



The University of Sydney

Parallel h-v Drawings of Binary Trees

Technical Report Number 480

March 1994

Panagiotis T Metaxas, Grammati E Pantziou
and Antonis Symvonis

ISBN 0 86758 911 6

**Basser Department of Computer Science
University of Sydney NSW 2006**

Parallel h-v Drawings of Binary Trees*

PANAGIOTIS T. METAXAS¹ GRAMMATI E. PANTZIOU²
ANTONIS SYMVONIS³

Technical Report 480

March 9, 1994

- (1) Department of Computer Science, Wellesley College, Wellesley MA 02181, USA
- (2) Department of Mathematics and Computer Science, Dartmouth College,
Hanover NH 03755, USA
- (3) Department of Computer Science, University of Sydney, N.S.W. 2006, Australia

Abstract

In this paper we present a method to obtain optimal h-v and inclusion drawings in parallel. Based on parallel tree contraction, our method computes optimal (with respect to a class of cost functions of the enclosing rectangle) drawings in $O(\log^2 n)$ parallel time by using a polynomial number of EREW processors. The number of processors reduces substantially when we study minimum area drawings. The method can be extended to compute optimal inclusion layouts in the case where each leaf l of the tree is represented by rectangle $l_x \times l_y$ (the dimensions of which are part of the input). For polynomial area layouts, our work places the problem of obtaining optimal size h-v or inclusion drawings in NC, presenting the first algorithm with polylogarithmic time complexity. Our method also yields an NC algorithm for the slicing floorplanning problem. Whether this problems was in NC was an open question [2].

1 Introduction

Drawing trees in a way that facilitates the understanding of the properties of the object being drawn is part of extensive research in the areas of visualization, computational geometry and documentation systems. In particular, rooted trees have been used to represent family trees, hierarchical structures and search trees.

In this paper we examine drawings of rooted binary trees. We study the h-v drawing convention studied by Eades, Lin and Lin [7] and Crescenzi, Di Battista and Piperno [3]. Our results extend to the inclusion convention [6], and to slicing floorplanning [9, 2].

*The work of the second author is partially supported by the EEC ESPRIT Basic Research Action No. 7141 (ALCOM II) and by the NSF postdoctoral fellowship No. CDA-9211155. Email: pmetaxas@lucy.wellesley.edu, pantziou@cs.dartmouth.edu, symvonis@cs.su.oz.au

The drawing of a rooted binary tree using the *h-v drawing convention* is a planar grid drawing in which tree nodes are represented as points (of integer coordinates) in the plane and tree edges as non-overlapping vertical or horizontal line segments. Moreover, each node is placed immediately to the right (same Y-coordinate) or immediately below (same X-coordinate) its parent and the drawings of subtrees rooted at nodes with the same parent are non-overlapping. Figure 1 shows three different h-v drawings of the same tree. The drawing in Figure 1.c differs from that of Figure 1.b in that the subtree rooted at node *c* is drawn above the subtree rooted at node *b*. The drawing in Figure 1.d differs from that of Figure 1.c in that the subtree rooted at node *b* is drawn to the right of its parent instead of below.

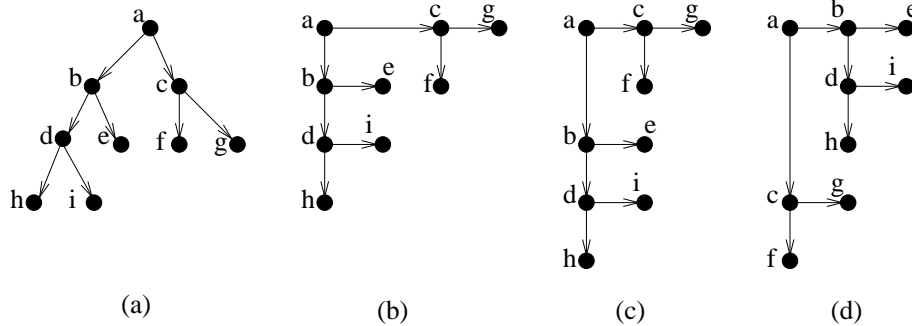


Figure 1: Examples of h-v drawings.

As it is evident, different h-v drawings of the same tree can be of different quality. The *quality* (or *cost*) is a function of the drawing. The most commonly used cost function is the area of the enclosing rectangle of the drawing. Other cost functions can be defined.

Eades et al. [7] showed how to compute in $O(n^2)$ time an optimal h-v drawing of a tree with n nodes with respect to a cost function $\psi(w, h)$ which is nondecreasing in both parameters w and h , where w and h are the width and the height of the enclosing rectangle of the drawing, respectively.

The same method can be used to develop optimal drawings (with respect to some cost function ψ) when the inclusion convention is adopted. In the *inclusion* convention, a node is represented by a rectangle and the parent-child relation by enclosing the rectangle which represents the child within that of the parent. Moreover, rectangles of nodes with the same parent are non-overlapping, next to each other (same X or Y coordinate of the top left corner) and in at least distance δ from each other. The rectangles representing the leaves are assumed to have sidelengths which are multiples of the “unit” of length.

The inclusion drawings of Figures 2.a and 2.b were obtained from the h-v drawings of Figures 1.b (or 1.c) and 1.d, respectively, by placing the node which is to the right of its parent in the h-v drawing on top of its sibling in the inclusion drawing. The inclusion drawing of Figure 2.c was obtained from the h-v drawing of Figure 1.b by placing it to the right of its sibling.

In this paper, we present a method that derives optimal drawings of rooted binary trees in parallel for both the h-v and the inclusion conventions. We choose to present our method for the h-v drawing convention since the drawings are still trees in their familiar “conventional”

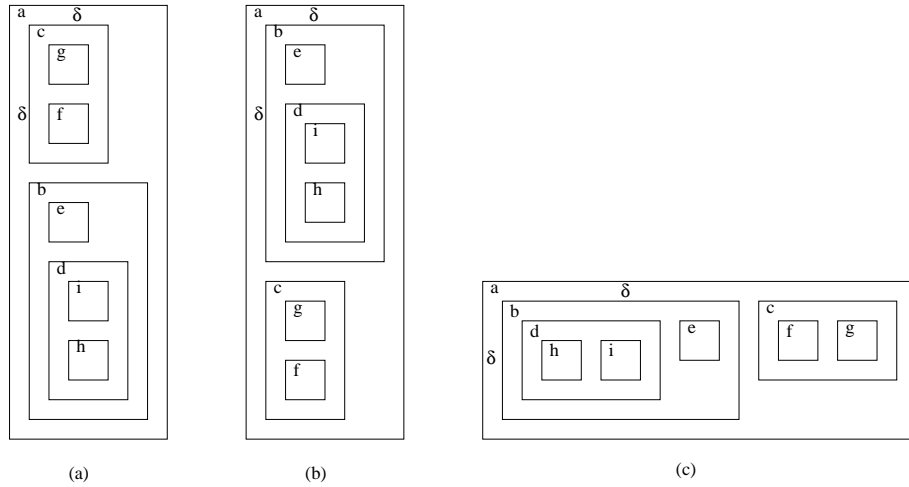


Figure 2: Examples of inclusion drawings.

form. Our method determines an optimal h-v drawing of a tree of n nodes in $O(\log^2 n)$ parallel time using $O(n^6/\log n)$ EREW processors. Even though the number of processors is too high for the method to be of practical interest, our work places the problem in the class NC. By realizing that h-v drawings can be easily converted to *upward* drawings, our method can be used for deriving (non optimal) upward drawings of binary trees in parallel. In the case that we want to minimize the area of the drawing, by using the fact that for a tree of n nodes there exists upwards layouts of area $n(\log n + 1)$ [3], we can reduce the number of processors to $O(n^4 \log n)$.

Even though there is no parallel algorithm for h-v or inclusion drawings, there exists a parallel algorithm for the closely related problem of slicing floorplanning. In *slicing floorplanning* we are given a regular binary tree (called *slicing tree*) in which leaves represent rectangular modules that are to be placed in the plane. Each internal node is an H or V node. In the final drawing, the modules are drawn as rectangles of the corresponding size but with a choice on their orientation (i.e., $x \times y$ or $y \times x$) and internal nodes by horizontal or vertical line segments. The drawing of the subtrees rooted at the children of a V-node (H-node) are drawn next to (on top of) each other. Alternatively, the drawing of the subtree rooted at a V-node (H-node) consists of two vertical (horizontal) slices, each next to (on top of) each other and containing the drawings of the subtrees rooted at its children. Figure 3 shows a slicing tree and a corresponding slicing floorplan.

