward drawing in which every edge from a parent to a child in the drawing has always one of south, east and southeast directions. Second, using the property, we transform the upward drawing into a strictly upward one without affecting area $\mathrm{O}(n(\log\log n)^2)$.

Drawings of the fragments produced by the base drawing algorithm satisfy the property clearly. In the merging step, however, some of the drawings are horizontally flipped, so edges may have west or southwest directions. This can be solved by modifying the merging algorithm as follows. Let F be the root fragment of FT with child fragments $F_1, F_2, \ldots, F_k$, left-to-right. Let FT_i be the subtree rooted at F_i in FT. Recursively construct the drawings for FT_i. If F is trivial, we combine Δ_F and Δ_{FT_i}'s in the same method as we did in $\mathcal{M}_v$ (refer to Fig. 3(b)). If F is a non-trivial fragment with one child fragment F_1, place Δ_{FT_1} so that the root of Δ_{FT_1} is at the point q and Δ_{FT_1} lies entirely in the right side of Δ_F. Note here that Δ_{FT_1} is not flipped. Then the separator between F and F_1 can be drawn as a straight-line with a southeast direction.

Let Δ_T be a drawing obtained by the above merging algorithm. It is clear that $H_T = \mathrm{O}(n \log\log n / \log n)$. Consider the width W_T. Let P be a path from the root to a leaf v of Δ_T. Suppose that P is passing through fragments $F_1, F_2, \ldots, F_h$, where $v \in F_h$ and F_i is a child fragment of F_{i-1} for $2 \leqslant i \leqslant h$. If F_i is trivial, we know that a separator from r_{F_i} to its child $r_{F_{i+1}}$ contributes to W_T by at most one. If F_i is non-trivial, a path from r_{F_i} to $r_{F_{i+1}}$ contributes to W_T by the width of Δ_{F_i}, i.e., $\mathrm{O}(\log n)$. Let n_1 and n_2 be the number of trivial fragments and non-trivial fragments on P, respectively. Then $W_T = n_1 + n_2 \times \mathrm{O}(\log n)$. Since $n_1 = \mathrm{O}(\log n)$ because T is balanced and $n_2 = \mathrm{O}(\log\log n)$ by (P2) of Lemma 7. Therefore, $W_T = \mathrm{O}(\log n) + \mathrm{O}(\log\log n \times \log n) = \mathrm{O}(\log n \log\log n)$.

Now it remains to transform the upward drawing to a strictly upward one. The transformation is done by modifying coordinates (x, y) of each vertex in Δ_T to $(x, x + y)$. Using the property on the edge directions of Δ_T, it is not difficult to show that the transformed drawing is planar and strictly upward [7]. Then the transformed drawing has height $\leqslant H_T + W_T$ and width W_T. Since $H_T \geqslant W_T$, the height of the drawing is bounded below by $2H_T$. Consequently, the strictly upward drawing has asymptotically the same bounds for height and width as the upward ones.

Theorem 4. *For a balanced tree T in the classes of k-balanced trees, red–black trees, BB$[\alpha]$-trees and (a, b)-trees, the above algorithm produces an $\mathrm{O}(n(\log\log n)^2)$-area order-preserving strictly upward straight-line drawing with height $\mathrm{O}(n \log\log n/\log n)$ and width $\mathrm{O}(\log n \log\log n)$.*

Remark. The multiplicative constants hidden in our area functions are not large as compared to those in the linear-area drawing algorithm [8,9], in which the constant for AVL trees turned out to be not greater than 36 for strictly upward standard (not analyzed for the other classes of balanced trees), and it decreased to 3.1 in experiments. Roughly analyzing the values in our order-preserving strictly upward drawing, the values for k-balanced, red–black, BB$[\alpha]$- and (a, b)-trees are no more than $32k/\log\phi$, 64, $-32/\log(1-\alpha)$ and $16b$, respectively. In practice, k and b are sufficiently small; 46.1 for AVL (1-balanced) trees, 64 for red–black trees, and 48 for $(2, 3)$-trees. Thus the constants are no more than 64 in most cases. The experiment for red–black trees in Section 7 shows that the value is actually no more than 1.2 in most cases. We might expect the constant values for the other balanced trees including AVL trees should drastically decrease in experiments.

6. Orthogonal straight-line drawing for binary trees

The main contribution in this section is algorithms which draw binary trees with $\mathrm{O}(n \log\log n)$ area, (or $\mathrm{O}(n \log\log n)$ volume in 3-dimension) and any given aspect ratio under the orthogonal straight-line standard. The algorithms are the first results in 2- and 3-dimensions which reduce the upper bound from $O(n \log n)$ to $O(n \log\log n)$. Furthermore, the algorithms can draw binary trees with any given aspect ratio. We will concentrate on a 2-dimensional drawing algorithm because a 3-dimensional drawing can be directly obtained by extending the 2-dimensional algorithm.

