ELSEVIER Theoretical Computer Science 181 (1997) 267–287

Theoretical Computer Science

Parallel algorithms for the minimum cut and the minimum length tree layout problems[1]

Josep Díaz[a], Alan Gibbons[b], Grammati E. Pantziou[c,*], Maria J. Serna[a], Paul G. Spirakis[c], Jacobo Toran[a]

[a] *Departament de Llenguatges i Sistemes, Universitat Politècnica Catalunya, Pau Gargallo 5, 08028-Barcelona, Spain*
[b] *Department of Computer Science, University of Warwick, Warwick, UK*
[c] *Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece*

Abstract

The minimum cut and minimum length linear arrangement problems usually occur in solving wiring problems and have a lot in common with job sequencing questions. Both problems are NP-complete for general graphs and in P for trees. We present here two parallel algorithms for the CREW PRAM. The first solves the minimum length linear arrangement problem for trees and the second solves the minimum cut arrangement for trees. We prove that the first problem belongs to NC for trees, and the second problem is in NC for bounded degree trees. To the best of our knowledge, these are the first parallel algorithms for the minimum length and the minimum cut linear arrangement problems.

1. Introduction

Given a graph $G = (V,E)$ with $|V| = n$, a *layout* of G is a one-to-one mapping φ from V to the first n integers $\{1,2,\dots,n\}$. The term layout is also known as *linear arrangement* [14, 13]. Notice that a layout φ on V determines a linear ordering of the vertices. Given a natural i, the *cut* of the layout at i is the number of edges that cross over i, i.e. the number of edges $\{u,v\} \in E$ with $\varphi(u) < i \leqslant \varphi(v)$. The *cutwidth* of φ, denoted by $\gamma(\varphi,G)$, is the maximum cut of φ over all integers from 1 to n. The *length* of φ, denoted by $\lambda(\varphi,G)$, is the sum over all edges (u,v) of $|\varphi(u)-\varphi(v)|$.

Graph layout problems are motivated as simplified mathematical models of VLSI layout. We can model a VLSI circuit by means of a graph, where the edges of the

* Corresponding author. E-mail: pantziou@cti.gr.

[1] A preliminary version of this paper was presented at *COCOON '95* [5]. This research was supported by the EEC ESPRIT Basic Research Project ALCOM II (contract no. 7141). The work of the third and the fifth author was also supported by the EEC ESPRIT Project GEPPCOM (contract No. 9072).

PII S0304-3975(96)00274-5

graph represent the wires, and the vertices represent the modules. Of course, this graph is an oversimplified model of the circuit, but understanding and solving problems in this simple model can help us to obtain better solutions for the real-world model (see the surveys by Shing and Hu [12], and Díaz [4]).

In this paper we shall consider two layout problems. The first problem is called *the minimum linear arrangement* (*MINLA*) problem. Given a graph $G = (V, E)$, find the layout φ which minimizes $\lambda(\varphi, G)$. The MINLA problem is NP-complete for general graphs [8]. Due to the importance of the problem, there has been some work trying to obtain polynomial-time algorithms for particular types of graphs. For instance, Even and Shiloach proved that the problem remains NP-complete for bipartite graphs [6]. Adolph and Hu gave an $O(n \log n)$ algorithm for the case that the graph is a rooted tree, where n is the size of the tree [2]. Finally, Shiloach solved the problem for undirected trees by an $O(n^{2.2})$ algorithm [13].

The second problem that we shall consider is *the minimum cut linear arrangement* (*MINCUT*) problem. Given a graph $G = (V, E)$, find the layout φ that minimizes the cutwidth $\gamma(\varphi, G)$. An important special case of this problem is *the graph bisection problem*; find a partition of $2n$ vertices into two subsets of size n such that the cutwidth between the two subsets is minimized. The MINCUT problem is NP-complete for general graphs [7], weighted trees and planar graphs [11]. The *graph bisection problem* is also NP-complete [8]. As in the case on the MINLA, the MINCUT has a history of results for particular types of graphs. Harper gave a polynomial-time algorithm for the n-dimensional hypercube [9]. Chung et al. [3] presented an $O(n(\log n)^{d-2})$ time algorithm to solve the MINCUT problem on trees, where d is the maximum degree of any vertex in the tree. Yannakakis gave an $O(n \log n)$ algorithm for the case that the graph is an undirected tree [14].

We present here two parallel algorithms. The first one solves the MINLA for undirected trees in $O(\log^2 n)$ time using $O(n^2 3^{\log n})$ processors on a CREW PRAM. The second algorithm solves the MINCUT for undirected trees of maximum degree d in time $O(d \log^2 n)$ using $O(n^2/\log n)$ CREW PRAM processors. To the best of our knowledge, these are the first parallel algorithms for the above problems.

2. A parallel algorithm for the MINLA problem on trees

2.1. Preliminaries

Let φ be a layout of a tree T of n vertices. φ is a *minimum length layout* of T if there is no other layout with smaller length. Let $\overline{\varphi}$ denote the layout obtained by reversing the order of the vertices. Note that $\lambda(\varphi, T) = \lambda(\overline{\varphi}, T)$.

Let v be a vertex of T. Deleting v and its incident edges from T, yields several subtrees of T. Each of them is called a *subtree of* $T \bmod v$. For each edge (v', v) there is a unique subtree T' of $T \bmod v$ such that $v' \in T'$. The vertex v' is *the root of* $T' \bmod v$.

Definition 1. A central vertex of T is a vertex v_* such that if $T_0, T_1, \ldots, T_k$ are all the subtrees of $T \bmod v_*$, then the number of vertices in each T_i, for $i = 0, 1, \ldots, k$, is at most $\lfloor n/2 \rfloor$.

In [13] has been proved that for each tree T, there exists a central vertex v_*.

Let T_0 be a subtree of $T \bmod v$, and let v_0 be its root $\bmod v$. Assume that we want to compute a minimum length layout of T. Computing minimum length layouts of T_0 and $T - T_0$ separately is wrong since we have no control on the length of the edge (v_0, v). In order to take into account this edge we consider *right* and *left anchored trees*. Let T be an n-vertex tree, let $v \in T$, and let φ be a layout of T. T is called *right anchored* at v, and is denoted by $\overrightarrow{T}(v)$ when its length is defined by $\lambda(\varphi, \overrightarrow{T}(v)) = \lambda(\varphi, T) + n - \varphi(v)$. T is called *left anchored* at v, and is denoted by $\overleftarrow{T}(v)$ when its length is defined by $\lambda(\varphi, \overleftarrow{T}(v)) = \lambda(\varphi, T) + \varphi(v) - 1$. In other words, in the length definition of a layout for a right (left) anchored tree, we consider an extra edge that covers the distance between v and the rightmost (resp., leftmost) vertex of T. Notice that finding a minimum length layout for right and left anchored trees is equivalent, since by reversing the order of the vertices a right anchored tree becomes a left anchored tree, while the total length remains unchanged. When considering all the subtrees $\bmod v$, all the anchored subtrees will be anchored at their root $\bmod v$. In such a case we will not state explicitly the corresponding root.

In the following, we use $T(\alpha)$ to denote a tree, with $\alpha = 0$ for free trees and $\alpha = 1$ for anchored trees. Further, $\lambda(T, v, \alpha)$ denotes the minimum length of a layout for $T(\alpha)$ where v is either the vertex at which the anchor is connected to T, if $\alpha = 1$, or a central vertex of T, if $\alpha = 0$. In both cases, we refer to the vertex v as the *root* of the tree.

Let $T_1, \ldots, T_k$ be trees, and let n_i denote the number of vertices of T_i, $i = 1, \ldots, k$. In order to simplify notation we will use $(T_1(\alpha_1), \ldots, T_k(\alpha_k))$, where $\alpha_i = 0$ if T_i is a free tree, and $\alpha_i = 1$ if T_i is an anchored tree, for $1 \leqslant i \leqslant k$, to represent the layout φ obtained from the layouts of the subtrees T_i, $1 \leqslant i \leqslant k$, which are composed together in such a way that the following holds:

$$\sum_{i=1}^{j-1} n_i < \varphi(v) \leqslant \sum_{i=1}^{j} n_i$$

for all $v \in T_j$ and for all $1 \leqslant j \leqslant k$.

Let v be the root of a tree $T(\alpha)$, and let $T_0, T_1, \ldots, T_k$ be all the subtrees of $T(\alpha) \bmod v$. In the sequel, we will assume that the subtrees are numbered so that $n_0 \geqslant n_1 \geqslant \cdots \geqslant n_k$, where n_i denotes the size of T_i, $i = 0, 1, \ldots, k$. Furthermore, $T - \{T_1, \ldots, T_k\}$ denotes the tree obtained by removing the vertices of $T_1, \ldots, T_k$ and their incident edges from T.