Chen and Tollis [2] described how to parallelize the sequential algorithm given by Stockmeyer [9]. They derived an algorithm which solved the slicing floorplanning problem in $O(n)$ parallel time with $O(n)$ processors, where n is the number of leaves of the slicing tree. Interestingly enough, the number of processors used in their algorithm (instead of the parallel time) depends on the depth of the slicing tree. For slicing trees of $O(\log n)$ depth, they produce an optimal layout in $O(n)$ parallel time with $O(\log n)$ processors. However, it is easy to derive a PRAM algorithm to compute an optimal slicing floorplan with $O(n^3/\log n)$ processors in $O(d \log n)$ parallel time, where d is the depth of the slicing tree. Applying our method to the slicing floorplanning problem yields an $O(\log^2 n)$ PRAM algorithm which

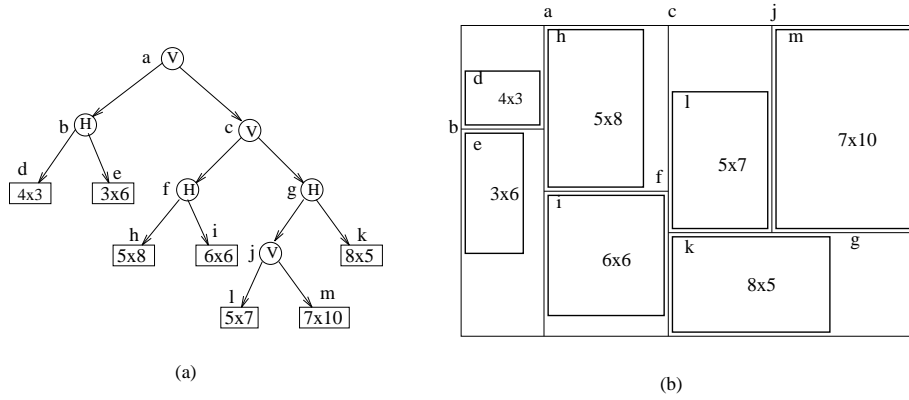


Figure 3: A Slicing tree and its corresponding slicing floorplan.

uses $O(L^6 / \log n)$ processors, where L is the sidelength of the floorplan of maximum enclosing rectangle. This proves that when $L = O(n^c)$, for some constant $c \geq 1$, the slicing floorplanning problem is in NC. The assumption $L = O(n^c)$ is a reasonable one since, otherwise, the layout would be of superpolynomial size.

The rest of the paper is organized as follows: In Section 2, we define the necessary terminology. In Section 3, we present our parallel algorithm for deriving optimal drawings for binary rooted trees using the h-v convention. We analyse it and prove its correctness. In Section 4, we study inclusion drawings and the slicing floorplanning problem. We conclude in Section 5.

2 Preliminaries

A *drawing* (or *layout*) Δ of a graph $G = (V, E)$ maps each node $v \in V$ to a point $P_v = (x_v, y_v)$ on the plane and each edge $(u, v) \in E$ to a simple Jordan curve with endpoints P_u and P_v . If all edges are mapped to straight-line segments, we have a *straight-line drawing*. Moreover, if all edges are line segments parallel to the X or Y axis, the drawing is called an *orthogonal straight-line drawing*. If all the nodes of G are mapped to points with integer coordinates, we have a *grid drawing*. When edges intersect only at common endpoints, the drawing is called *planar*. In this paper, we will study orthogonal straight-line planar grid drawings of rooted binary trees. Unless otherwise specified we will refer to them simply as drawings.

Given a graph $G = (V, E)$ and a drawing Δ of G on the plane, the drawing of any subgraph H of G resulting from Δ is called a *partial drawing* of H (with respect to Δ).

The *enclosing rectangle* of a drawing is the smallest rectangle with sides parallel to the axes which contains all points of the drawing. Let $X_{\max} = \max_{v \in V} \{x_v\}$, $X_{\min} = \min_{v \in V} \{x_v\}$, $Y_{\max} = \max_{v \in V} \{y_v\}$, $Y_{\min} = \min_{v \in V} \{y_v\}$. X_{\max} , X_{\min} , Y_{\max} and Y_{\min} completely define the enclosing rectangle of a drawing. Two rectangles are *overlapping* if they share at least a point of the plane. Otherwise, they are *non-overlapping* or *disjoint*. The *width* of a drawing is equal to $X_{\max} - X_{\min}$ while its *height* is equal to $Y_{\max} - Y_{\min}$. A drawing is *reduced* if

1. For all integers i such that $X_{\min} \leq i \leq X_{\max}$ there exists node $v \in V$ with $x_v = i$.
2. For all integers i such that $Y_{\min} \leq i \leq Y_{\max}$ there exists node $v \in V$ with $y_v = i$.

In the rest of the paper we will assume only reduced drawings. It is trivial to convert a non-reduced drawing to a reduced one.

A *rooted tree* $T = (V, E)$ is a weakly connected directed graph in which all nodes but the root are of indegree 1. The root has no incoming edges. If $(u, v) \in E$, we say that u is the *parent* of v or, equivalently, that v is the *child* of u . A node with no children is called a *leaf*. If $(u, v) \in E$ and $(u, w) \in E$, we say that v and w are *siblings*. We will use the notation $|T|$ to denote the number of nodes of tree T . If each node has at most two children the tree is called *binary*. If each node has either zero or two children we have a *regular binary* tree. If it is possible to reach node v from node u by following a directed path of the tree, we say that u is an *ancestor* of v or, equivalently, that v is a *descendant* of u . The *subtree rooted at* v , denoted T_v , consists of v , all of v 's descendants and the edges between them. A *partial tree* is a weakly connected subgraph of a rooted tree. Note the difference between a subtree and a partial tree. A *perfectly balanced binary tree* is a binary tree in which the depths of the subtrees rooted at children of any internal node differ by at most one.

The enclosing rectangle of a drawing can be completely described by its width, height and the coordinates of one of its corners, say the left-top one. Most times, during the description of our algorithm, we will assume that the left-top corner of the enclosing rectangle has coordinates $(0, 0)$. By fixing a point of reference, it is sufficient to describe a rectangle R by a pair of two integers, its width and height, i.e., $R = (w, h)$, $w \geq 0$, $h \geq 0$. Two rectangles are called *equal* if they have identical width and height.

Given two rectangles $R_1 = (w_1, h_1)$ and $R_2 = (w_2, h_2)$ we say that rectangle R_1 *dominates* (or *fits in*) R_2 if:

$$w_1 \leq w_2 \text{ and } h_1 \leq h_2.$$

Given a set S of rectangles, an *atom* is an element of S which dominates no other rectangle in S . Any set of atoms that are sorted in increasing order with respect to their widths, are also sorted in decreasing order with respect to their heights.

With each drawing we associate a cost. Our objective is to derive drawings of minimum cost. In this paper, the cost function will be a function of the enclosing rectangle of the drawing, i.e., a function $\psi : \mathbb{R}^2 \rightarrow \mathbb{R}$. Our results will hold for any function that is nondecreasing in both parameters, i.e., $\psi(x_1, y_1) \geq \psi(x_2, y_2)$ whenever $x_1 \geq x_2$ and $y_1 \geq y_2$. Let $R = (width, height)$. The following are commonly used cost functions that are non-decreasing in both parameters:

1. $area(R) = width \cdot height$
2. $perimeter(R) = 2(width + height)$
3. $minimum_enclosing_square(R) = \max(width, height)$
4. $height_for_a_given_width(R, w) = \begin{cases} height & \text{if } width \leq w \\ \infty & \text{otherwise} \end{cases}$

An *h-v drawing* of a binary rooted tree is an orthogonal straight-line planar grid drawing which also satisfies the following restrictions:

1. Any tree node v is drawn at the left-top corner of the enclosing rectangular in the partial drawing of the subtree rooted at v .
2. The enclosing rectangles of the partial drawings of the subtrees rooted at sibling nodes are non-overlapping.

The problem of *minimum size h-v drawing of a binary rooted tree T* is the problem of determining an h-v drawing of T of minimum cost with respect to some cost function ψ . Figure 4 shows two h-v drawings of the same tree. The drawing of Figure 4.a is of minimum area while that of Figure 4.b is of minimum enclosing square. Both drawings are of minimum perimeter.