We use a variation of the procedure PartitionTree as a partitioning method $\mathcal{P}$. Let T be a binary tree of n ($\geqslant 2$) vertices. Suppose that a vertex $\mu_T \in T$ is not the root of T and has at most one child in T. Given a partition parameter m ($\leqslant n$) and the vertex μ_T, we will partition T into $O(n/m)$ fragments, each of size $O(m)$, so that

(i) μ_T itself constitutes a trivial fragment, and
(ii) $r_F \neq c_F$ for every non-trivial fragment F.

If F has no connection vertices, we pick an arbitrary leaf in F as c_F. Moreover, Lemma 4 must hold for this partitioning method.

Such a partitioning method can be achieved as follows. Let ν be the child of μ_T in F (if it exists) and F_ν be the subtree rooted at ν in F. To make μ_T an m-critical vertex, i.e., a trivial fragment, we attach a dummy tree D as the subtree of μ_T. Now μ_T has at most two subtrees F_ν and D. The size of D is determined as follows: if $\text{size}(F_\nu) \leqslant 2m$, then $\text{size}(D) = 2m - \text{size}(F_\nu)$, otherwise $\text{size}(D) = m$. Then μ_T will become an m-critical vertex by the definition of the m-critical vertex. Note that $\text{size}(D) \leqslant 2m$, so $\text{size}(T) \leqslant n + 2m \leqslant 3n$. We now partition T by calling PartitionTree(T, m). The number of fragments is still $O(n/m)$ by Lemma 3, and μ_T is defined as an m-critical vertex. Thus, condition (i) is satisfied. Let F be a non-trivial fragment such that $r_F = c_F$. Since $\text{size}(F) \geqslant 2$, r_F has exactly one child w in F. To satisfy condition (ii), we define r_F as a trivial fragment and the new non-trivial fragment F_w as a non-trivial fragment. Here, F_w represents a subtree rooted at w in F. Thus, condition (ii) is satisfied. Since F has only one connection vertex c_F ($= r_F$), F_w does not have a connection vertex any more. This implies that Lemma 4 still holds. Finally, we delete D from T. Consequently, T is divided into $O(n/m)$ fragments satisfying two conditions and Lemma 4. We shall refer to this partitioning method as Procedure PartitionTree*(T, m, μ_T).

Before describing our drawing algorithm, let us explain a basic idea to control the aspect ratio of Δ_T while achieving $O(n \log\log n)$ area. To do it, we shall apply our partition-and-merge method twice. First, partition T with a partition parameter m_1 ($\leqslant n$). Suppose that we have drawn each fragment by some drawing method such as $\mathcal{A}_h$ and vertically stack the fragment drawings by $\mathcal{M}_v$. If m_1 is very smaller than n, e.g., $m_1 = \log n$, then the final drawing becomes longish because the height is relatively larger than the width. If m_1 is larger than $O(\log n)$, then the shape of drawing will be square-like more and more, but the drawing area may be greater than $O(n \log\log n)$; in fact, the area becomes $O(n \log n)$ if we use either $\mathcal{A}_h$ or $\mathcal{A}_v$ to draw the fragments. In order to control the aspect ratio as keeping the area $O(n \log\log n)$, we draw each fragment by another drawing method based on the partition-and-merge strategy, rather than by $\mathcal{A}_h$ and $\mathcal{A}_v$. At this time, each fragment is again partitioned with another partition parameter m_2 ($\leqslant m_1$) into several sub-fragments. The sub-fragments are drawn by some base drawing algorithm, denoted here by $\mathcal{B}^*$, and their drawings are horizontally arranged by $\mathcal{M}_h$. Then we can keep the area $O(n \log\log n)$ and can also control the aspect ratio according to the values of m_1 and m_2.

The base drawing algorithm $\mathcal{B}^*$ draws each non-trivial fragment F of size m in $O(m \log m)$ area such that r_F is placed at the upper-left corner of Δ_F and c_F is at the upper-right corner of ΔF. We can obtain $\mathcal{B}^*$ through a minor modification of $\mathcal{A}_v$ of Lemma 2 as follows.

1. Let $P = (r_F = v_1, v_2, \ldots, v_{h-1}, v_h = c_F)$ be a path from r_F to c_F in F. Let F_i be the subtree of v_i attached to the path P not containing v_{i+1}.