We define $p(T, v, \alpha)$ as the value of the greatest integer p satisfying $n_i > \lfloor \frac{1}{2}(n_0 + 2) \rfloor + \lfloor \frac{1}{2}(n_* + 2) \rfloor$ for $i = 1, 2, \ldots, 2p - \alpha$, where $n_* = n - \sum_{i=0}^{2p-\alpha} n_i$ and n is the number

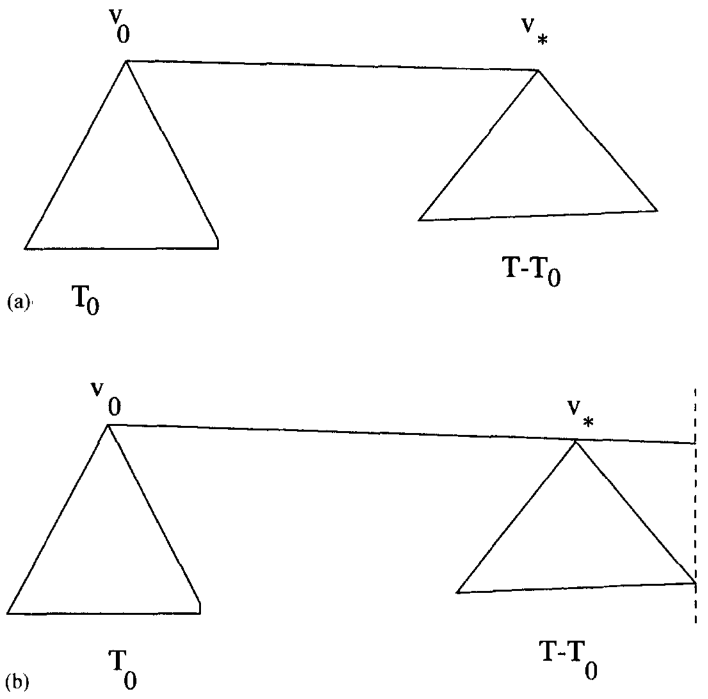

Fig. 1.

of vertices of T. If such a p does not exist, then we set $p(T,v,\alpha)=0$. We will denote by T_* the tree $T(\alpha)-\{T_1,\dots,T_{2p-\alpha}\}$, for $p=p(T,v,\alpha)$.

We state now the main result given in [13]. For motivation and more detailed discussion of the result, we refer the reader to the original paper by Shiloach [13].

Theorem 1 (Shiloach [13]). *Let $T(\alpha)$ be a tree with root v_*, and let $T_0,\dots T_k$ be all its subtrees* $\operatorname{mod} v_*$. *Let $p=p(T,v,\alpha)$ and $T_*=T-\{T_1,\dots,T_{2p-\alpha}\}$.*

(a) *If $p=0$ then, if $\alpha=0$ then $T(\alpha)$ has a minimum length layout of type $A=(\overrightarrow{T_0},\overleftarrow{T-T_0})$ (see Fig.* 1(*a*)), *with length*

$$\lambda(\varphi,T)=\lambda(\varphi,\overrightarrow{T_0}(v_0))+\lambda(\varphi,\overleftarrow{T-T_0}(v_*))+1$$

else (*i.e.*, $\alpha=1$) *$T(\alpha)$ has a minimum length layout φ of type $A=(\overrightarrow{T}_0,T-T_0)$ (see Fig.* 1(*b*)), *with length*

$$\lambda(\varphi,\overrightarrow{T})=\lambda(\varphi,\overrightarrow{T_0}(v_0))+\lambda(\varphi,T-T_0(v_*))+n-n_0,$$

where n is the number of vertices of $T(\alpha)$, and n_0 is the number of vertices of T_0.

(*b*) *If $p>0$ then $T(\alpha)$ has a minimum length layout of type A (defined as in case* (*a*)) *or of type $B=(\overrightarrow{T_1},\overrightarrow{T_3},\dots,\overrightarrow{T_{2p-1}},T_*,\overleftarrow{T_{2p-2\alpha}},\dots,\overleftarrow{T_4},\overleftarrow{T_2})$ (see Figs.* 2(*a*)

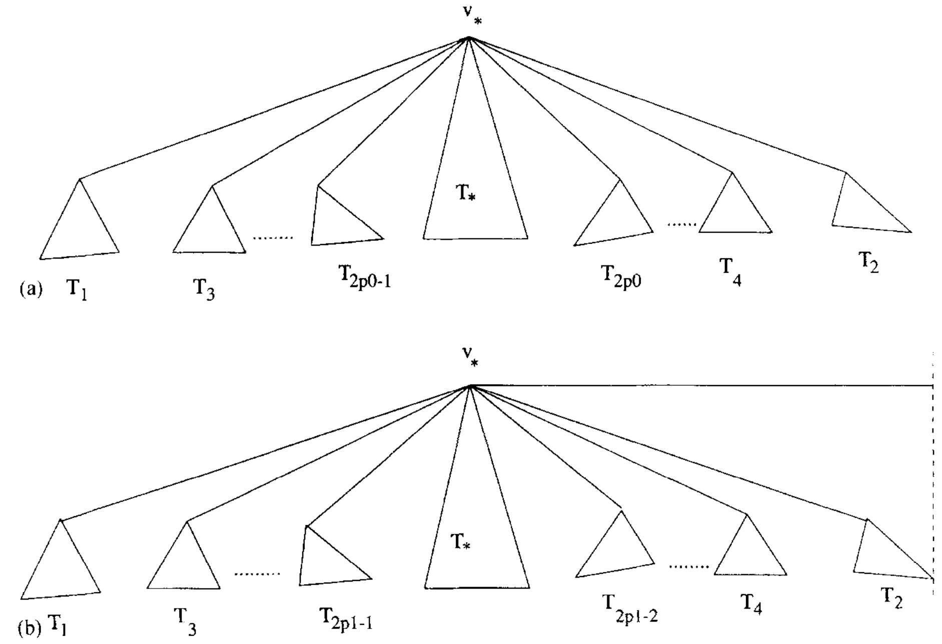

Fig. 2.

and (b)), with length

$$\lambda(\varphi, T(a)) = \sum_{i=1,\, i \text{ odd}}^{2p-1} \lambda(\varphi, \overrightarrow{T_i}(v_i)) + \sum_{i=1,\, i \text{ even}}^{2p-2\alpha} \lambda(\varphi, \overleftarrow{T_i}(v_i)) + \lambda(\varphi, T_*) + S_\alpha$$

where

$$S_0 = (n_3 + n_4) + 2(n_5 + n_6) + \cdots + (p-1)(n_{2p-1} + n_{2p}) + p(n_* + 1),$$

$$S_1 = (n_2 + n_3) + 2(n_4 + n_5) + \cdots + (p-1)(n_{2p-2} + n_{2p-1}) + p(n_* + 1) - 1,$$

with n_i being the number of vertices of T_i for $i \in \{0, \ldots, 2p-\alpha\}$, and $n_* = n - \sum_{i=0}^{2p-\alpha} n_i$.

Fact 1 (Shiloach [13]). (a) *If φ is a minimum length layout of $T(\alpha)$ of type A, then φ/T_0 (φ restricted to T_0) is a minimum length layout of $\overrightarrow{T_0}(v_0)$ and $\varphi/T - T_0$ (φ restricted to $T - T_0$) is a minimum length layout of $\overleftarrow{T - T_0}(v_*)$ (or $T - T_0$ if $\alpha = 1$).*

(b) *If φ is a minimum length layout of $T(\alpha)$ of type B, then φ/T_i (φ restricted to T_i) is a minimum length layout of $\overrightarrow{T_i}(v_i)$, for $i = 1, 3, \ldots, 2p-1$, and of $\overleftarrow{T_i}(v_i)$ for $i = 2, 4, \ldots, 2p-2\alpha$. φ/T_* is a minimum length layout of T_*.*

The design of the sequential algorithm is based on the decomposition given in Theorem 1. The correctness of the algorithm comes from Theorem 1 and Fact 1. Using the parameter α, the algorithms for free and anchored trees are combined together. Each of them recursively computes a minimum length layout of a tree T from minimum length layouts of its subtrees. Notice that if $p(T,v,\alpha) > 0$, then the algorithm computes both types of layouts and takes the one with the smaller length.

2.2. The parallel algorithm

Our parallel algorithm for the MINLA problem will be divided into two stages. In the first stage, starting with T, we recursively decompose each tree into subtrees until all subtrees have size one. At the same time, we keep the appropriate information that will allow us to compute a minimum length layout of each tree from minimum length layouts of its subtrees. In the second stage, we reconstruct the layouts, until we get a minimum length layout for the whole tree T. We only present the decomposition stage and specifically, the decomposition of both free and anchored trees. The reconstruction stage can be easily derived in view of the decomposition one.

The decomposition of free and anchored trees is based on Theorem 1 and exploits the properties of type A and B layouts, as well as the properties of the central vertices and the parameters $p(T,v,\alpha)$. We first prove the basic lemmas used for deriving the decomposition of free and anchored trees.

Lemma 1. *Let T be a free tree with central vertex v_*, and let $T_0,\dots T_k$ be all its subtrees* mod v_*. *Let $p = p(T,v_*,0)$, $n_* = n - \sum_{i=0}^{2p} n_i$, and $T_* = T - \{T_1,\dots,T_{2p}\}$. If $p>0$ then $|T_*| \leqslant n/2$, i.e., the number of vertices of T_* is at most $n/2$, where n is the number of vertices of T, and n_i is the number of vertices of T_i for $i \in \{0,\dots,k\}$.*

Proof. As $p \geqslant 1$, we have

$$n_{2p} > \left\lfloor \frac{n_0+2}{2} \right\rfloor + \left\lfloor \frac{n_*+2}{2} \right\rfloor = \left\lfloor \frac{n_0+2}{2} \right\rfloor + \left\lfloor \frac{n_{2p+1}+\cdots+n_k+3}{2} \right\rfloor$$

Thus, $2n_{2p} > n_0 + n_{2p+1} + \cdots + n_k + 1 = |T_*|$. Furthermore, $2p > 1$, thus $2n_{2p} \leqslant n_1 + \cdots + n_{2p}$. As $n_0 + n_1 + \cdots + n_k + 1 = |T|$, we get $|T_*| < n/2$. □

Lemma 2. *Let T be a free tree with central vertex v_*, and let T_0, T_1 be the two heaviest subtrees* mod v_*. *If $|T - \{T_0,T_1\}| \geqslant n/2$ then v_* is a central vertex of $T - \{T_0,T_1\}$, where n is the number of vertices of T.*

Proof. As the size of $T - \{T_0,T_1\}$ is $n - n_0 - n_1$ and, all n_i are sorted, we need only to show that $n_2 < \frac{1}{2}(n - n_0 - n_1)$ to prove that v_* is a central vertex of $T - \{T_0,T_1\}$.