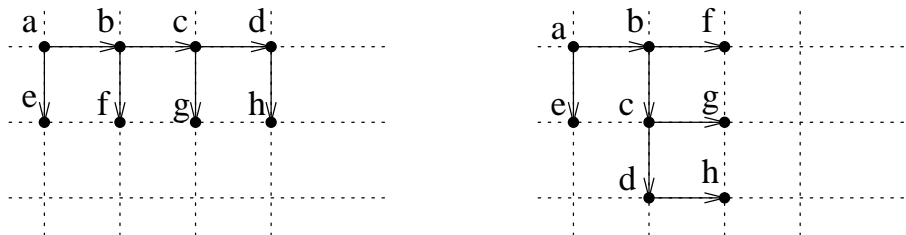


Figure 4: Examples of minimum size h-v drawings.

3 The Parallel Algorithm for h-v Drawings

The algorithm for finding a minimum size h-v drawing of a binary tree $T = (V, E)$ is based on the parallel tree contraction technique [1]. Within a logarithmic number of phases, the parallel tree-contraction algorithm contracts a tree T to its root by processing a logarithmic number of intermediate binary trees $T(i) = (V(i), E(i))$, $i = 0, 1, \dots, k$, with $k = O(\log |T|)$. Note that the algorithm starts off with $T(0) = T$, and proceeds by contracting tree $T(i-1)$ to tree $T(i)$ of $|T(i)| \leq \varepsilon |T(i-1)|$ nodes, $0 < \varepsilon < 1$. At the end, $T(k)$ contains only one node.

During the i th phase of the algorithm, the tree $T(i)$ is obtained from $T(i-1)$ by applying a local operation, called *shunt*, to a subset of the leaves of $T(i-1)$. The shunt operation is composed of two steps: In the first step, a subset of the tree leaves are removed (an operation called *pruning*) and their parent (or sibling) nodes are updated to reflect this fact. In the second step, each of these parents is removed (by an operation called *shortcutting*), and its child is updated. The shunt operation removes roughly half of the tree nodes, so ε is roughly $1/2$. To use the tree contraction technique, one has to describe the updates that take place during the shunt operation. Before we do that, we give some notation and describe the information associated with each node of the tree.

3.1 Data Structures and Useful Operations

If $v \in V(i)$ then let $T_v^{c_i}$ be the partial tree (of T) contracted to v after applying the shunt operation to siblings of v during the first i phases of the parallel tree-contraction algorithm.

With each node u in $V(i)$ we associate a tuple L_u containing the following information: The root r_u of the partial tree $T_u^{c_i}$ that has been contracted to u , and a set R_u that keeps information for all *useful* drawings of $T_u^{c_i}$ (the notion of a useful drawing will be defined shortly). Each element of R_u corresponds to a specific reduced *partial drawing*¹ π and consists of 3 tuples, i.e., $\pi = ((W_u, H_u), (A_u, B_u), (x_u, y_u))$. The first tuple, i.e., (W_u, H_u) , describes the width and the height of the enclosing rectangle of π . The second tuple, i.e., (A_u, B_u) describes the width A_u and the height B_u of the largest rectangle (having u at its top left corner) that can be included in the partial drawing π such that the enclosing rectangle (W_u, H_u) of π remains unchanged and π is still a valid h-v drawing of $T_u^{c_i}$. We shall refer to (A_u, B_u) as the *empty rectangle* corresponding to R_u . Finally, the third tuple, i.e., (x_u, y_u) , is the location of u in π , where $(0, 0)$ is the coordinate of the top left corner of any partial drawing. Note that, u is a leaf in the partial tree $T_u^{c_i}$. For each $u \in V = V_0$, we initialize $L_u = \langle u; ((0, 0), (0, 0), (0, 0)) \rangle$.

Suppose that at some phase of the parallel tree-contraction algorithm we want to include the partial drawing π of a partial tree rooted at a node v in a partial drawing π' of another partial tree whose v is a leaf. Then, we need to know the position of v in π' as well as how much the inclusion of π in π' will change the rectangle corresponding to π' . Thus, all the parameters associated above with each node of each $T(i)$, $i = 0, \dots, \log |T|$, are necessary for the parallel tree-contraction approach to work. In Section 3.3, we shall show that the information kept by those parameters is all that is needed for the algorithm to work.

Let $\pi^1 = ((W_u^1, H_u^1), (A_u^1, B_u^1), (x_u^1, y_u^1))$ and $\pi^2 = ((W_u^2, H_u^2), (A_u^2, B_u^2), (x_u^2, y_u^2))$ be two partial drawings of $T_u^{c_i}$.

Definition 3.1 We say that (partial) drawing π^1 dominates (or fits in) (partial) drawing π^2 if the enclosing rectangle of π^1 fits in the enclosing rectangle of π^2 .

Definition 3.2 Partial drawing π^1 prevails partial drawing π^2 with respect to integer $\lambda > 0$ if π^1 fits in π^2 and at least one of the following conditions is satisfied:

- a) (A_u^2, B_u^2) fits in (A_u^1, B_u^1)
- b) $A_u^2 \leq A_u^1$ and $B_u^2 > B_u^1 \geq \lambda$
- c) $A_u^2 > A_u^1 \geq \lambda$ and $B_u^2 \leq B_u^1$
- d) $A_u^2 > A_u^1 \geq \lambda$ and $B_u^2 > B_u^1 \geq \lambda$

The situation described by the above conditions is demonstrated in Figure 5. In this paper, when using the notion of “prevail”, λ will usually be the size (i.e., the number of nodes) of a partial tree.

¹We use the term *partial drawing* for two reasons: Firstly, it is a drawing of a partial tree, and secondly, only information about enclosing rectangles is kept.

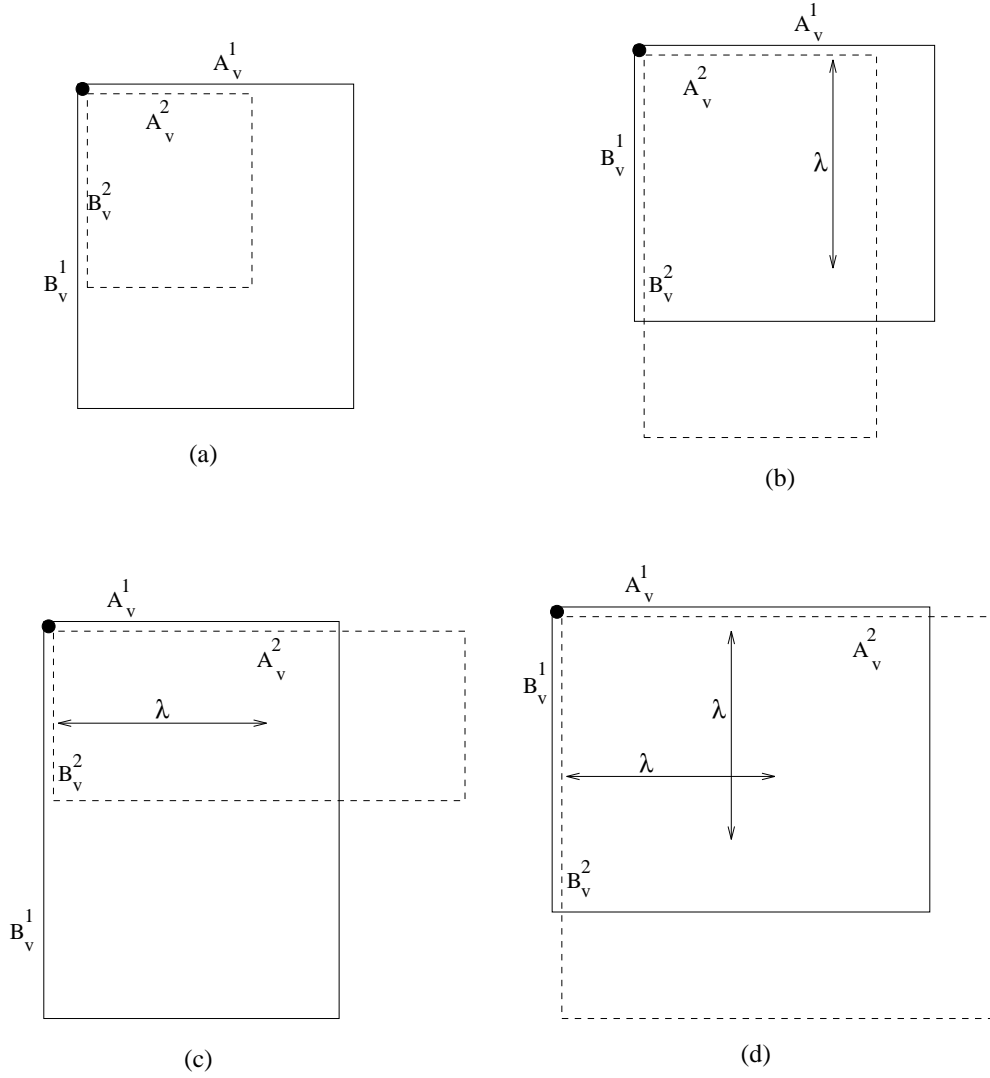


Figure 5: The situation described by Conditions a, b, c and d of the definition of *prevail*.