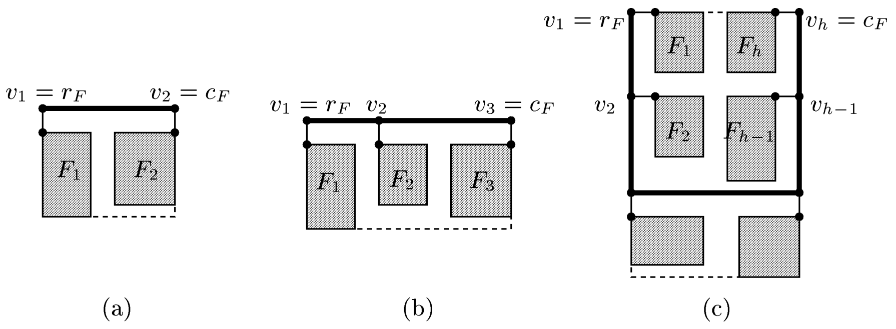

Fig. 9. (a), (b) When the length of P is less than 4, horizontally arrange Δ_{F_i}'s left-to-right. Put v_i directly above the root of Δ_{F_i}. (c) Otherwise, stack Δ_{F_i}'s up-to-down and down-to-up such that the shape of the path P is like a character "U".

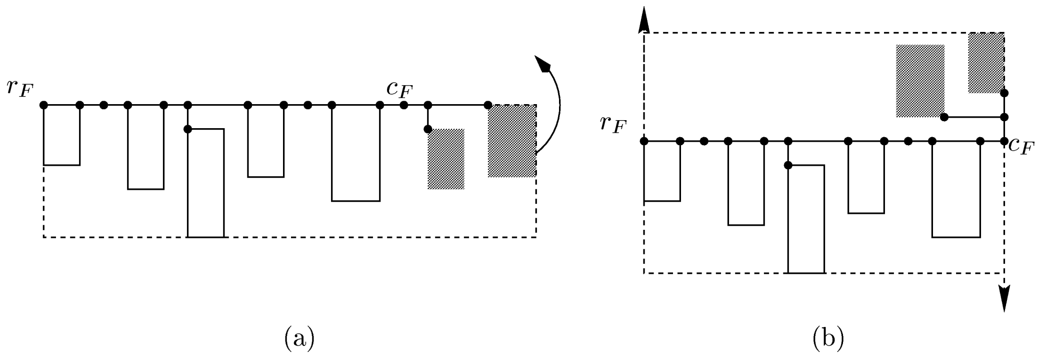

Fig. 10. Modification of Δ_F so that c_F is south-open and lies on the left side.

2. Draw F_i by $\mathcal{A}_v$ for each i so that $H_{F_i} = \mathrm{O}(\mathrm{size}(F_i))$ and $W_{F_i} = \mathrm{O}(\log(\mathrm{size}(F_i)))$.
3. Arrange Δ_{F_i}'s and vertices on P, as shown in Fig. 9, according to the length of P; horizontally flip some Δ_{F_i}'s, if necessary. Next, draw edges on P and adjacent to P.

Trivially, $H_F = \mathrm{O}(m)$. Since at most three Δ_{F_i}'s are horizontally laid, $W_F = \mathrm{O}(\log m)$.

Lemma 8. *For a non-trivial fragment F of m vertices, the procedure $\mathcal{B}^*$ produces an orthogonal straight-line drawing Δ_F with height $\mathrm{O}(m)$, width $\mathrm{O}(\log m)$ and area $\mathrm{O}(m \log m)$. Two vertices r_F and c_F are placed at the upper-left and upper-right corners of Δ_F, respectively.*

Now we are ready to give the whole drawing algorithm. The algorithm takes as input a binary tree T, two integers m_1 and m_2 ($n \geqslant m_1 > m_2$).

1. Partition T by PartitionTree(T, m_1). Let FT be the fragment tree of T. Note here that the procedure PartitionTree is used as $\mathcal{P}$.
2. For each non-trivial fragment $F \in FT$, do the following steps.
   (i) Partition F by PartitionTree$^*(F, m_2, c_F)$. Let FT' be the fragment tree of F. Note that $\mu_F = c_F$, so c_F is trivial in FT'. Then FT' consists of $\mathrm{O}(m_1/m_2)$ fragments, each with $\mathrm{O}(m_2)$ vertices.