Suppose that $n_2 > \frac{1}{2}(n-n_0-n_1)$, then $2n_2 > n-n_0-n_1 > n/2$. Thus $n_2 > n/4$. As n_2 is smaller than n_0 and n_1, we have that $n_0+n_1 > n/2$. But this implies $n-n_0-n_1 < n/2$, and we get a contradiction. □

Lemma 3. *Let T be tree anchored at v_*, and let $T_0,\dots,T_k$ be all its subtrees* mod v_*. *Let $p = p(T,v_*,0)$, and $T_* = T - \{T_1,\dots,T_{2p-1}\}$. Then, if $p>0$ then $|T_*| \leqslant 3n/4$, where n is the number of vertices of T.*

Proof. Let $n_b = |T_*|$. We consider two cases, depending on whether $n_0 \geqslant n/2$ or not.

Case 1. $n_0 < n/2$: In the case that $p > 1$ we have to remove at least three subtrees from T, and using the same argument as in Lemma 1 we get $n_b \leqslant n/2$. When $p = 1$, we have $n_1 > \frac{n-n_1}{2}$ that is $n_1 > n/3$. As $n_b = n - n_1$ we get $n_b \leqslant 2n/3$.

Case 2. $n_0 \geqslant n/2$: Notice that in this case $p < 2$ because otherwise, n_1, n_2, n_3 must be bigger than $n_0/2$ (from the definition of p) that is, bigger than $n/4$ and thus, n_0 could not be bigger than $n/2$. When $p = 1$ we have that $n_1 > n_0/2$ and therefore, $n_b \leqslant 3n/4$. □

The decomposition stage consists of a number of phases. The size of the trees which are decomposed during phase i is at most $n/(4/3)^i$. Therefore, in $O(\log n)$ phases we have trees of size one.

Free tree decomposition. The idea of the decomposition in the case of a free tree, is as follows. Let T be a free tree of size n, and let v_* be its central vertex. According to Theorem 1, if $p(T, v_*, 0) = 0$ then T has a minimum length layout of type A, while if $p(T, v_*, 0) > 0$, then a minimum length layout of T is computed as the minimum of type A and type B layouts. To simplify our discussion, we will consider the case that $p(T, v_*, 0) > 0$. (This case is more general in the sense that both type A and B layouts should be computed.) Notice that all subtrees that appear in the type B layout have size smaller than $n/2$ (T_i by the central vertex property and T_* by Lemma 1). Therefore, the size of the problem is reduced by a constant factor and in $O(\log n)$ phases we will have trees of constant size. In the case of the type A layout, if $T - T_0$ has size at most $n/2$ then the size of the problem is reduced by a constant factor. Otherwise, we have to further decompose the tree $\overleftarrow{T - T_0}(v_*)$ in the current phase. Notice that the root is still v_*, therefore, the largest subtree is T_1. Now a layout of type B again verifies the properties according to Lemma 3. In the case of the type A layout, if $T - \{T_0, T_1\}$ has size at most $n/2$ we are done, i.e., the size of the problem has been reduced. Otherwise, we have to further decompose $T - \{T_0, T_1\}$. But from Lemma 2, v_* is still a central vertex for this tree thus, its subtrees $\bmod v_*$ are $T_2, \ldots, T_k$. The above procedure should be repeated until the size of the derived subtrees is less than or equal to $n/2$. Note that the procedure should be repeated for at most β times, where β is the first index for which $|T - \{T_0, \ldots, T_\beta\}| \leqslant n/2$.

Before we give the algorithm for the free tree decomposition we need the following definition.

Definition 2. Let $T_1, \ldots, T_k$ be subtrees of $T(\alpha)$. Let $T_b = T(\alpha) - \{T_1, \ldots, T_k\}$, $n_i = |T_i|$, for $i \in \{1, \ldots, k\}$ and $n_1 \geqslant n_2 \geqslant \cdots \geqslant n_k$. A *balanced layout* of the subtrees $T_1, \ldots, T_k, T_b$ is the layout

$$(\overrightarrow{T_1}, \overrightarrow{T_3}, \overrightarrow{T_5}, \ldots, T_b, \ldots, \overleftarrow{T_6}, \overleftarrow{T_4}, \overleftarrow{T_2}).$$

If φ is a balanced layout of $T_1, \ldots, T_k, T_b$, and $n_b = |T_b|$, then the length of φ is computed as in the following lemma. We assume that k is even (the case that k is odd is similar).

Lemma 4. *Let $T_1,\dots,T_k$ be subtrees of $T(\alpha)$ and $T_b = T(\alpha) - \{T_1,\dots,T_k\}$. If φ is a balanced layout of $T_1,\dots,T_k,T_b$ then*

$$\lambda(\varphi, T(a)) = \sum_{i=1, i \text{ odd}}^{k-1} \lambda(\varphi, \overrightarrow{T_i}) + \sum_{i=1, i \text{ even}}^{k} \lambda(\varphi, \overleftarrow{T_i}) + \lambda(\varphi, T_b) + Q_\alpha$$

where

$$Q_0 = (n_3 + n_4) + 2(n_5 + n_6) + \cdots + (k/2 - 1)(n_{k-1} + n_k) + k/2(n_b + 1),$$

$$Q_1 = (n_2 + n_3) + 2(n_4 + n_5) + \cdots + (k/2 - 1)(n_{k-2} + n_{k-1})$$
$$+ k/2(n_k + n_b + 1) - 1.$$

Proof. The proof is by a straightforward calculation using the subtree sizes, which follows from the definition of a balanced layout, and the elementary definitions of the length of free and anchored tree layouts. □

Fact 2. *If φ is a minimum length layout of $T(\alpha)$ computed as a balanced layout of $T_1,\dots,T_k,T_b$, then φ/T_i (φ restricted to T_i) is a minimum length layout of $\overrightarrow{T_i}$ for $i = 1,3,\dots,k-1$, and of $\overleftarrow{T_i}$ for $i = 2,4,\dots,k$. φ/T_b is a minimum length layout of T_b.*

Proof. The proof is immediate since Q_α for $\alpha \in \{0,1\}$, is independent of φ. □

The decomposition of a free tree T of size n, is obtained as follows.

1. Compute a central vertex v_* of T.
2. Compute in parallel the sizes of the subtrees of $T \bmod v_*$. Let $T_0,\dots,T_k$ be the subtrees of $T \bmod v_*$, and $|T_i| \geqslant |T_{i+1}|$, for all $i \in \{0,\dots,k-1\}$.
3. Compute β the first index for which $|T - \{T_0,\dots,T_\beta\}| \leqslant n/2$.
4. Let $p_0 = p(T, v_*, 0)$ and B_0 be the layout $(\overrightarrow{T_1}, \overrightarrow{T_3}, \dots, \overrightarrow{T_{2p_0-1}}, T_*, \overleftarrow{T_{2p_0}}, \dots, \overleftarrow{T_4}, \overleftarrow{T_2})$.
5. For each $i = 1,\dots,\beta$
   - 5.1. Compute in parallel $p_i = p(T - \{T_0,\dots,T_{i-1}\}, v_*, \alpha_i)$, where $\alpha_i = 0$ for i even, and $\alpha_i = 1$ for i odd. If $p_i = 0$ then B_i is empty. Otherwise, B_i is computed in step 5.3.
   - 5.2. Let $T_*^i = T - \{T_0,\dots,T_{i-1},T_{i+1},\dots T_{i+2p_i-\alpha_i}\}$, where $\alpha_i = 0$ for i even, and $\alpha_i = 1$ for i odd.
   - 5.3. Let B_i be the balanced layout of $T_0,\dots,T_{i-1},T_{i+1},\dots T_{i+2p_i-\alpha_i}, T_*^i(\alpha_i)$ (see Definition 2).
6. Let $T_*^{\beta+1} = T - \{T_0, T_1,\dots,T_\beta\}$. Let $B_{\beta+1}$ be the balanced layout of $T_0, T_1,\dots,T_\beta$, $T_*^{\beta+1}(\alpha_{\beta+1})$, where $\alpha_{\beta+1} = 1$ if β is even, and $\alpha_{\beta+1} = 0$ if β is odd.

Let T be a free tree, and v_* be a central vertex of T. Let $T_0,\dots,T_k$ be all the subtrees of $T \bmod v_*$. Let β, B_i, for $i \in \{0,1,\dots,\beta+1\}$, be as they are defined above. Then we have the following.