Definition 3.3 Let $u \in V(i)$, T_u^{ci} be the partial tree (of T) contracted to u during the first i phases of the parallel tree contraction algorithm, T_u be the subtree of T rooted at u and R a set of partial drawings of T_u^{ci} . A partial drawing π in R is called useful if π is prevailed, with respect to the size $|T_u|$ of tree T_u , by no other partial drawing in R .

Lemma 3.1 Let $u \in V(i)$, T_u^{ci} be the partial tree (of T) contracted to u during the first i phases of the parallel tree contraction algorithm and T_u be the subtree of T rooted at u . Then the number of useful partial drawings in L_u is at most $O(\min(|T_u|, |T_u^{ci}|) \cdot |T_u^{ci}|^2)$.

Proof: Let $\pi^1 = ((W_u^1, H_u^1), (A_u^1, B_u^1), (x_u^1, y_u^1))$ and $\pi^2 = ((W_u^2, H_u^2), (A_u^2, B_u^2), (x_u^2, y_u^2))$ be two partial drawings in R_u such that π^1 fits in π^2 and $A_u^2 > A_u^1 \geq |T_u|$ and $B_u^2 > B_u^1 \geq |T_u|$. Then, based on the definition of the *prevail* operation, π^1 prevails π^2 and π^2 is eliminated. π^2 is also eliminated in the case that $A_u^2 \leq A_u^1$ and $B_u^2 \leq B_u^1$. This means that for partial

drawings that have empty rectangles with width and/or height greater than $|T_u|$, the width and/or the height of the empty rectangle is irrelevant for the prevail operation. Thus, given a partial drawing $\pi = ((W_u, H_u), (A_u, B_u), (x_u, y_u))$ that has A_u and/or B_u greater than $|T_u|$, we may replace it in R_u by a partial drawing π' whose A_u and/or B_u parameters are equal to $|T_u|$. Let R'_u be the new set of partial drawings.

Notice that in R'_u , we cannot have two partial drawings with the same enclosing rectangle whose empty rectangles dominate each other. The maximum size of any one of the sides of an empty rectangle can be at most $|T_u^{c_i}|$, but we only need rectangles big enough to fit the drawing of T_u i.e., we need empty rectangles with side size at most $|T_u|$. Thus, the interesting partial drawings are the ones with empty rectangles whose side size is at most $\min(|T_u|, |T_u^{c_i}|)$. Note that the maximum number of different empty rectangles such that no empty rectangle is dominated by another one is at most $\min(|T_u|, |T_u^{c_i}|)$ and the number of different enclosing rectangles is $O(|T_u^{c_i}|^2)$. This results to a total of at most $O(\min(|T_u|, |T_u^{c_i}|) \cdot |T_u^{c_i}|^2)$ useful partial drawings. ■

3.2 The Shunt Updates

Let l be a leaf in tree $T(i-1)$, s be l 's sibling, f be l 's parent and p be f 's parent. Let also $L_f = \langle r_f; R_f \rangle$, where

$$R_f = \{((W_f^1, H_f^1), (A_f^1, B_f^1), (x_f^1, y_f^1)), \dots, ((W_f^i, H_f^i), (A_f^i, B_f^i), (x_f^i, y_f^i))\},$$

$L_s = \langle r_s; R_s \rangle$, where

$$R_s = \{((W_s^1, H_s^1), (A_s^1, B_s^1), (x_s^1, y_s^1)), \dots, ((W_s^j, H_s^j), (A_s^j, B_s^j), (x_s^j, y_s^j))\}$$

and $L_l = \langle r_l; R_l \rangle$, where

$$R_l = \{((W_l^1, H_l^1), (\cdot, \cdot), (\cdot, \cdot)), \dots, ((W_l^k, H_l^k), (\cdot, \cdot), (\cdot, \cdot))\},$$

be the information associated with f , s and l respectively².

Recall that R_l , R_f and R_s keep information for all useful partial drawings of $T_l^{c_{i-1}}$, $T_s^{c_{i-1}}$, and $T_f^{c_{i-1}}$ respectively. Note that since l is a leaf of T , $T_{r_l} = T_l^{c_{i-1}}$. Note also that r_l and r_s are the children of f in the tree $T = T(0)$.

During the i th phase of the tree contraction algorithm, we apply the shunt operation to a set of leaves of $V(i-1)$. The shunt operation to a leaf l of $V(i-1)$ consists of two stages, namely, a *pruning* and a *shortcutting* stage (Figure 6).

3.2.1 The Pruning Stage

In the pruning stage, we use the tuples L_l and L_s to construct the tuple $L_{f'} = \langle f; R_{f'} \rangle$ containing information about all useful (with respect to the size $|T_s|$ of T_s) partial drawings of the partial tree rooted at f and including the subtree T_{r_l} and the partial tree $T_s^{c_{i-1}}$.

²Because we deal with the drawing of subtree T_{r_l} rather than a partial tree, we do not have to record any information about an empty rectangle. Thus, the notation $((W_{l'}, H_{l'}), (\cdot, \cdot), (\cdot, \cdot))$.

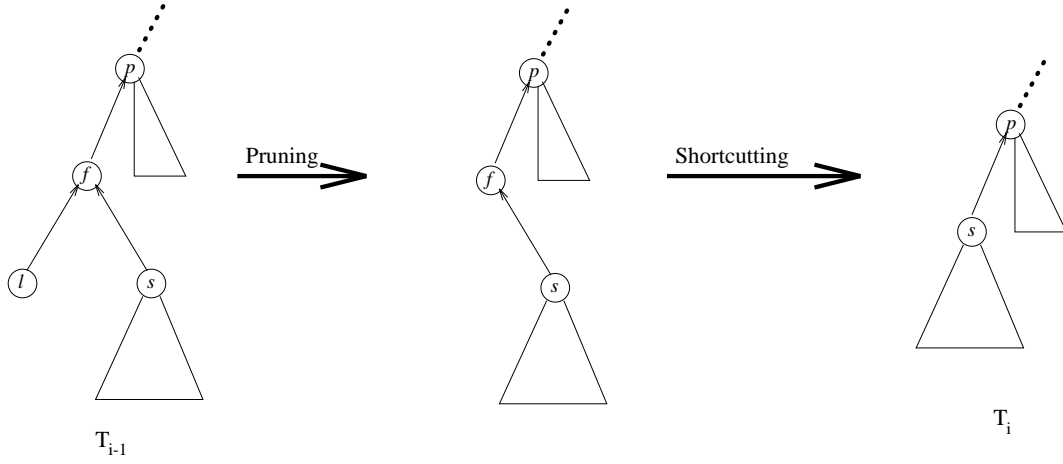


Figure 6: Application of the shunt operation to leaf l .

Let $\pi_s = ((W_s, H_s), (A_s, B_s), (x_s, y_s))$ be an element of the set R_s and $\pi_l = ((W_l, H_l), (\cdot, \cdot), (\cdot, \cdot))$ be an element of the set R_l .

There are essentially 4 ways to arrange T_{r_l} and $T_u^{c_i-1}$. For each one we compute the new drawing. In what follows, the superscripts in $(x_s^{f'}, y_s^{f'})$ are used to avoid confusion with (x_s, y_s) . $(x_s^{f'}, y_s^{f'})$ are the coordinates of s in the drawing of the partial tree rooted at f and including the subtree T_{r_l} and $T_s^{c_i-1}$ while, (x_s, y_s) are the coordinates of s in the partial drawing of $T_s^{c_i-1}$.