(ii) For each non-trivial fragment $F' \in FT'$, draw F' by the base algorithm $\mathcal{B}^*$. By Lemma 8, $H_{F'} = \mathrm{O}(m_2)$ and $W_{F'} = \mathrm{O}(\log m_2)$.
(iii) Let R' be the root fragment of FT' and let C' be the fragment of FT' containing c_F. Actually, C' consists of c_F only. Transform FT' so that a path from R' to C' in FT' becomes the rightmost path of FT', and all subtrees in FT' attached to the path into right-heavy trees.
(iv) Merge $\Delta_{F'}$'s by the merging algorithm $\mathcal{M}_h$; this is always possible because the root and connection vertex of F' satisfy the assumptions needed to apply $\mathcal{M}_h$.

Since the path from R' to C' is the rightmost path of FT', c_F is located on the top side of Δ_F (see Fig. 10(a)). In the step (iii), FT' is not a perfectly right-heavy tree because of the rightmost path from R' to C'. However, it is easy to show that $\mathrm{NR}(FT') \leqslant \lfloor \log \mathrm{size}(FT') \rfloor + 1$. Combining this fact with Lemma 5, we can know that Δ_F has height

$$H_F = \mathrm{O}\big(H_{F'} + \mathrm{NR}(FT')\big) = \mathrm{O}\left(m_2 + \log \frac{m_1}{m_2}\right)$$

and

$$W_F = \mathrm{O}\left(W_{F'} \frac{m_1}{m_2}\right) = \mathrm{O}\left(\frac{m_1 \log m_2}{m_2}\right).$$

3. Transform FT into a left-heavy tree.
4. Merge Δ_F's by the merging algorithm $\mathcal{M}_v$ as follows. To apply $\mathcal{M}_v$, r_F should lie on the left side of Δ_F and c_F should lie on the right side of Δ_F. Of course, r_F must be north-open and c_F be south-open. However, as shown in Fig. 10(a), Δ_F produced in step 2 may not satisfy the condition about c_F. We modify Δ_F as shown in Fig. 10(b). It is always possible because c_F has at most one subtree in F. Then r_F and c_F respectively lie on the left and right sides of Δ_F, and are north- and south-open in Δ_F. By Lemma 5, Δ_T has

$$H_T = \mathrm{O}\left(H_F \frac{n}{m_1}\right) = \mathrm{O}\left(\frac{nm_2}{m_1} + \frac{n}{m_1} \log \frac{m_1}{m_2}\right)$$

and

$$W_T = \mathrm{O}\big(W_F + \mathrm{NL}(FT)\big) = \mathrm{O}\left(\frac{m_1 \log m_2}{m_2} + \log \frac{n}{m_1}\right).$$

Theorem 5. *Given a binary tree T and $\mu_T \neq r_T$ with at most one child of T, the above algorithm produces an orthogonal straight-line drawing Δ_T with area* $\mathrm{O}(n \log\log n)$ *and any aspect ratio in the range of* $[\mathrm{O}(1), \mathrm{O}(n \log\log n / \log^2 n)]$.

Proof. Let a be a real number such that $0 < a < 1$. Setting $m_1 = \mathrm{O}(n^a \log n / \sqrt{\log\log n})$ and $m_2 = \mathrm{O}(\log n)$,

$$H_T = \mathrm{O}\left(\frac{nm_2}{m_1} + \frac{n}{m_1} \log \frac{m_1}{m_2}\right) = \mathrm{O}\left(\frac{n\sqrt{\log\log n}}{n^a} + \frac{n\sqrt{\log\log n}}{n^a \log n} \log n\right) = \mathrm{O}\big(n^{1-a}\sqrt{\log\log n}\,\big),$$
$$W_T = \mathrm{O}\left(\log \frac{n}{m_1} + \frac{m_1}{m_2} \log m_2\right) = \mathrm{O}\left(\log n^{1-a} + \frac{n^a}{\sqrt{\log\log n}} \log\log n\right) = \mathrm{O}\big(n^a\sqrt{\log\log n}\,\big).$$

The area of Δ_T is $H_T \times W_T = \mathrm{O}(n \log\log n)$. Without loss of generality, H_T is assumed to be no less than W_T, that is, $0 < a \leqslant 1/2$. Then the aspect ratio is $H_T / W_T = \mathrm{O}(n^{1-2a})$. This implies that for any constant

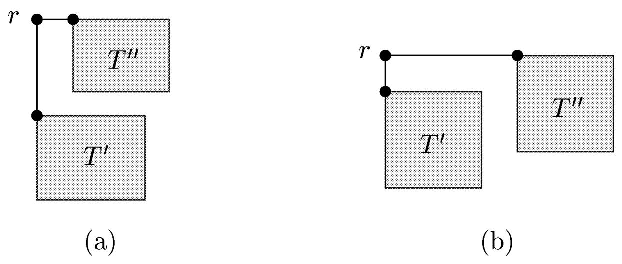

Fig. 11. Arrangements for h–v drawing.