Lemma 5. *A minimum length layout of T can be computed as the layout out of B_i, $i = 0, \dots, \beta + 1$, which attains minimum length. Furthermore, the size of all subtrees that appear in any such layout is at most $3n/4$.*

Proof. Let T be a free tree of size n. Notice that all subtrees that appear in the type B layout have size smaller than $n/2$, T_i by the central vertex property and T_* by Lemma 1. This layout is B_0. If $T - T_0$ has size at most $n/2$ we are done, because $\beta = 0$. Note that the size of each subtree in B_0 is at most $n/2$. Otherwise, we have to decompose the tree $\overleftarrow{T - T_0}(v_*)$. Notice that the root is still v_*, therefore, the largest subtree is T_1. Now a layout of type B again verifies the properties according to Lemma 3. Furthermore, putting T_0 on the left it corresponds to layout B_1. The size of each subtree in B_1 is at most $3n/4$. If $T - \{T_0, T_1\}$ has size at most $n/2$ we are done; layout $B_{\beta+1} = B_2$ is the balanced layout of $T_0(1), T_1(1), T - \{T_0, T_1\}$. Notice again that the size of each subtree in $B_{\beta+1}$ is at most $n/2$. Otherwise, we have to further decompose $T - \{T_0, T_1\}$. But, from Lemma 2, v_* is still a central vertex for this tree therefore, its subtrees $\bmod v_*$ are $T_2, \dots, T_k$. The procedure is repeated for at most β times, and the layouts B_i, for $i = 0, 1, \dots, \beta + 1$, are obtained. Notice that the subtrees that appear in the above layouts have size at most $3n/4$. Notice also that the layouts B_i for $i = 0, 1, \dots, \beta + 1$, have been obtained by simply combining type A and type B layouts of subtrees of $T \bmod v_*$, in a way that the sequential algorithm would follow. Therefore, a minimum length layout of T can be correctly computed as the layout out of B_i, for $i = 0, 1, \beta + 1$, which attains minimum length. The lengths of these layouts can be computed using Theorem 1 and Lemma 4. □

Anchored tree decomposition. Before we give the anchored tree decomposition we introduce some additional notation. Let $\overrightarrow{T}(v_*)$ be an anchored tree of size n. We will denote by $T_0^0, T_1^0, \dots$ its subtrees $\bmod v_*$ sorted by size, and by $v_0^0, v_1^0, \dots$ the corresponding roots $\bmod v_*$. Recursively, whenever we have an anchored tree $\overrightarrow{T_0^i}(v_0^i)$, by $T_0^{i+1}, T_1^{i+1}, \dots$ we will denote its subtrees $\bmod v_0^i$, and by $v_0^{i+1}, v_1^{i+1}, \dots$ we will denote their corresponding roots $\bmod v_0^i$.

The idea of the decomposition of $\overrightarrow{T}(v_*)$ is based on the following. If $n_0 = |T_0^0| \leqslant \lfloor n/2 \rfloor$ then v_* is a central vertex and $\overrightarrow{T}(v_*)$'s decomposition is similar to the decomposition of a free tree. Consider now the case where $n_0 > \lfloor n/2 \rfloor$. According to Theorem 1, if $p(\overrightarrow{T}, v_*, 1) = 0$ then $\overrightarrow{T}$ has a minimum length layout of type A, while if $p(\overrightarrow{T}, v_*, 1) > 0$ then a minimum length layout of $\overrightarrow{T}$ is computed as the minimum of type A and type B layouts. As in the case of the free tree decomposition, to simplify our discussion we will consider the case that $p(\overrightarrow{T}, v_*, 1) > 0$. From Lemma 3, since $n_0 \geqslant n/2$, $p(\overrightarrow{T}, v_*, 1)$ cannot be bigger than 1. Therefore, the two trees that appear in the type B layout have size smaller than $3n/4$ (T_1^0 has size smaller than T_0^0, and $T_* = T - T_1^0$ by Lemma 3). Therefore, the size of the problem is reduced by a constant factor and in $O(\log n)$ phases we may have trees of constant size. In the case of the type A layout, note that $T - T_0^0$ has size at most $n/2$. If T_0^0 has also size smaller than $n/2$, then the size of the problem is reduced by a constant factor. Otherwise, we have to further decompose the tree $\overleftarrow{T_0^0}$ in the current phase. Now a layout of type B again verifies the properties according to Lemma 3. In the case of the type A layout, if $T_0^1(1)$ (i.e. the anchored tree T_0^1) has size smaller than $n/2$ we are done, i.e., the size of the problem has been reduced. Otherwise, we have to further decompose $T_0^1(1)$. Note that the above procedure should be repeated for at most γ times, where γ is the first index for which $|T_0^\gamma| < n/2$.

Therefore, the decomposition of an anchored tree $\overrightarrow{T}(v_*)$ is obtained as follows.

1. Compute γ the first index for which $|T_0^\gamma| < n/2$.
2. If $p(\overrightarrow{T}, v_*, 1) = 1$ then let $\Gamma_0 = (\overrightarrow{T_1}^0, T - T_1^0)$
3. For each $i = 1, \ldots, \gamma$:
   - 3.1. Compute in parallel, $p_i = p(T_0^i, v_0^i, 1)$. If $p_i = 0$ then Γ_i is empty. Otherwise, Γ_i is computed in step 3.2.
   - 3.2. Let Γ_i be the layout $(\overrightarrow{T_1^i}(v_1^i), T_0^{i-1} - T_1^i, \ldots, T_0^0 - T_0^1, T - T_0^0)$.
4. Let $\Gamma_{\gamma+1}$ be the layout $(\overrightarrow{T_0^\gamma}, T_0^{\gamma-1} - T_0^\gamma, \ldots, T_0^0 - T_0^1, T - T_0^0)$. In the case that the size of $T_0^{\gamma-1} - T_0^\gamma$ is bigger than $3n/4$ apply the free decomposition on it, until the size of each of $T_0^{\gamma-1} - T_0^\gamma$'s subtrees is smaller than $3n/4$.

Let $\overrightarrow{T}(v_*)$ be an anchored tree and let γ, Γ_i, for $i \in \{0, 1, \ldots, \gamma + 1\}$, be as they are defined above. Notice that all subtrees appearing in the Γ_i layouts, for $i = 0, \ldots, \gamma$, have size smaller than $3n/4$. The only tree that can have big size in the $\Gamma_{\gamma+1}$ layout, is $T_0^{\gamma-1} - T_0^\gamma$. In such a case we decompose it according to Lemma 5, taking as a parameter the size of the original tree.

Lemma 6. *A minimum layout for an anchored tree T can be computed as the layout out of Γ_i, $i = 0, 1, \ldots \gamma + 1$, which attains minimum length.*

The proof of the previous lemma follows from the above discussion. The length of each layout Γ_i, for $i = 0, 1, \ldots, \gamma + 1$, can be easily computed from the lengths of the layouts of the subtrees that appear at it. Notice that the extra length that should be added due to the anchors, depends only on the size of the subtrees.

We can now prove the following theorem.

Theorem 2. *Given an undirected tree T with n vertices, there exists a parallel algorithm that computes a minimum length layout of T. The algorithm needs* $O(\log^2 n)$ *time using* $O(n^{3.6})$ CREW PRAM *processors, where n is the size of the tree.*

Proof. The algorithm consists of two stages. In the first stage (decomposition stage), the tree is decomposed into subtrees of size one. The decomposition stage consists of $O(\log n)$ phases. At phase i, a number of subtrees of size at most $n/(\frac{4}{3})^i$ are decomposed into a number of subtrees of size at most $n/(\frac{4}{3})^{i+1}$. (At phase 0 the whole tree T is decomposed into subtrees of size at most $3n/4$.) A free tree is decomposed using the free tree decomposition procedure, and an anchored tree using the anchored tree decomposition procedure. During the decomposition phases, we keep the necessary information that will be used to compute a minimum length layout of a tree from minimum length layouts of its subtrees. Therefore, appropriate expressions that keep the lengths of layouts B_i and Γ_i (as they are defined in the free and anchored tree decomposition procedures) are kept. Those expressions will be used at the second stage of the algorithm (reconstruction stage) to compute the minimum length layout of T, by first computing minimum length layouts of the subtrees constructed at the decomposition stage. Notice that the reconstruction stage consists also of $O(\log n)$ phases. The correctness of the algorithm follows from Theorem 1, Fact 2, and Lemmas 5 and 6.

Before we discuss the implementation details of our algorithm, as well as its complexity, we first discuss the way the trees are represented. Each tree will be represented by a linked list keeping an Euler tour representation, together with a mask that keeps which vertices of the original tree are present in it. This mask will also contain pointers to the linked list. To distinguish between free and anchored trees we keep the parameter α and the corresponding root for anchored trees. We record in a matrix, pointers to the subtree masks that form part of any of the layouts in a decomposition phase, together with the additional information required to trace back the length of any layout.

A central vertex of a free tree T_m of m vertices, can be computed in time $O(\log m)$ using $O(m^2)$ CREW PRAM processors. This is done as follows. We compute for each edge the sizes of the two subtrees using the Euler tour technique [10], and compute the difference of subtree sizes. We take as central vertex the root of the heaviest subtree corresponding to an edge of minimum difference.

Once we have the central vertex of a free tree, we have to compute subtree sizes (now the tree is rooted) using the Euler tour technique, and sort subtrees by size. From the tree sizes using suffix sums we compute $\beta, p_0, p_1, \ldots, p_\beta$. Consider the free tree decomposition. There are two ways to create new subtrees at each decomposition phase of a free tree. First, trees obtained just removing an edge, i.e., all subtrees $\bmod v$ for a given root v. Second, the union of some of the subtrees $\bmod v$ rooted at a "new" copy of v. In the first case, we compute the corresponding subtrees by removing the root and running a rooting algorithm (in parallel) for each root $\bmod v$, in order to separate subtrees. This part will be the basic step for the second case; now we just have to merge the corresponding trees adding a new vertex as root. So for a tree of size m, we can maintain the tree representation using CREW PRAM $O(m^2)$ processors in time $O(\log m)$.

In the case of an anchored tree T_m, the root of T_m is the vertex in which the anchor is connected to T. We first compute subtree sizes using the standard Euler tour technique, and then again with the same technique, we find a path of roots of trees of maximum cardinality. Finally, using suffix sums we compute the index γ. From the anchored tree decomposition, it is easy to compute the representation of each subtree.