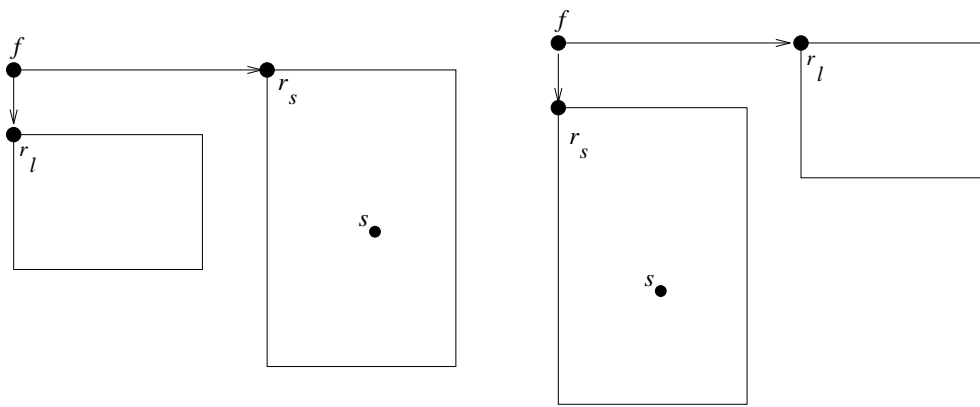
Case 1: The situation in case 1 is described in Figure 7.a. The produced partial drawing $((W_{f'}, H_{f'}), (A_{f'}, B_{f'}), (x_s^{f'}, y_s^{f'}))$ is defined by:

$$\begin{aligned}
W_{f'} &:= W_l + W_s + 1 \\
H_{f'} &:= \max(H_l + 1, H_s) \\
A_{f'} &:= A_s \\
B_{f'} &:= B_s + \max(0, H_l + 1 - H_s) \\
x_s^{f'} &:= x_s + W_l + 1 \\
y_s^{f'} &:= y_s
\end{aligned}$$

The correctness of the computed values for $W_{f'}, H_{f'}, A_{f'}, x_s^{f'}, y_s^{f'}$ is obvious from Figure 7.a. For $B_{f'}$, note that we simply extend the height of the empty rectangle up to the bounds of the enclosing rectangle.

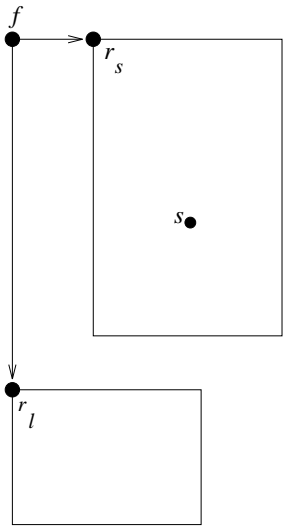
Case 2: The situation in case 2 is described in Figure 7.b. The produced partial drawing $((W_{f'}, H_{f'}), (A_{f'}, B_{f'}), (x_s^{f'}, y_s^{f'}))$ is defined by:

$$\begin{aligned}
W_{f'} &:= W_l + W_s + 1 \\
H_{f'} &:= \max(H_s + 1, H_l) \\
A_{f'} &:= A_s \\
B_{f'} &:= B_s + \max(0, H_l - H_s - 1)
\end{aligned}$$

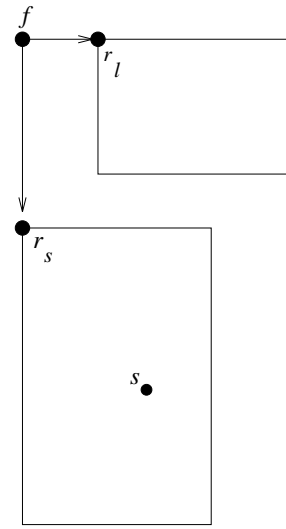


(a)

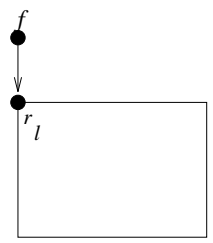
(b)



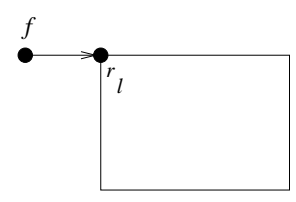
(c)



(d)



(e)



(f)

Figure 7: The cases which occur when combining the subtree T_{r_l} together with the partial tree $T_s^{r_{l-1}}$ during the pruning phase of the shunt operation.

$$\begin{aligned}x_s^{f'} &:= x_s \\y_s^{f'} &:= y_s + 1\end{aligned}$$

Case 3: The situation in case 3 is described in Figure 7.c. The produced partial drawing $((W_{f'}, H_{f'}), (A_{f'}, B_{f'}), (x_s^{f'}, y_s^{f'}))$ is defined by:

$$\begin{aligned}W_{f'} &:= \max(W_s + 1, W_l) \\H_{f'} &:= H_s + H_l + 1 \\A_{f'} &:= A_s + \max(0, W_l - W_s - 1) \\B_{f'} &:= B_s \\x_s^{f'} &:= x_s + 1 \\y_s^{f'} &:= y_s\end{aligned}$$

Case 4: The situation in case 4 is described in Figure 7.d. The produced partial drawing $((W_{f'}, H_{f'}), (A_{f'}, B_{f'}), (x_s^{f'}, y_s^{f'}))$ is defined by:

$$\begin{aligned}W_{f'} &:= \max(W_l + 1, W_s) \\H_{f'} &:= H_l + H_s + 1 \\A_{f'} &:= A_s + \max(0, W_l + 1 - W_s) \\B_{f'} &:= B_s \\x_s^{f'} &:= x_s \\y_s^{f'} &:= y_s + H_l + 1\end{aligned}$$

Note that in all of the above cases the size of the empty rectangle is extended up to the bounds of the enclosing rectangle.

It is possible that the sibling s of leaf l does not exist. This can happen when tree T is not a regular binary tree. This leads to the following additional cases:

Case 5: The situation in case 5 is described in Figure 7.e. The produced drawing $((W_{f'}, H_{f'}), (\cdot, \cdot), (\cdot, \cdot))$ of subtree T_{r_f} is defined by:

$$\begin{aligned}W_{f'} &:= W_l \\H_{f'} &:= H_l + 1\end{aligned}$$

Case 6: The situation in case 6 is described in Figure 7.f. The produced drawing $((W_{f'}, H_{f'}), (\cdot, \cdot), (\cdot, \cdot))$ of subtree T_{r_f} is defined by:

$$\begin{aligned}W_{f'} &:= W_s + 1 \\H_{f'} &:= H_s\end{aligned}$$

After computing all the possible partial drawings, the prevail operation (with respect to the size $|T_s|$ of T_s) is applied to them and the set $R_{f'}$ is obtained. I.e., all the drawings that are prevailed by other drawings are eliminated.

The parallel implementation of the prevail operation consists of two major steps. In the first step, each drawing $((W_{f'}, H_{f'}), (A_{f'}, B_{f'}), (x_s^{f'}, y_s^{f'}))$ with $A_{f'} > |T_s|$ and/or $B_{f'} > |T_s|$ is substituted by the drawing $((W_{f'}, H_{f'}), (A, B), (x_s^{f'}, y_s^{f'}))$ where $A = |T_s|$ and/or $B = |T_s|$. In the second step, basic parallel algorithmic techniques are employed (e.g., sorting, pointer doubling and list ranking) and the drawings that are prevailed by other drawings are eliminated.

3.2.2 The Shortcutting Stage

In the shortcutting stage, the tuples $L_{f'}$ and L_f are used to construct the new tuple for L_s . The root of the new L_s will be r_f . To determine an element π of the new set R_s we combine drawings $\pi^{f'} = ((W_{f'}, H_{f'}), (A_{f'}, B_{f'}), (x_s^{f'}, y_s^{f'}))$ of $R_{f'}$ with drawing $\pi^f = ((W_f, H_f), (A_f, B_f), (x_f, y_f))$ of R_f . In simple words, we embed $\pi^{f'}$ into π^f .

The new element $\pi = ((W_s, H_s), (A_s, B_s), (x_s, y_s))$ of R_s produced by embedding $\pi^{f'}$ into π^f is computed as follows:

$$\begin{aligned}
W_s &:= W_f + \max(0, W_{f'} - A_f) \\
H_s &:= H_f + \max(0, H_{f'} - B_f) \\
A_s &:= A_{f'} + \max(0, A_f - W_{f'}) \\
B_s &:= B_{f'} + \max(0, B_f - H_{f'}) \\
x_s &:= x_f + x_s^{f'} \\
y_s &:= y_f + y_s^{f'}
\end{aligned} \tag{1}$$

The correctness of the computed values for W_s and H_s is obvious. For A_s and B_s , note that in the case that $A_f > W_{f'}$ and/or $B_f > H_{f'}$, A_s equals $A_{f'} + A_f - W_{f'}$ and/or $B_s = B_{f'} + B_f - H_{f'}$. I.e., the empty rectangle of π is extended to be as large as the difference between the sizes of the enclosing rectangle of $\pi^{f'}$ and the empty rectangle of π^f allow.

Finally, the prevail operation (with respect to the size $|T_s|$ of T_s) is applied to the computing drawings and the new set R_s is obtained. The implementation of the prevail is done in the same way as in the pruning stage.