$0 \leqslant \varepsilon < 1$ we can obtain a drawing with aspect ratio $O(n^{\varepsilon})$. Leiserson [19] showed that the minimum side length of an orthogonal drawing must be $\Omega(\log n)$. Using the fact, we can see that the possible maximum value of the aspect ratio of $O(n \log\log n)$-area drawings is $O(n \log\log n/\log^2 n)$. When $m_1 = n/k$ and $m_2 = \log n$ for some constant $k > 1$, then $H_T = O(\log n)$ and $W_T = O(n \log\log n/\log n)$. As a consequence, we can achieve Δ_T with any aspect ratio in a range between the possible smallest and largest values, i.e., $[O(1), O(n \log\log n/\log^2 n)]$. $\square$

To get a 3-dimensional orthogonal straight-line drawing with any given aspect ratio, we apply the partition-and-merge method three times. Partition a binary tree T with parameter $m_0 \leqslant n$, and draw each fragment with a proper aspect ratio on xy-plane. Here, we use Theorem 5 to draw each fragment, in which the partition-and-merge process occurs twice again. Next, stack Δ_F's along the z-axis in a similar way to $\mathcal{M}_v$, and draw separators.

Theorem 6. *Any binary tree T with n vertices has a 3-dimensional orthogonal straight-line drawing Δ_T with volume $O(n \log\log n)$ and any aspect ratio in $[O(1), O(n \log\log n/\log n)]$.*

7. Experimental results

We have implemented most algorithms presented in this paper on SGI machines in C language and have experimented to see how large the multiplicative constants in the area functions are. In particular, we have focused on the $O(n \log\log n)$-area upward drawing algorithm of Theorem 1 and the $O(n(\log\log n)^2)$-area order-preserving strictly upward drawing algorithm of Theorem 4 for red–black trees.

For the upward drawing algorithm of Theorem 1, we have prepared three sets of binary trees; randomly generated unordered rooted trees (in short, the collection BT) which is generated by an algorithm presented in [1], randomly built binary search trees (in short, the collection BST) which is generated by inserting keys, starting from an empty binary search tree, one by one from n-size random permutation [6], and randomly built red–black trees (in short, the collection RBT) [6]. For each set, we have generated 40 binary trees with the same number of vertices which ranges from 100 to 10^5. Clearly, the trees in BT are likely to be more unbalanced than those in BST and in RBT, and the trees of RBT are most balanced.

We have first checked how large the constants, for each set of trees, in the area function are. Fig. 12 shows the ratios of the actual drawing areas obtained by Theorem 1 to the value of $n \log\log n$. The

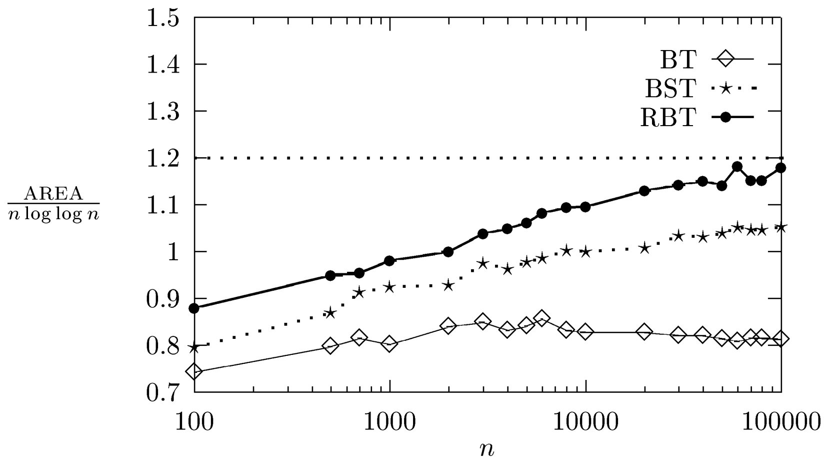

Fig. 12. Constant values in the area function of the upward drawing algorithm (Theorem 1) for three sets, BT, BST and RBT, of trees.