Taking into account that the sum of the sizes of the trees obtained in the decomposition of a tree T_m is at most 3 times the number of vertices of T_m, the number of processors needed in any phase is at most 3 times the number of processors in the previous one. Thus, the maximum number of processors needed by the algorithm is $O(3^{\log n} n^2) = O(n^{3.6})$. Furthermore, the time used in each phase in the first and second stage is $O(\log n)$. Thus, the algorithm needs $O(\log^2 n)$ time and $O(n^{3.6})$ CREW PRAM processors. $\square$

3. A parallel algorithm for the MINCUT problem on trees

In this section we give an $O(n^2/\log n)$-processor, $O(d \log^2 n)$-time parallel algorithm which finds a minimum cut layout of a tree T of maximum degree d, where n is the number of vertices of T.

For each vertex $v \in T$, the parallel algorithm proceeds as follows: Let T_v be the tree that contains all the vertices of T and is rooted at v. The algorithm converts T_v into a binary tree T_{v0} and then applies the parallel tree contraction technique to T_{v0} to compute a minimum cut layout of T_v. Let φ_v be a minimum cut layout of T_v. The layout φ_v, $v \in T$, with the minimum cutwidth is output as a minimum cut layout of T.

Before we describe the algorithm we give some terminology and definitions.

3.1. Terminology and definitions

Let T be a tree which we convert into a binary one T_0. Let v be a vertex of T with degree d and let $w_1, \ldots, w_d$ be its children. Then, the vertex set of T_0 includes vertices $v^1, \ldots, v^{d+1}$. For $1 \leqslant i \leqslant d$, v^{i+1} is the right child of v^i in T_0 (see Fig. 3). We will say that the vertices $v^i, 1 \leqslant i \leqslant d$, are of the *same label* since they are *coming from* the same vertex of T (e.g., in Fig. 3 the vertices v^2 and v^{d+1} are of the same label while w_d^1 and v^2 are not).

With each vertex $u \in T$, we associate two pieces of information: (i) A layout sequence, φ_u, realizing the layout of the subtree rooted at u and u's position in this layout and (ii) a cost-sequence, $cost(\varphi_u)$ of the layout sequence φ_u defined in the sequel.

(In the sequel, we will use φ_u to denote both the layout of a tree T_u rooted at u, and the layout sequence realizing the layout.)

Given a layout φ_u for the subtree rooted at u, $\langle leftcost(\varphi_u)\rangle$ is a sequence $\langle \gamma_1, \eta_1, \gamma_2, \eta_2, \ldots\rangle$ where parameters γ_i and η_i are defined as follows: γ_1 is the largest cut (in φ_u) occurring on the left side of u. Let w_1 be the point where the cut of γ_1 occurs. If w_1 is immediately to the left of u then $\langle leftcost(u)\rangle = \langle \gamma_1 \rangle$. Otherwise, let η_1 be the smallest cut between w_1 and u and let w_2 be the point closest to u where η_1 occurs. Suppose that γ_2 is the maximum cut between w_2 and u and w_3 is the point closest to u where γ_2 occurs. If $\gamma_2 = \eta_1$ or w_3 is immediately to the left of u then $\langle leftcost(u)\rangle = \langle \gamma_1, \eta_1, \gamma_2\rangle$. Otherwise, we continue similarly by taking the smallest cut between w_3 and u. $\langle rightcost(u)\rangle$ is a sequence $\langle \gamma'_1, \eta'_1, \gamma'_2, \ldots\rangle$ where γ'_1 is the largest cut in φ_u occuring on the right side of u. The rest of the sequence is defined in a way similar to that of $\langle leftcost(u)\rangle$ but we now work on the right side of u. Clearly,

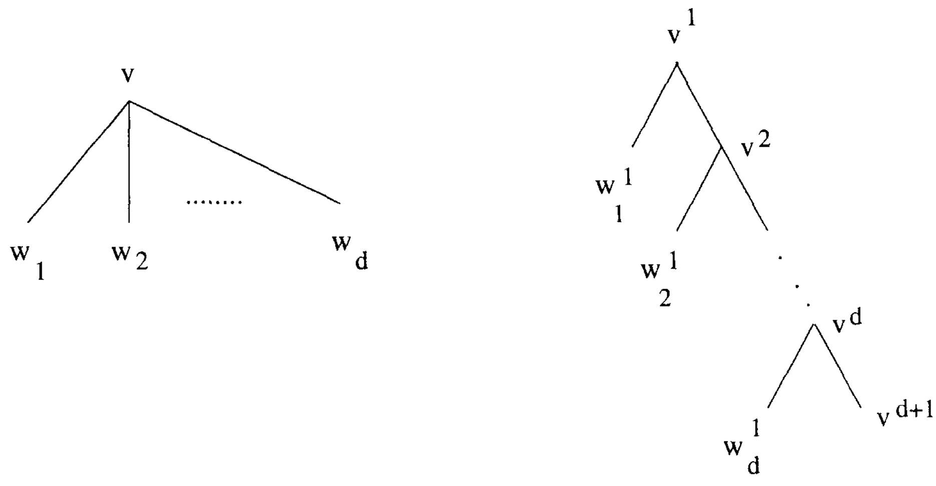

Fig. 3.

$\gamma_1 \geqslant \gamma_2 \geqslant \cdots$, $\eta_1 \leqslant \eta_2 \leqslant \cdots$, $\gamma'_1 \geqslant \gamma'_2 \geqslant \cdots$, $\eta'_1 \leqslant \eta'_2 \leqslant \cdots$. Also, $\gamma_1 \geqslant \eta_1$, $\gamma_2 \geqslant \eta_2$, etc., and $\gamma'_1 \geqslant \eta'_1$, $\gamma'_2 \geqslant \eta'_2$, etc.

When $\gamma_1 = \gamma'_1$ we say that the layout φ_u is *balanced*; otherwise, it is *unbalanced*.

Definition 3. Let φ_u be a layout sequence and let γ_1, γ'_1, $leftcost(\varphi_u)$, $rightcost(\varphi_u)$ be as they are defined above. Then the cost of the layout φ_u is $\gamma_u = \max\{\gamma_1, \gamma'_1\}$, and the *cost-sequence* of the layout φ_u is $cost(\varphi_u) = (leftcost(\varphi_u), *, rightcost(\varphi_u))$ where the "$*$" denotes the position of u.

The algorithm involves comparisons of cost sequences in order to construct a minimum cut layout of a subtree rooted at v using the minimum cut layouts of subtrees rooted at v's children. This is motivated by the fundamental work of Yannakakis [14]. In the sequel, we describe how we compare cost sequences.

Let a and b be the two subsequences of a cost sequence *cost*. If $a \neq b$, and neither is a prefix of the other, then $a > b$ iff a is lexicographically larger than b. If a is a prefix of b and a ends with a γ_i entry, then $a > b$, while if a ends with a η_i entry, then $a < b$. If $leftcost(\varphi_u) > rightcost(\varphi_u)$ then we call the left side of φ_u (with respect to the position of u) heavy side, and the *right* side of u *light* side.

Let $cost_1 = (heavyside_1, *, lightside_1)$, $cost_2 = (heavyside_2, *, lightside_2)$ be two cost sequences corresponding to two layouts for the same tree. Let $heavyside_i = \gamma_{1i}, \eta_{1i}, \gamma_{2i}, \eta_{2i}, \ldots$ and $lightside_i = \gamma'_{1i}, \eta'_{1i}, \gamma'_{2i}, \eta'_{2i}, \ldots$, for $i \in \{1, 2\}$. To compare the cost sequences $cost_1$ and $cost_2$, we construct the sequence $compare_i = \langle \gamma^c_{1i}, \eta^c_{1i}, \gamma^c_{2i}, \eta^c_{2i}, \ldots \rangle$, for $i \in \{1, 2\}$, as follows: If $\gamma_{1i} \neq \gamma'_{1i}$ then $\gamma^c_{1i} = \gamma_{1i}$ and $compare_i = \langle \gamma^c_{1i} \rangle$. If $\gamma_{1i} = \gamma'_{1i}$ and there are no next entries η_{1i}, η'_{1i} in $heavyside_i$, $lightside_i$, respectively, then let $\gamma^c_{1i} = \gamma_{1i}$ and $compare_i = \langle \gamma^c_{1i}, \gamma^c_{1i} \rangle$. If only one of the $heavyside_i$ or $lightside_i$ has an entry following γ_{1i} or γ^c_{1i} then call that entry η^c_{1i} and let $compare_i = \langle \gamma^c_{1i}, \eta^c_{1i} \rangle$. If $\eta_{1i} \neq \eta'_{1i}$ then let

$\eta^c_{1i} = \eta'_{1i}$ and $compare_i = \langle \gamma^c_{1i}, \eta^c_{1i} \rangle$. If $\eta_{1i} = \eta'_{1i}$ then in the case that $\gamma_{2i} = \eta_{1i}$ or $\gamma_{2i} \neq \gamma'_{2i}$, let $\eta^c_{1i} = \eta_{1i}$, $\gamma^c_{2i} = \gamma_{2i}$ and $compare_i = \langle \gamma^c_{1i}, \eta^c_{1i}, \gamma^c_{2i} \rangle$. In the case that $\gamma_{2i} = \gamma'_{2i}$, we continue in the same way as in the case (above) where $\gamma_{1i} = \gamma'_{1i}$.

We say that $cost_1 = cost_2$ iff $compare_1 = compare_2$. If the sequences $compare_1 \neq compare_2$ and neither is a prefix of the other, then $cost_1 \prec cost_2$ iff $compare_1$ is lexicographically smaller than $compare_2$. If $compare_1$ is a prefix of $compare_2$ and $compare_1$ is of odd length then $cost_1 \prec cost_2$ while if $compare_1$ is of even length then $cost_2 \prec cost_1$. Note that $\prec$ is a transitive relation.