3.3 Correctness and Analysis

To prove the correctness of the algorithm we have to establish that during the course of the algorithm we keep all the necessary information. To do that we have to prove two things: Firstly, that it is enough to keep only one drawing out of those that differ only in the coordinates of the “interface to bellow”, i.e., they have the same enclosing and “empty”

rectangles and, secondly, that we can safely use the prevail operation to shorten our R -sets after each stage.

Lemma 3.2 Consider two drawings $\pi 1_s^{i-1} = ((W_s, H_s), (A_s, B_s), (x_{1_s}, y_{1_s}))$ and $\pi 2_s^{i-1} = ((W_s, H_s), (A_s, B_s), (x_{2_s}, y_{2_s}))$ of R_s that differ only in the coordinates of s in the drawing. Also assume a drawing $\pi 1_i^{i-1} = ((W_i, H_i), (\cdot, \cdot), (\cdot))$ of R_i and a drawing $\pi 1_f^{i-1} = ((W_f, H_f), (A_f, B_f), (x_f, y_f))$ of R_f . Let $\pi 1_s^i$ be the drawing obtained by combining $\pi 1_s^{i-1}$, $\pi 1_i^{i-1}$ and $\pi 1_f^{i-1}$ during phase i . Let $\pi 2_s^i$ be the drawing obtained by combining $\pi 2_s^{i-1}$, $\pi 1_i^{i-1}$ and $\pi 1_f^{i-1}$ during phase i . Then, $\pi 1_s^i$ and $\pi 2_s^i$ have the same enclosing and “empty” rectangles.

Proof: Easy. Simply observe that the rules for computing the enclosing and empty rectangles during the pruning and shortcutting stages of each phase, do not use the coordinates of s . ■

Lemma 3.3 We can “safely” eliminate prevailed elements from R_s after the end of each shortcutting stage.

Proof: Let $\pi^1 = ((W_s^1, H_s^1), (A_s^1, B_s^1), (\cdot, \cdot))$ and $\pi^2 = ((W_s^2, H_s^2), (A_s^2, B_s^2), (\cdot, \cdot))$ be two partial drawings of R_s such that π^1 prevails π^2 . Then, π^1 fits in π^2 and at least one of the following conditions is true:

- i) $A_s^2 \leq A_s^1$ and $B_s^2 \leq B_s^1$
- ii) $A_s^2 \leq A_s^1$ and $B_s^2 > B_s^1 \geq \min(|T_s|, |T_s^{c_i}|)$
- iii) $A_s^2 > A_s^1 \geq \min(|T_s|, |T_s^{c_i}|)$ and $B_s^2 \leq B_s^1$
- iv) $A_s^2 > A_s^1 \geq \min(|T_s|, |T_s^{c_i}|)$ and $B_s^2 > B_s^1 \geq \min(|T_s|, |T_s^{c_i}|)$.

We distinguish among the following cases:

- a) (π^1 fits in π^2) and ($A_s^2 \leq A_s^1$ and $B_s^2 \leq B_s^1$).

Then, according to the definition of *prevail*, π^2 is eliminated. Assume that it is not safe to do so. That means that there exists a drawing (atom) of T_{r_s} that we only get by using the partial drawing π^2 which we want to eliminate. Say that this atom has dimensions $(W_{r_s}^2, H_{r_s}^2)$. Also assume that the drawing of T_s in this atom is π_{T_s} of dimensions (W, H) .

Then, we must have that:

$$\begin{aligned} W_{r_s}^2 &= W_s^2 + \max(0, W - A_s^2) \\ H_{r_s}^2 &= H_s^2 + \max(0, H - B_s^2) \end{aligned} \tag{2}$$

But, consider the drawing obtained by using π_{T_s} and π^1 . We must have that:

$$\begin{aligned} W_{r_s}^1 &= W_s^1 + \max(0, W - A_s^1) \\ H_{r_s}^1 &= H_s^1 + \max(0, H - B_s^1) \end{aligned} \tag{3}$$

Note that:

$$\begin{aligned}
\max(0, W - A_s^1) &\leq \max(0, W - A_s^2) \\
\max(0, H - B_s^1) &\leq \max(0, H - B_s^2) \\
W_s^1 &\leq W_s^2 \\
H_s^1 &\leq H_s^2
\end{aligned} \tag{4}$$

From (1),(2) and (3) we get that

$$W_{r_s}^1 \leq W_{r_s}^2 \quad \text{and} \quad H_{r_s}^1 \leq H_{r_s}^2.$$

Thus π^2 is not an atom, a contradiction.

b) (π^1 fits in π^2) and ($A_s^2 \leq A_s^1$ and $B_s^2 > B_s^1 \geq \min(|T_s|, |T_s^{c_i}|)$)

Again, according to the definition of prevail, π^2 is eliminated. This is safe, since in this case the interesting drawings are the drawings $\pi^{1'}$ and $\pi^{2'}$ that have the same enclosing rectangles with π^1 and π^2 respectively, but their empty rectangles are $(A_s^1, \min(|T_s|, |T_s^{c_i}|))$ and $(A_s^2, \min(|T_s|, |T_s^{c_i}|))$, respectively. Now, because of case a), $\pi^{2'}$ can be safely eliminated.

c) (π^1 fits in π^2) and ($A_s^2 > A_s^1 \geq \min(|T_s|, |T_s^{c_i}|)$ and $B_s^2 \leq B_s^1$).

In this case, the interesting drawings $\pi^{1'}$ and $\pi^{2'}$ have as empty rectangles the $(\min(|T_s|, |T_s^{c_i}|), B_s^1)$ and $(\min(|T_s|, |T_s^{c_i}|), B_s^2)$ respectively. Thus, $\pi^{2'}$ can be safely eliminated.

d) (π^1 fits in π^2) and ($A_s^2 > A_s^1 \geq \min(|T_s|, |T_s^{c_i}|)$ and $B_s^2 > B_s^1 \geq \min(|T_s|, |T_s^{c_i}|)$).

Obviously, the drawings $\pi^{1'}$ and $\pi^{2'}$ have the same empty rectangle $(\min(|T_s|, |T_s^{c_i}|), \min(|T_s|, |T_s^{c_i}|))$. Thus, $\pi^{2'}$ can be safely eliminated. ■

Lemma 3.4 *We can “safely” eliminate prevailed elements from R_s after the end of each pruning stage.*

Proof: Let $T_{f'}$ be the partial tree rooted at f and containing T_l and $T_s^{c_i-1}$ as f 's children. Let $\pi_{f'}^1 = ((W_{f'}^1, H_{f'}^1), (A_{f'}^1, B_{f'}^1), (\cdot, \cdot))$ and $\pi_{f'}^2 = ((W_{f'}^2, H_{f'}^2), (A_{f'}^2, B_{f'}^2), (\cdot, \cdot))$ be two partial drawings of $R_{f'}$ such that $\pi_{f'}^1$ prevails $\pi_{f'}^2$.

Then $\pi_{f'}^1$ fits in $\pi_{f'}^2$, and at least one of the following conditions is true:

- i) $A_{f'}^2 \leq A_{f'}^1$ and $B_{f'}^2 \leq B_{f'}^1$
- ii) $A_{f'}^2 \leq A_{f'}^1$ and $B_{f'}^2 > B_{f'}^1 \geq |T_{f'}|$
- iii) $A_{f'}^2 > A_{f'}^1 \geq |T_{f'}|$ and $B_{f'}^2 \leq B_{f'}^1$
- iv) $A_{f'}^2 > A_{f'}^1 \geq |T_{f'}|$ and $B_{f'}^2 > B_{f'}^1 \geq |T_{f'}|$.

Let $\pi_f = ((W, H), (A, B), (\cdot, \cdot))$ be a partial drawing of $T_f^{c_i-1}$ with which $\pi_{f'}^1$ and $\pi_{f'}^2$ will be combined during the shortcutting stage. We will show that it is safe to eliminate $\pi_{f'}^2$.

Again, we distinguish 4 cases:

a) $(\pi_{f'}^1 \text{ fits in } \pi_{f'}^2)$ and $(A_{f'}^2 \leq A_{f'}^1 \text{ and } B_{f'}^2 \leq B_{f'}^1)$.