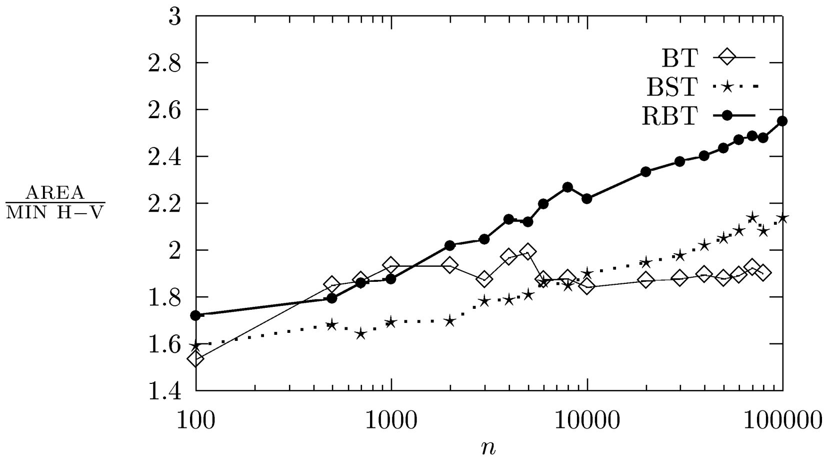

Fig. 13. Ratios of the upward drawing area (Theorem 1) to the minimum area of h–v drawing for BT, BST and RBT.

constant values for all three sets do not exceed 1.2, and are quite smaller than the theoretical bound 12 (refer to the remark in Section 4).

We have next compared the drawing area with the minimum area of h–v drawing. An *h–v drawing* of T, consisting of root r and two subtrees T' and T'', is obtained by placing r at the upper left-hand corner and by connecting, in one of two ways shown in Fig. 11, r and the recursively drawn h–v drawings of T' and T'' by horizontal or vertical line segments. Eades et al. [11] proposed an algorithm of computing the minimum area of an h–v drawing for any n-vertex binary tree in $O(n^{3/2} \log n)$ time by employing dynamic programming techniques.

Actually, an upward drawing obtained by Theorem 1 is not an h–v drawing. Thus one may have a doubt about the fairness of the comparison. But we believe that the h–v drawing algorithm is the best

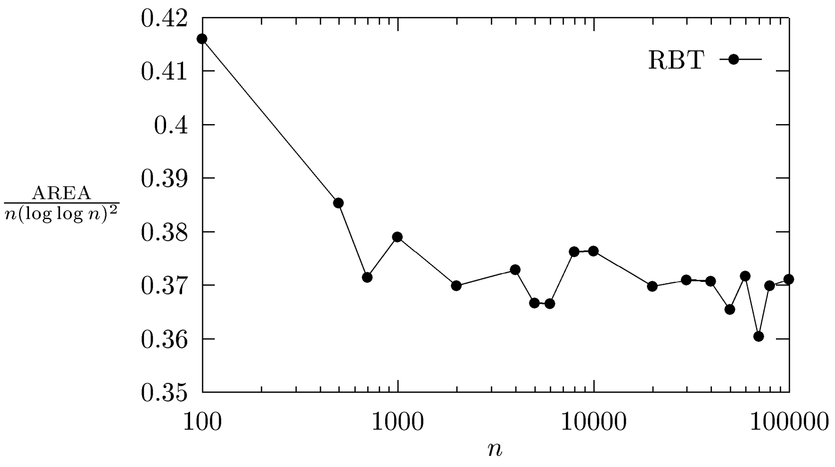

Fig. 14. Constant values in the area function of the order-preserving strictly upward drawing algorithm (Theorem 4) for a set, RBT, of red–black trees.

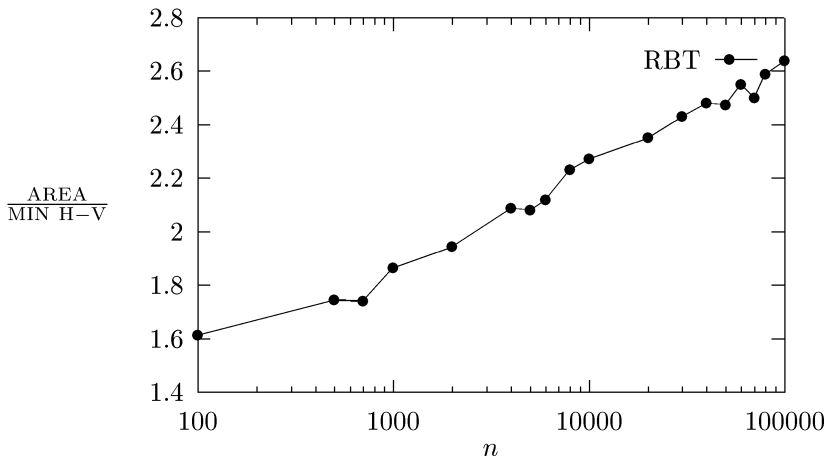

Fig. 15. Ratios of the order-preserving strictly upward drawing area (Theorem 4) to the minimum area of the order-preserving h–v drawing for RBT.

upward drawing algorithm known so far. Fig. 13 illustrates the area ratio of two algorithms, and shows that our algorithm produces an upward drawing with area less than three times the minimum h–v area in most cases.