Definition 4. Let T_u be a tree rooted at a vertex u, and let φ_u be a layout of T_u. φ_u is *optimal* iff there is no other layout φ'_u of T_u such that $cost(\varphi'_u) \prec cost(\varphi_u)$.

3.2. *The parallel algorithm*

The parallel tree-contraction algorithm (see [1]) evaluates the root of a tree T by processing a logarithmic number of binary trees $T_0, T_1, \ldots, T_k$, $k = O(\log|T|)$, where $T_0 = T$ and T_k contains only one vertex. Also, $|T_i| \leqslant \varepsilon |T_{i-1}|$, $0 < \varepsilon < 1$. 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 of our algorithm involves the construction of an optimal layout for a subtree T_v, rooted at a vertex v, using optimal layouts of the subtrees $T_{v_1}, \ldots, T_{v_d}$ rooted at v's children, $v_1, \ldots, v_d$. The correctness of the approach is based on the following fact proved by Yannakakis [14]. Let T be a tree that consists of two rooted trees T_1, T_2 rooted respectively at vertices v_1, v_2, and the edge $\{v_1, v_2\}$. Then the cutwidth of T depends on the cost-sequences $cost(\varphi_1)$, $cost(\varphi_2)$ of the optimal layouts φ_1 and φ_2 of T_1 and T_2, respectively. Furthermore, the cutwidth of T is a monotonic function of both $cost(\varphi_1)$ and $cost(\varphi_2)$, i.e., if we replace T_1 by another tree T'_1 with optimal layout φ'_1 and $cost(\varphi_1) \prec cost(\varphi'_1)$ or $cost(\varphi_1) = cost(\varphi'_1)$, and replace T_2 by another tree T'_2 with optimal layout φ'_2 and $cost(\varphi_2) \prec cost(\varphi'_2)$ or $cost(\varphi_2) = cost(\varphi'_2)$, then the cutwidth of the resulting tree T' is at least as large as the cutwidth of T.

The *shunt* operation involves two merge-operations on the layout sequences:

Merge-operation A: Let T_{uv} be a tree rooted at a vertex u. T_{uv} consists of two trees T_u, T_v (rooted at u, v respectively) and the edge $\{u, v\}$. Suppose that φ_u and φ_v are optimal layouts of T_u, T_v, respectively. The merge-operation A uses φ_u and φ_v to compute an optimal layout φ_{uv} of T_{uv}.

Merge-operation B: Let T_u be a tree rooted at u with children $u_1, \ldots, u_d$ and T_v a tree rooted at v with children $v_1, \ldots, v_{d'}$. Suppose that we are given the optimal layouts of T_u, T_v. The merge-operation B computes an optimal layout φ_{uv} realizing a layout of the tree T_{uv} which is rooted at u and has as children the children of both T_u and T_v.

We give now the shunt operation of the tree contraction technique. Suppose that l_i is the leaf which is ready to perform the *shunt* and that f_i is the father of l_i, $p(f_i)$ is the father of f_i and f_j is the other child of f_i. Suppose also that optimal layouts of $l_i, f_i, f_j, p(f_i)$ are given. For the sake of simplicity in the notation, assume that these layouts are denoted by $c(l_i), c(f_i), c(f_j), c(p(f_i))$, respectively. In the sequel, f_{ij} will

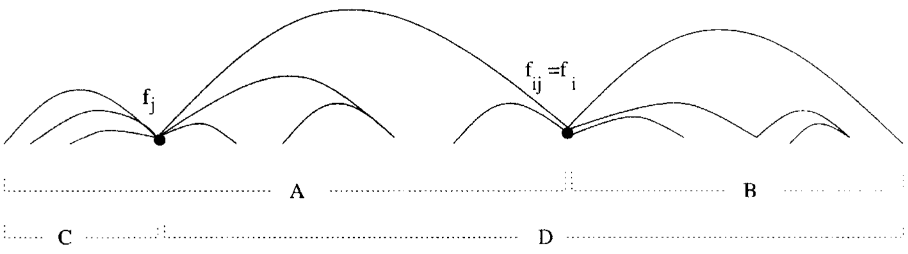

Fig. 4.

be the vertex which is the result of the *shunt* operation. A and B will denote the merge operations. We will use $A(\varphi_u, \varphi_v)$ $(B(\varphi_u, \varphi_v))$ to denote the layout sequence we get by applying the merge operation A (respectively, B) to the layout-sequences φ_u and φ_v.

Case 1. l_i is the left leaf of f_i. (l_i, f_j cannot be of the *same label*.)

$$c(f_il_i) = A(c(l_i), c(f_i))$$

Case 1.a. f_i, f_j are of the *same label*. $c(f_{ij}) = B(c(f_il_i)), c(f_j))$.

Case 1.b. f_i, f_j are not of the *same label*. $c(f_{ij}) = A(c(f_il_i)), c(f_j))$.

Case 2. l_i is the right leaf of f_i. (f_i, f_j cannot be of the *same label*.)

Case 2.a. l_i, f_i are of the *same label*. $c(f_il_i) = B(c(l_i), c(f_i))$

Subcase 2.a.a. f_i, $p(f_i)$ are also of the *same label*.

$$c(p(f_i)) = B(c(f_il_i), c(p(f_i)))$$

$$c(f_{ij}) = c(f_j).$$

Subcase 2.a.b. f_i, $p(f_i)$ are not of the *same label*.

$$c(f_{ij}) = A(c(f_il_i), c(f_j)).$$

Suppose that the resulting sequence $c(f_{ij})$ is as in the Fig. 4, i.e., $c(f_{ij}) = (A, *, B)$. From $c(f_{ij})$ we easily take the layout-sequence $c'(f_{ij}) = (C, *, D)$, where the "$*$" denotes the position of f_j (see Fig. 4). Let $T_{f_{ij}}$ be the subtree – of the current T_{v0} – rooted at f_{ij}. In the sequel, every merge operation of f_{ij} with a vertex w of $T_{f_{ij}}$ is done using the layout sequence $c'(f_{ij})$ while every merge operation of f_{ij} with a vertex of $T_{v0} - T_{f_{ij}}$ is done using the layout sequence $c(f_{ij})$.

Case 2.b. l_i, f_j are not of the *same label*. This case is similar to the above ones.

The efficient parallel implementation of the merge operations A considers a number of cases depending on whether the leaf that is going to perform the shunt is a right or a left leaf and whether or not it has the same label with its father.

Suppose that

$$\varphi_u = (x_1, x_2, \ldots, x_i, *, x'_{i1}, \ldots, x'_2, x'_1),$$

$$\varphi_v = (y_1, y_2, \ldots, y_j, *, y'_{j1}, \ldots, y'_2, y'_1)$$

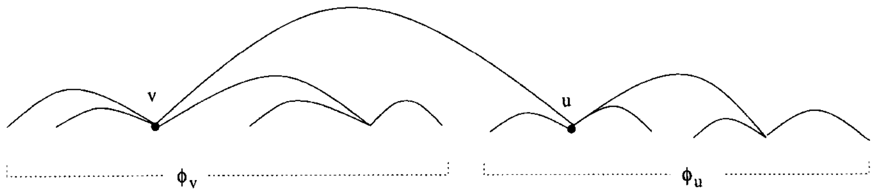

Fig. 5.

$$cost(\varphi_u) = (\gamma_{1u}, \eta_{1u}, \ldots, \gamma_{ku}, *, \gamma'_{lu}, \ldots, \eta'_{1u}, \gamma'_{1u}),$$

$$cost(\varphi_v) = (\gamma_{1v}, \eta_{1v}, \ldots, \gamma_{mv}, *, \gamma'_{nv}, \ldots, \eta'_{1v}, \gamma'_{1u})$$

and

$$\gamma_u = \max\{\gamma_{1u}, \gamma'_{1u}\}, \gamma_v = \max\{\gamma_{1v}, \gamma'_{1v}\}.$$

Merge-operation A

Case 1: $\gamma_u = \gamma_v$ and φ_u, φ_v balanced. In this case, it is clear that the cost of an optimal layout sequence φ_{uv} cannot be less than $\gamma_u + 1$.

Subcase 1.1. Suppose that *leftcost*$(\varphi_u) \leqslant$ *rightcost*(φ_u) and *rightcost*$(\varphi_v) \leqslant$ *leftcost* (φ_v) and $n_{1u} \neq 1$, $n'_{1v} \neq 1$. Then we construct φ_{uv} from φ_u, φ_v as follows (see Fig. 5):

$$\varphi_{uv} = (y_1, \ldots, y_j, y'_{j1} + 1, \ldots, y'_1 + 1, 1, x_1 + 1, \ldots, x_i + 1, *, x'_{i1}, \ldots, x'_1)$$

and the cost sequence of φ_{uv} is

$$cost(\varphi_{uv}) = (\gamma_{1u} + 1, \eta_{1u} + 1, \ldots, \gamma_{ku} + 1, *, \gamma'_{lu}, \ldots, \eta'_{1u}, \gamma'_{1u}).$$

The other subcases are similar.

Case 2: $\gamma_u > \gamma_v$ and φ_u balanced and φ_v unbalanced. Let H_u be the heavy side of u and L_u the light one. W.l.o.g. let $H_u =$ *leftcost*(φ_u) and $L_u =$ *rightcost*(φ_u). The procedure which gives the cost γ_{uv} of an optimal layout sequence φ_{uv}, is as follows:

if $\eta_{1u} + \gamma_v \leqslant \gamma_u$ or $\eta'_{1u} + \gamma_v \leqslant \gamma_u$ **then** $\gamma_{uv} = \gamma_u$ **else** $\gamma_{uv} = \gamma_u + 1$.