This can be equivalently written as:

$$\begin{aligned} W_{f'}^1 &\leq W_{f'}^2 \\ H_{f'}^1 &\leq H_{f'}^2 \\ A_{f'}^2 &\leq A_{f'}^1 \\ B_{f'}^2 &\leq B_{f'}^1 \end{aligned} \tag{5}$$

Consider the partial drawings π_s^1 and π_s^2 we get when we combine π_f with $\pi_{f'}^1$ and $\pi_{f'}^2$, respectively. For π^1 we have:

$$\begin{aligned} W_s^1 &= W + \max(0, W_{f'}^1 - A) \\ H_s^1 &= H + \max(0, H_{f'}^1 - B) \\ A_s^1 &= A_{f'}^1 + \max(0, A - W_{f'}^1) \\ B_s^1 &= H_{f'}^1 + \max(0, B - H_{f'}^1) \end{aligned} \tag{6}$$

For π^2 we have:

$$\begin{aligned} W_s^2 &= W + \max(0, W_{f'}^2 - A) \\ H_s^2 &= H + \max(0, H_{f'}^2 - B) \\ A_s^2 &= A_{f'}^1 + \max(0, A - W_{f'}^2) \\ B_s^2 &= H_{f'}^1 + \max(0, B - H_{f'}^2) \end{aligned} \tag{7}$$

Combining (4), (5) and (6) we get that π_s^1 fits in π_s^2 and also (A_s^2, B_s^2) fits in (A_s^1, B_s^1) . Thus π_s^1 prevails π_s^2 . So, the use of $\pi_{f'}^2$ resulted to a partial drawing that will be eliminated (according to Lemma 3.3) after the shortcutting stage.

The proof for the remaining three cases proceeds on the same lines as in Lemma 3.3. ■

Theorem 3.1 *A minimum size h-v drawing of a binary tree with n nodes can be correctly found in $O(\log^2 n)$ parallel time using $O(n^6/\log n)$ EREW processors.*

Proof (sketch): In the pruning stage, there are at most four possible ways to arrange the subtrees T_{r_i} and $T_s^{c_i-1}$. All of them have been considered and the corresponding partial drawings have been obtained. For the shortcutting stage, there is only one way to embed $\pi_{f'}$ into π_f . In addition, by the previous lemmata, the prevail operation does not eliminate useful partial drawings. At the end of the tree contraction, the root of the tree has a list of n drawings (atoms). We evaluate the cost function ψ for each drawing and we select the one of minimum cost (size). Thus, the algorithm correctly computes a minimum size h-v drawing.

For the time and processor bounds of the algorithm we note the following: During pruning, since l is a leaf, the tree T_l includes only l . Thus, we may have only $O(|T_{r_i}|)$ elements in the set

R_l . Also, from Lemma 3.1, the list R_s contains at most $O(\min(|T_s|, |T_s^{c_{i-1}}|) \cdot |T_s^{c_{i-1}}|^2)$ partial drawings. This means that at most $O(|T_{r_l}| \cdot \min(|T_s|, |T_s^{c_{i-1}}|) \cdot |T_s^{c_{i-1}}|^2)$ partial drawings are computed, on which we apply the prevail operation to get $R_{f'}$. The number of partial drawings in $R_{f'}$ is at most $O(\min(|T_{r_l}| + |T_s^{c_{i-1}}|, |T_s|) \cdot (|T_{r_l}| + |T_s^{c_{i-1}}|)^2)$.

After shortcutting, the number of computed partial drawings is $|R_f| \cdot |R_{f'}| = \min(|T_f^{c_{i-1}}|, |T_f|) \cdot |T_f^{c_{i-1}}|^2 \cdot \min(|T_{r_l}| + |T_s^{c_{i-1}}|, |T_s|) \cdot (|T_{r_l}| + |T_s^{c_{i-1}}|)^2$. Again, the prevail operation is applied on them. Note that, at each phase of the parallel tree contraction algorithm, each tree node contributes to the complexity of only one shunt operation (applied to a leaf l). This comes from the fact that the number of computed partial drawings depends on the number of nodes that have already been contracted to the nodes f , l and s . Thus, on each phase of the parallel tree contraction algorithm, for both the computation of the elements and the implementation of the prevail operation, a total number of $O(n^6)$ processors is employed. $O(\log n)$ time is needed for the implementation of the prevail. This gives a total of $O(\log^2 n)$ parallel time and $O(n^6/\log n)$ processors for the parallel tree contraction algorithm. ■

3.4 Minimum Area h-v Drawings

When we are after minimum area h-v drawings, the number of processors required by the parallel algorithm can be substantially reduced. To achieve that, we used a result due to Crescenzi, Di Battista and Piperno which was developed in the context of upward drawings of binary trees [3]. An upward drawing of a binary tree is quite similar to an h-v drawing. The only difference is that in an upward drawing of a tree the enclosing rectangles which correspond to partial drawing of subtrees rooted at sibling nodes are allowed to overlap. Related results regarding minimum area upward drawings of general trees were obtained by Garg, Goodrich and Tamassia [8].

Theorem 3.2 (Crescenzi, Di Battista, Piperno, [3]) *For any binary tree T of n nodes, there exists an h-v drawing of T with at most $n(\log n + 1)$ area. Moreover, the width of the layout is at most $\log n + 1$ while its height is at most n . ■*

By using the above result we can reduce the number of partial drawings of $T_u^{c_i}$ in R_u during each tree contraction phase. More precisely, we only keep partial drawings with area bounded by $n(\log n + 1)$. As a result, at the end of each pruning stage there are at most $n^2(\log n + 1)$ partial drawings in any R-list while, during the shortcutting stage, a total of at most $n^4(\log n + 1)^2$ partial drawings might be created. Thus, on the same lines with Theorem 3.1, we can prove:

Theorem 3.3 *A minimum area h-v drawing of a binary tree with n nodes can be correctly found in $O(\log^2 n)$ parallel time using $O(n^4 \log n)$ EREW processors. ■*

In the case we are interested in some layout of area bounded by $n(\log n + 1)$ rather than a minimum area layout, the number of processors required by our parallel solution can be further reduced to $O(n^2 \log^3 n)$. This is achieved by taking into account that there exist layouts with width at most $\log n + 1$ which satisfy this area requirement.

Theorem 3.4 *A h-v drawing of area at most $n(\log n + 1)$ of a binary tree with n nodes can be correctly found in $O(\log^2 n)$ parallel time using $O(n^2 \log^3 n)$ EREW processors. ■*

4 Related Problems

4.1 Inclusion Drawings of Binary Trees

When the inclusion drawing convention is used to draw a rooted binary tree each tree node is represented by a rectangle which has its sides parallel to the axes and the parent-child relationship is represented by enclosing the rectangle associated with the child within that of the parent. Rectangles associated with sibling nodes are non-overlapping, next to each other (same X or Y coordinate of, say, the top-left corner) and in at least distance δ from each other and from the sides of the rectangle of their parent. Leaves are represented by rectangles of size $a \times b$. We assume that δ , a and b are constants and multiples of the “unit” of length.

The method used to compute minimum size h-v drawings can be also used to compute inclusion drawings. The data structures are almost identical, i.e., we keep lists of partial drawings, where, each partial drawing consists of an enclosing rectangle, an empty rectangle and the coordinates of the left-top corner of the empty rectangle. Initially, the list of each leaf contains the partial drawing $((a, b), (0, 0), (0, 0))$ while the lists of internal nodes contain the partial drawing $((0, 0), (0, 0), (0, 0))$.

There are five possible arrangements to consider during the pruning stage (of leaf l with sibling s and parent f) shown in Figure 8.

For the arrangement in Figure 8.a we obtain the partial drawing with parameters (we use notation identical to that of the h-v drawings):

$$\begin{aligned} W_{f'} &:= W_l + W_s + 3\delta \\ H_{f'} &:= \max(H_l, H_s) + 2\delta \\ A_{f'} &:= A_s \\ B_{f'} &:= B_s + \max(0, H_l - H_s) \\ x_s^{f'} &:= x_s + W_l + 2\delta \\ y_s^{f'} &:= y_s + \delta \end{aligned}$$

In a similar fashion we can calculate the partial drawings for the remaining four arrangements as well as those for the shortcutting stage. Lemmata which establish that we can safely use the prevail operation can be also proved. Putting everything together, we get:

Theorem 4.1 *A minimum size inclusion drawing of a binary tree with n nodes can be found in $O(\log^2 n)$ parallel time using $O(((\delta + \max(a, b))n)^6 / \log n)$ EREW processors.*

In the case that the rectangle associated with leaf l is of dimensions $l_x \times l_y$ rather than