Next, we have considered order-preserving strictly upward drawings for randomly built red–black trees, RBT. As shown in Figs. 14 and 15, the constant value for red–black trees in the drawings obtained by Theorem 4 is not greater than 0.42, and the area is not larger than three times the minimum area of the order-preserving h–v drawing. Notice that the theoretical constant value for red–black trees is 64 (refer to the remark in Section 5).

8. Concluding remarks

We have investigated several straight-line drawing problems for some classes of bounded-degree trees under various drawing standards: (1) upward straight-line, (2) order-preserving (strictly) upward straight-line, and (3) non-upward orthogonal straight-line standards. Main contribution of this paper is a unified drawing framework which allows us to obtain various tree drawings with area strictly less than the best bound $O(n \log n)$ [7,12] known so far. We also show that arbitrary aspect ratio can be achieved in non-upward orthogonal straight-line drawings of binary trees. Table 1 is a summary of the results.

Let us discuss what other drawing problems can be solved by a unified drawing framework presented in this paper.

- We believe that the unified framework is useful to achieve an orthogonal drawing with the small area and any given aspect ratio. As a representative example, we have already presented an orthogonal straight-line drawing algorithm in Section 6. If we combine the Valiant's result [28] and our framework, we can also draw any tree with maximum degree of 4 with area $O(n)$, any aspect ratio and $O(\log \log n)$ bends per edge. This drawing is superior to that of Valiant [28] whose area is $O(n)$, aspect ratio is $O(1)$ and the number of bends per edge is $O(\log n)$. The details are given in [25]. If the algorithm extends to 3-dimension, we can obtain a 3-dimensional orthogonal drawing for any tree with maximum degree of 6 so that the volume is $O(n)$, the number of bends per edge is at most seven, and almost any aspect ratio is allowed.
- Until now, the upward standard has been considered in 2-dimensional drawings. But, we can also define the upwardness in 3-dimensional drawings similarly. That is, if each vertex and its child in a drawing can be connected by a monotone curve with respect to an axis, the drawing is said to be upward with respect to the axis. Thus it will be interesting to characterize the area-tradeof between upwardness and aspect ratio in 2- and 3-dimensions. For instance, we can show that using the framework used in this paper, any binary tree with height $O(\log n)$ has a two-axis upward orthogonal straight-line drawing with volume $O(n \log \log n)$ and arbitrary aspect ratio in 3-dimension. But, we do not know whether or not it is possible in 2-dimension.
- In [8], the authors presented a strictly upward straight-line drawing algorithm for balanced search trees which guarantees area $O(n)$ and arbitrary aspect ratio with the shorter side length of at least $\log^{\alpha} n$ for any $\alpha > 1$. But it is impossible to get a longish drawing with shorter side length of $O(\log n)$, so it remains as an open problem. We can obtain such a longish drawing by our framework. The detail for AVL trees is given in [17].

We will finish this section with a list of open problems. For upward straight-line drawings, we do not know yet if any unbounded-degree tree can be drawn in area $O(n \log \log n)$, or if there exists a class of trees requiring $\Omega(n \log \log n)$, hence both remain open. For order-preserving upward straight-line drawings, it was proved that a class of logarithmic trees, in which the height of any subtree is logarithmic to the number of vertices, can be drawn in linear area [8], but it remains open if any tree of height $O(\log n)$ can be drawn in linear area. Recently, Chan [3] developed an algorithm to draw any binary tree in $O(n^{1+\varepsilon})$ area with stronger order-preserving standard, called *strongly order-preserving standard*, in which each edge from the parent to the left (respectively right) child should be monotone decreasing (respectively increasing) in the x-direction. Thus it would be interesting to draw any (binary) tree in super-linear area under (strongly) order-preserving (strictly) upward straight-line standards, together with raising the lower bound, $\Omega(n \log n)$ [12].

Acknowledgements

We thank to anonymous referees for many useful suggestions that have improved the presentation of this paper.