In order to find the position between the vertices of T_u where we will insert T_v, we proceed as follows. Let i be the index of the largest η_{iu} in H_u for which $\eta_{iu} + \gamma_v \leqslant \gamma_{iu} - 1$. Let j be the index of the largest η'_{ju} in L_u for which $\eta'_{ju} + \gamma_v \leqslant \gamma'_{iu}$. Suppose now that $c(h_{uv})$ ($c(l_{uv})$) is the layout sequence we take if we insert φ_v between the vertices – in φ_u – where the cut of η_{iu} (resp., η'_{ju}) occurs. Then, φ_{uv} is this one from $c(h_{uv})$, $c(l_{uv})$, with the smallest cost sequence.

Suppose that we know the position where we will insert φ_v and want to compute the cost sequence of the layout sequence φ_{uv}. We consider only the case where $\eta_{1u} + \gamma_v \geqslant \gamma_u$ and $\eta'_{1u} + \gamma_v \leqslant \gamma_u$ (see Fig. 6) (the other cases are similar). Let x'_{ik} be the point in the layout of T_u where the cut η'_{1u} occurs. Then,

$$\varphi_{uv} = (x_1, \ldots, x_i, *, x'_{i1} + 1, \ldots, x'_{ik} + 1, y'_1 + 1, \ldots, y'_{j1} + 1, y_j, \ldots, y_1, x'_{ik}, \ldots, x'_1).$$

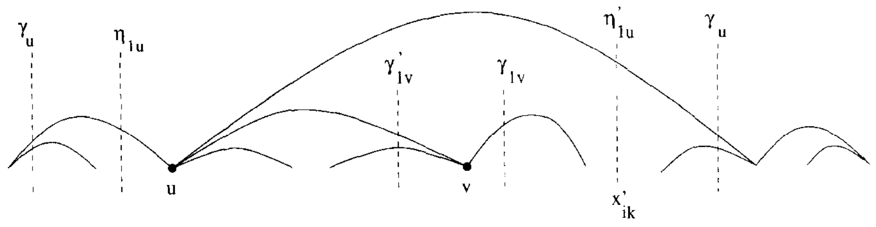

Fig. 6.

In order to compute the cost sequence we distinguish subcases:

I. $x_{ik} < min\{\eta_{1v}, \eta'_{1v}\}$ and $x_{ik} + 1 < min\{\eta_{1v}, \eta'_{1v}\}$. In this case we have

```
    if $\gamma_v < \gamma'_{2u} + 1$
    then $cost(\varphi_{uv}) = (\gamma_{1u}, \eta_{1u}, \ldots, *, \ldots, \eta'_{2u} + 1, \gamma'_{2u} + 1, \eta'_{1u}, \gamma'_{1u})$
    else $cost(\varphi_{uv}) = (\gamma_{1u}, \eta_{1u}, \ldots, *, \ldots, \eta'_{2u} + 1, \gamma'_{2u} + 1, \eta'_{1u} + 1, \gamma_v, \eta'_{1u}, \gamma'_{1u})$
```

II. $x_{ik} > max\{\eta_{1v}, \eta'_{1v}\}$. Let $\eta_{1v} < \eta'_{1v}$ and $\gamma_{\max} = max\{\gamma_{2v}, \gamma'_{1v} + 1, \gamma'_{2u} + 1\}$.
If $\gamma_{max} = \gamma'_{2u} + 1$ then

$$cost(\varphi_{uv}) = (\gamma_{1u}, \eta_{1u}, \ldots, *, \ldots, \eta'_{2u} + 1, \gamma'_{2u} + 1, \eta_{1v}, \gamma_{1u}).$$

If $\gamma_{max} = \gamma'_{1v} + 1$ then

$$cost(\varphi_{uv}) = (\gamma_{1u}, \eta_{1u}, \ldots, *, \ldots, \eta'_{2u} + 1, \gamma'_{2u} + 1, \eta_{1u} + 1, \gamma'_{1v} + 1, \eta_{1v}, \gamma_{1u}).$$

Let $\gamma_{max} = \gamma_{2v}$. We consider the case that $\gamma'_{1v} + 1 > \gamma'_{2u} + 1$ and $\eta'_{1v} + 1 < \eta_{1u} + 1$ (the other cases are similar). We use binary search to find in the sequence $(\gamma_{mv}, \eta_{(m-1)v}, \ldots, \eta_{2v}, \gamma_{2v})$ the γ_{lv} with the *smallest* possible l such that: $\gamma_{lv} < \gamma'_{1v} + 1$ or $\eta_{lv} > \eta'_{1v} + 1$. Suppose that for $l = l_1$ we have $\gamma_{l_1v} > \gamma'_{1v} + 1$ and $\eta_{l_1v} > \eta'_{1v} + 1$ (again the other cases are similar). Then, $cost(\varphi_{uv}) = (\gamma_{1u}, \eta_{1u}, \ldots, *, \ldots, \eta'_{2u} + 1, \gamma'_{2u} + 1, \eta_{1u} + 1, \gamma'_{1v} + 1, \eta'_{1v} + 1, \gamma_{l_1v}, \eta_{(l_1-1)v}, \ldots, \gamma_{2v}, \eta_{1v}, \gamma_{1u})$.

III. The other subcases are similar to the above ones.

Case 3: $\gamma_u > \gamma_v$ and φ_u, φ_v balanced.

Subcase 3.1. Suppose that $\gamma_v + \eta_{1u} = \gamma_u$ and $\eta'_{1u} = \eta_{1u}$. If we insert T_v between the vertices where the cut η_{1u} (or η'_{1u}) is realized, then the cost of φ_{uv} cannot be less than $\gamma_u + 1$. For this reason, we try to spread the vertices of T_v between the vertices of T_u in such a way that $\gamma_{uv} = \gamma_u$. To see this, suppose that $\gamma_{2u} + \eta_{1v} \leqslant \gamma_u$ and $\gamma'_{2u} + \eta'_{1v} \leqslant \gamma_u$. Also suppose that $min\{\gamma_{2v} + \eta_{2u}, \gamma'_{2v} + \eta_{2u}\} \leqslant \gamma_u - 1$ and $max\{\gamma_{2v} + \eta_{2u}, \gamma'_{2v} + \eta_{2u}\} \leqslant \gamma_u$. Let $c(v_{\gamma 1})$ $(c(v_{\gamma' 1}))$ be the part of the left (resp., right) sequence of φ_v – with respect to the position of v – from the beginning until the point realizing the cut γ_{1v} (resp., γ'_{1v}). We insert $c(v_{\gamma 1})$ $(c(v_{\gamma' 1}))$ in the position of φ_u realizing the cut of η_{1u} (resp., η'_{1u}) and the rest of φ_v in the position of φ_u realizing the cut of η_{2u} (see Fig. 7). (Notice that the cost of the resulting layout is γ_u but it is not necessarily optimal.)

A parallel procedure implementing the merge-operation A in this case follows.
Let k be the largest index such that (i) $\gamma_{kv} = \gamma'_{kv}$ and (ii) $\forall i \leqslant k, \gamma_{iv} = \gamma'_{iv}$ (notice that we can find this index easily in parallel in $O(\log k)$ time using m processors, where m is the length of the cost sequence $cost(\varphi_u)$).

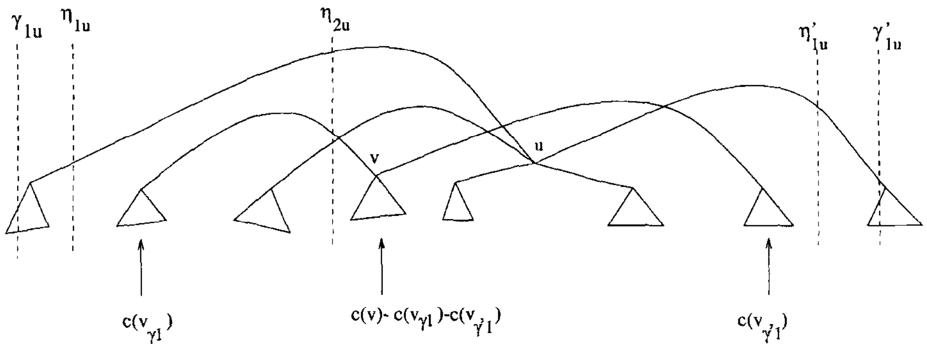

Fig. 7.

For each γ_{iv}, $1 \leqslant i \leqslant k$, we examine if the following holds:

$$(A) \qquad \gamma_{iv} + \eta_{iu} \leqslant \gamma_u - 1 \quad \text{or} \quad \gamma_{iv} + \eta'_{iu} \leqslant \gamma_u - 1.$$

For each γ_{iv}, $1 \leqslant i \leqslant k$, that does not satisfy condition (A) we examine if

$$(B) \qquad \gamma_{iv} + \eta_{iu} = \gamma_u \quad \text{and} \quad \eta_{iu} = \eta'_{iu} \quad \text{and} \quad max\{\gamma_{(i+1)u} + \eta_{iv}, \gamma'_{(i+1)u} + \eta'_{iv}\} \leqslant \gamma_u.$$

Also, for $\gamma_{(k+1)v}$ we examine if

$$min\{\gamma_{(k+1)v} + \eta_{(k+1)u}, \gamma'_{(k+1)v} + \eta_{(k+1)u}\} \leqslant \gamma_u - 1$$

and

$$max\{\gamma_{(k+1)v} + \eta_{(k+1)u}, \gamma'_{(k+1)v} + \eta_{(k+1)u}\} \leqslant \gamma_u$$

or

$$min\{\gamma_{(k+1)v} + \eta'_{(k+1)u}, \gamma'_{(k+1)v} + \eta'_{(k+1)u}\} \leqslant \gamma_u - 1$$

and

$$max\{\gamma_{(k+1)v} + \eta'_{(k+1)u}, \gamma'_{(k+1)v} + \eta'_{(k+1)u}\} \leqslant \gamma_u.$$

We call the above condition (C).