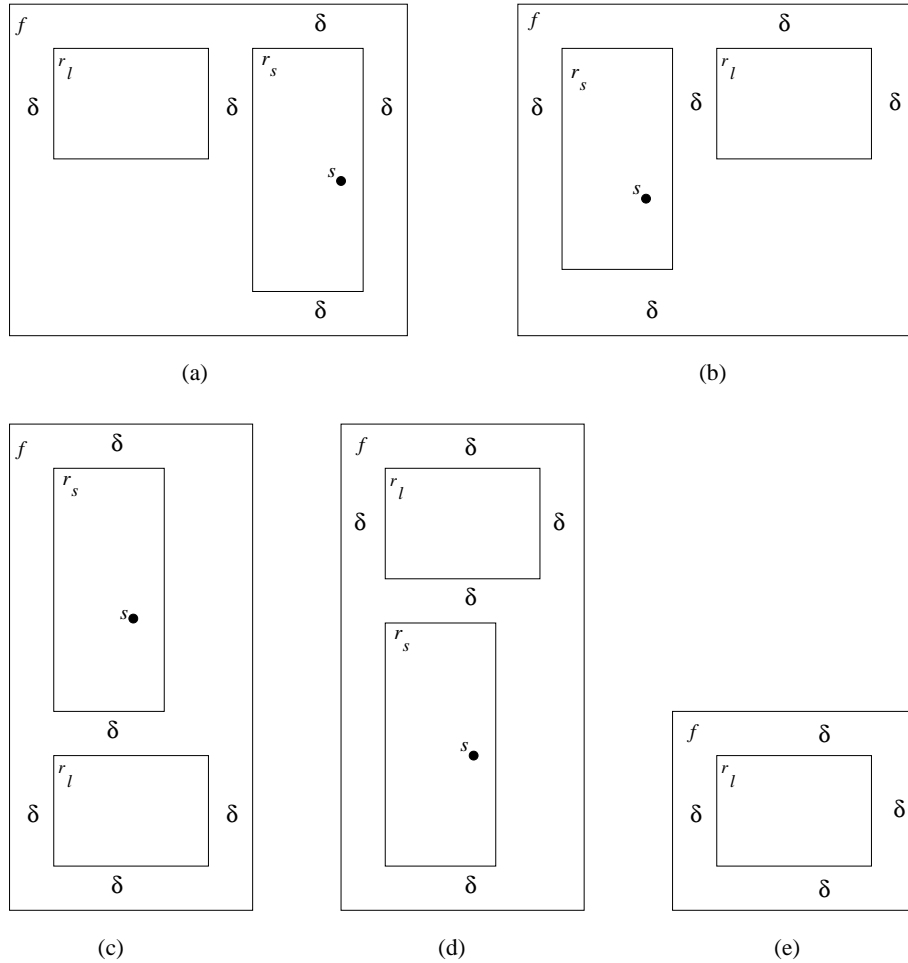


Figure 8: The cases which occur in inclusion drawings when combining the subtree T_{r_l} together with the partial tree $T_s^{c_{i-1}}$ during the pruning phase of the shunt operation.

$a \times b^3$, the number of required processors increases. Let $L = \sum_{\text{leaves } l} \max(l_x, l_y)$. Then, as a corollary of Theorem 4.1 we get:

Corollary 4.1 *A minimum size inclusion drawing of a binary tree with n nodes in which each leaf l is associated with a rectangle of size $l_x \times l_y$ can be found in $O(\log^2 n)$ parallel time using $O(\delta n + L)^6 / \log n$ EREW processors, where $L = \sum_{\text{leaves } l} \max(l_x, l_y)$*

In this case, the problem is shown to be NP-hard and a pseudopolynomial dynamic programming algorithm is known [6].

³Again, l_x and l_y are supposed to be multiples of the “unit” of length.

4.2 Slicing Floorplanning

In slicing floorplanning we are given a slicing tree in which leaves represent modules that are to be placed in the plane. Each internal node is an H or V node. In the final drawing, the leaves are drawn as rectangles of the corresponding module size in vertical or horizontal orientation (i.e., $x \times y$ or $y \times x$) and internal nodes by horizontal or vertical line segments. The drawing of the subtrees rooted at the children of a V-node (H-node) are drawn next to (on top of) each other. Alternatively, the drawing of the subtree rooted at a V-node (H-node) consists of two vertical (horizontal) slices, each next to (on top of) each other and containing the drawings of the subtrees rooted at its children.

The problem appears to be quite similar to that of inclusion drawings, so, a similar approach is expected to work. However, there are two important differences which make slicing floorplanning to be easier than that of inclusion drawing, at least with respect to sequential algorithms. Firstly, while for inclusion drawings there are four ways to combine drawings of the subtrees rooted at the children of node v to the drawing of the subtree rooted at v , in slicing floorplanning there is only one way determined by the type (H or V) of node v . Secondly, in inclusion drawings the rectangles which represent leaves are of fixed orientation while, in slicing floorplanning we have a choice on the orientation of the rectangle, i.e., for a leaf of size $a \times b$ we have associated the rectangles $a \times b$ and $b \times a$. So, for a regular binary tree of $n + 1$ leaves there can be $2^{n+1} = 2 \cdot 2^n$ possible different slicing floorplans and $4^n = (2^n)^2$ possible different inclusion drawings. The fact that the number of possible different slicing floorplans is smaller than that of inclusion drawings together, possibly, with the fact that it is easier to handle choices at the leaf level, make an important difference, at least for sequential algorithms. Stockmayer's algorithm [9] computes a minimum size slicing floorplan in $O(n^2)$ time while, an almost identical algorithm [6] computes in the same time bound minimum inclusion drawings only for perfectly balanced trees.

We can apply our method to get a parallel solution for the slicing floorplanning problem. We set δ to 0 while, for the pruning stage, we compute partial floorplans based on whether the parent of the leaf is an H or V node. Initially, the list of partial floorplans of each leaf of the slicing tree contains two partial floorplans, one for each orientation of the leaf's module.

Let the module associated with leaf l of the slicing tree have dimensions $l_x \times l_y$, $l_x \leq l_y$. Let $L = \sum_{\text{leaves } l} l_y$. From Corollary 4.1 we get:

Theorem 4.2 *A minimum size slicing floorplan of a slicing tree with n leaves in which each leaf l is associated with a module of size $l_x \times l_y$, $l_x \leq l_y$, can be found in $O(\log^2 n)$ parallel time using $O(L^6 / \log n)$ EREW processors, where $L = \sum_{\text{leaves } l} l_y$.*

Thus, for slicing floorplans which can be drawn in polynomial area, the problem of determining an optimal area slicing floorplan is placed in the class NC.

5 Conclusions

In this paper, we presented a parallel method which constructs optimal h-v and inclusion drawings of binary trees. Even though the number of processors involved in our method is high, our work places the problem in NC, presenting the first algorithm with polylogarithmic time complexity. The number of processors involved in the NC algorithm is large. It will be nice to get a solution with a smaller number of processors. The only sequential solution we know is based on the bottom-up approach and takes $O(n^2)$ time. It is open to derive an $o(n^2)$ sequential algorithm.

References

- [1] K. Abrahamson, N. Dadoun, D. Kirkpatrick and T. Przytycka, A Simple Parallel Tree Contraction Algorithm. *J. of Algorithms*, 10(1989), pp. 287–302.
- [2] Chen, C-H and Tollis, I. Parallel Algorithms for Slicing Floorplan Designs. In *Proc. of SPDP '90*. Dec. 1990, pp. 279–282.
- [3] Crescenzi, P., Di Battista, G. and Piperno, A. A Note on Optimal Area Algorithms for Upward Drawings of Binary Trees. *Computational Geometry: Theory and Applications*, Vol. 2, pp. 187–200, 1992.
- [4] Di Battista, G. and Tamassia, R. Algorithms for Plane Representation of Acyclic Digraphs. *Theoretical Computer Science*, Vol. 61, pp. 175–198, 1988.
- [5] Di Battista, G., Eades, P., Tamassia, R. and Tollis, I.G. Algorithms for Drawing Graphs: an Annotated Bibliography. Tech. Report, Brown University, June 1993.
- [6] Eades, P., Lin, T., and Lin, X. Two Tree Drawing Conventions. TR 174, Key Centre for Software Technology, Dept. of Computer Science, The University of Queensland, 1990. (To appear in *Computational Geometry and Applications*.)
- [7] Eades, P., Lin, T., and Lin, X. Minimum Size h-v Drawings, *Advanced Visual Interfaces (Proceedings of AVI 92)*, World Series in Computer Science Vol. 36, pp. 386–394, 1992.
- [8] Garg, A., Goodrich, M.T., and Tamassia, R. Area-Efficient Upward Tree Drawings, 9th Annual Symposium on Computational Geometry, pp. 359–368, 1992.
- [9] Stockmeyer, L. Optimal Orientations of Cells in Slicing Floorplan Designs. *Information and Control* **57**, pp. 91–101, 1983