References

[1] L. Alonso, R. Schott, Random Generation of Trees, Kluwer Academic Publishers, Dordrecht, 1995.
[2] S.N. Bhatt, F.T. Leighton, A framework for solving VLSI graph layout problems, J. Comput. Syst. Sci. 28 (1984) 300–343.
[3] T. Chan, A near-linear area bound for drawing binary trees, in: Proc. 10th ACM–SIAM Symp. on Discrete Algorithms, 1999.
[4] T. Chan, M.T. Goodrich, S.R. Kosaraju, R. Tamassia, Optimizing area and aspect ratio in straight-line orthogonal tree drawings, in: S. North (Ed.), Graph Drawing (Proc. GD ’96), Lecture Notes in Computer Science, Vol. 1353, Springer, Berlin, 1997.
[5] R.F. Cohen, P. Eades, T. Lin, F. Ruskey, Three-dimensional graph drawing, in: R. Tamassia, I.G. Tollis (Eds.), Proc. Graph Drawing: DIMACS International Workshop, GD’94, Lecture Notes in Computer Science, Vol. 894, Springer, Berlin, 1994, pp. 1–11.
[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.
[7] P. Crescenzi, G. Di Battista, A. Piperno, A note on optimal area algorithms for upward drawings of binary trees, Computational Geometry 2 (1992) 187–200.
[8] P. Crescenzi, P. Penna, Strictly-upward drawings of ordered search trees, Theoret. Comput. Sci. 203 (1998) 51–67.
[9] P. Crescenzi, P. Penna, A. Piperno, Linear area upward drawings of AVL trees, Computational Geometry 9 (1998) 25–42 (Special Issue on Geometric Representations of Graphs, edited by G. Di Battista and R. Tamassia).
[10] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, Algorithms for drawing graphs: an annotated bibliography, Computational Geometry 4 (1994) 235–282.
[11] P. Eades, T. Lin, X. Lin, Minimum size h–v drawings, in: Proc. Advanced Visual Interfaces, World Scientific Series in Computer Science, Vol. 36, 1992, pp. 386–394.
[12] A. Garg, M.T. Goodrich, R. Tamassia, Area-efficient upward tree drawings, in: Proc. 9th Ann. ACM Symp. Comput. Geom., 1993, pp. 359–368.
[13] H. Gazit, G.L. Miller, S.-H. Teng, Optimal tree contraction in an EREW model, in: S.K. Tewksbury, B.W. Dickinson, S.C. Schwartz (Eds.), Concurrent Computations: Algorithms, Architecture and Technology, Plenum Press, New York, 1988, pp. 139–156.
[14] R.L. Graham, D.E. Knuth, O. Patashnik, Concrete Mathematics, Addison-Wesley, Reading, MA, 1989.
[15] L.J. Guibas, J. Hershberger, D. Leven, M. Sharir, R.E. Tarjan, A linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons, Algorithmica 2 (1987) 209–233.
[16] S.K. Kim, Simple algorithms for orthogonal upward drawings of binary and ternary trees, in: Proc. 7th Canadian Conference on Computational Geometry, 1995, pp. 115–120.
[17] S.K. Kim, Logarithmic width, linear area upward drawing of AVL trees, in: Inform. Process. Lett. 63 (1997) 303–307.
[18] F.T. Leighton, A. Rosenberg, 3D circuit layouts, SIAM J. Comput. 15 (1986) 793–813.
[19] C.E. Leiserson, Area Efficient VLSI Computation, MIT Press, Cambridge, MA, 1983.
[20] R.J. Lipton, R.E. Tarjan, Applications of a planar separator theorem, SIAM J. Comput. 9 (1980) 615–627.
[21] K. Mehlhorn, Data Structures and Algorithms 1: Sorting and Searching, Springer, Berlin, 1984.

[22] M.H. Overmars, The Design of Dynamic Data Structures, Lecture Notes in Computer Science, Vol. 156, Springer, Heidelberg, Germany, 1983.
[23] Y. Shiloach, Arrangements of planar graphs on the planar lattice, Ph.D. Thesis, Weizmann Institute of Science, 1976.
[24] C.-S. Shin, S.K. Kim, K.-Y. Chwa, Area-efficient algorithms for upward straight-line tree drawings, in: The 2nd International Computing and Combinatorics Conference (COCOON'96), Lecture Notes in Computer Science, Vol. 1090, Springer, Berlin, 1996, pp. 106–116.
[25] C.-S. Shin, S.K. Kim, K.-Y. Chwa, Algorithms for drawing binary trees in the plane, Inform. Process. Lett. 66 (3) (1998) 133–139.
[26] R. Tamassia, Graph drawing, in: J.E. Goodman, J. O'Rourke (Eds.), Handbook of Discrete and Computational Geometry, CRC Press, Boca Raton, FL, 1997, Chapter 44, pp. 815–832.
[27] J.D. Ullman, Computational Aspects of VLSI, Morgan Kaufmann, San Mateo, CA, 1992.
[28] L.G. Valiant, Universality considerations of VLSI circuits, IEEE Trans. Comput. 30 (12) (1981) 135–140.