In the sequel, for each γ_{iv}, $1 \leqslant i \leqslant k$, we create a vertex with label 1 if γ_{iv} satisfies condition (A), label 0 if γ_{iv} does not satisfy (A) but satisfies (B) and *null* if γ_{iv} does not satisfy neither (A) nor (B). Also, we create a vertex for $\gamma_{(k+1)v}$ with label 1 if it satisfies (C) and label 0 if it does not. From each γ_{iv} with label 0 or 1 we draw an arc to $\gamma_{(i+1)v}$ if $\gamma_{(i+1)v}$ has label 0 or 1. In this way, we create one or more lists. Using the pointer doubling technique we find, in the list with head γ_{1v}, the first γ_{iv} with label 1. If such a γ_{iv} does not exist, there is no optimal layout of T_{uv} with cost γ_u. Otherwise, there is one (or more) layout of T_{uv} with cost γ_u. In order to find an optimal one do the following. Find the largest η_{ju}, $j \geqslant i$ satisfying

$$min\{\gamma_{iv} + \eta_{ju}, \gamma'_{iv} + \eta_{ju}\} \leqslant \gamma_{ju} + \eta_{(i-1)v} - 2$$

and

$$max\{\gamma_{iv} + \eta_{ju}, \gamma'_{iv} + \eta_{ju}\} \leqslant \gamma_{ju} + \eta_{(i-1)v} - 1$$

and the largest η'_{ju}, $j \geqslant i$ satisfying

$$min\{\gamma_{iv} + \eta'_{ju}, \gamma'_{iv} + \eta'_{ju}\} \leqslant \gamma'_{ju} + \eta'_{(i-1)v} - 2$$

and

$$max\{\gamma_{iv} + \eta'_{ju}, \gamma'_{iv} + \eta'_{ju}\} \leqslant \gamma'_{ju} + \eta'_{(i-1)v} - 1.$$

If such a j does not exist, then let $j = i$.

Let $c_1(uv)$ ($c_2(uv)$) be the layout sequence which we take if we insert the part of φ_v realizing the cut γ_{kv} into η_{kv}, the cut γ'_{kv} into η'_{ku}, $1 \leqslant k \leqslant (i-1)$, and the rest of φ_v into η_{ju} (resp., η'_{ju}). Then, φ_{uv} is this one from $c_1(uv)$, $c_2(uv)$, that has the smallest cost sequence.

Once we have the layout sequence φ_{uv} we can construct its cost sequence. The construction is similar to that of case 2.

The other subcases of case 3 as well as the remaining cases are similar to the above ones.

Lemma 7. *Let T_{uv} be a tree rooted at a vertex u that consists of the trees T_u, T_v rooted at u, v, respectively, and the edge $\{u, v\}$. Let also φ_u, φ_v be optimal layouts of T_u and T_v, respectively. Then, the merge-operation A correctly computes an optimal layout φ_{uv} of T_{uv} in* O($\log n$) *time using* O(n) CREW PRAM *processors, where n is the maximum of the lengths of φ_u, φ_v.*

Proof. The correctness is provided by considering the cost sequences of T_u and T_v and proving that in each one of the cases, there is no layout of T_{uv} with cost sequence smaller than the cost sequence of the layout produced by the procedure implementing the merge-operation A.

It is easy to see that in case 1.1 any other layout with cost $\gamma_u + 1$ (which is the best possible) has cost sequence larger than or equal to the one given above.

To see the correctness of case 2, we suppose w.l.o.g. that the cost sequence of the optimal layout is as follows: $(\gamma_{1u}, \eta_{1u}, \ldots, \gamma_{iu}, \eta_{iu}, \gamma_{(i+1)u} + 1, \eta_{(i+1)u} + 1, \ldots, \gamma_{ku} + 1, *, \gamma'_{lu}, \ldots, \eta'_{1u}, \gamma'_{1u})$ (i.e., i is the largest index such that $\eta_{iu} + \gamma_v \leqslant \gamma_{iu} - 1$ and the cost sequence of $c(h_{uv})$ is smaller than the corresponding cost sequence of $c(l_{uv})$). We also suppose that $compare_u = \langle \gamma^c_{1u}, \eta^c_{1u}, \gamma^c_{2u}, \eta^c_{2u}, \ldots, \gamma_{iu}, \eta_{iu}, \gamma_{(i+1)u}, \eta_{(i+1)u} \rangle$. Note that in this case the "compare" sequence of the new cost sequence will be as follows: $\langle \gamma^c_{1u}, \eta^c_{1u}, \gamma^c_{2u}, \eta^c_{2u}, \ldots, \gamma_{iu}, \eta_{iu}, \gamma_{yu} + 1, \eta_{yu} + 1 \rangle$ for $i < y \leqslant k$. Note also that if we insert φ_v in the position where the cut of η_{xu} is realized, for x either smaller or larger than i, then the "compare" sequence of the corresponding cost sequence will be lexicographically larger than the above one.

For the other cases the correctness comes also easily from their description. □

Merge-operation B

The parallel implementation of merge-operation B is based on merge-operation A. Let T_v be a tree rooted at v with children $v_1,\dots,v_d$ and T_u a tree rooted at u with children $u_1,\dots,u_{d'}$. Let also d' be greater than or equal to d. Suppose that φ_u, φ_v are optimal layouts of T_u, T_v, respectively. Then, the merge-operation B computes an optimal layout φ_{uv} of the tree T_{uv} rooted at u and having as children the children of both T_u and T_v, as follows:

Consider the layout φ_{vi}, for each $i \in \{1,\dots,d\}$, which is the restriction of φ_v to the subtree T_i rooted at v_i, for each $i \in \{1,\dots,d\}$. Note that each φ_{vi}, is an optimal layout of T_i, for $i = 1,\dots,d$ (if not then there is another layout φ'_{vi} with smallest cost sequence resulting in another layout φ'_v for T_v which has smaller cost sequence than φ_v; but φ_v is optimal). Then, sequentially use merge-operation A to merge each one of φ_{vi}, for $i = 1,\dots,d$, with φ_u. Notice that from the optimality of the merge-operation A, we have that the resulting layout φ_u is optimal.

The proof of the following lemma is based on Lemma 7 and the description of the merge-operation B.

Lemma 8. *Let T_u be a tree rooted at u, T_v a tree rooted at v and φ_u, φ_v be optimal layouts of T_u, T_v, respectively. Then, the merge-operation B correctly computes an optimal layout φ_{uv} of T_{uv} which is rooted at u and has as children the children of both T_u and T_v, in* $O(d \log n)$ *time using* $O(n)$ CREW PRAM *processors, where d is the minimum of the degrees of u, v and n is the maximum of the sizes of T_u, T_v.*

We give now the parallel algorithm for the mincut linear arrangement problem on trees.

```
for all $v \in T$ in parallel do
    Let $T_v$ be the tree $T$ rooted at $v$. Convert $T_v$ into a binary tree $T_{v0}$.
    Apply the parallel tree contraction technique to $T_{v0}$ to compute an
    optimal layout sequence $\varphi_v$ of $T_v$ and its cost sequence $cost(\varphi_v)$.
$cost(\varphi_T) = min\{cost(\varphi_v) \mid v \in T\}$
$MINCUT(T) = \varphi_T$ {* $\varphi_T$ is a layout sequence with cost sequence $cost(\varphi_T)$ *}
```

Theorem 3. *Given an undirected tree T with n vertices, the above algorithm constructs a minimum cut layout of T in time* $O(d \log^2 n)$ *using* $O(n^2/\log n)$ CREW PRAM *processors, where d is the maximum degree of T.*

Proof. The correctness of the approach comes from the fact that a minimum cut layout of T is computed as the layout out of minimum cut layouts of T_v, for each $v \in T$, which attains minimum cutwidth, and the fact that the shunt operation correctly computes an optimal layout of a subtree rooted at a vertex w, using optimal layouts of the subtrees rooted at w's children (recall the discussion at the beginning of the subsection). The correctness of the shunt operation follows from Lemmas 7, 8, and the fact that the merge-operations A and B are combined correctly (see the description of the shunt operation). The time/processor bounds of the algorithm come from the bounds of the parallel tree contraction technique and Lemmas 7 and 8. □

Acknowledgements

We are grateful to Mihalis Yannakakis and to the anonymous referees for their valuable comments on a previous version of this work.

References

[1] K. Abrahamson, N. Dadoun, D. Kirkpatrick and K. Przytycka, A simple parallel tree contraction algorithm, *J. Algorithms* **10** (1989) 287–302.

[2] D. Adolphson and T.C. Hu, Optimal linear ordering, *SIAM J. Appl. Math.* **25** (1973) 403–423.

[3] M. Chung, F. Makedon, I.H. Sudborough and J. Turner, Polynomial time algorithms for the min cut problem on degree restricted trees, in: *Proc. IEEE Symp. on Foundations of Computer Science (FOCS)* (1982) 262–271.

[4] J. Díaz, Graph layout problems, in: *Proc. of Symp. on Mathematical Foundations of Computer Science (MFCS)*, Lecture Notes in Computer Science, Vol. 629 (Springer, Berlin, 1992) 14–23.

[5] J. Díaz, A. Gibbons, G. Pantziou, M. Serna, P. Spirakis and J. Toran, Parallel algorithms for some tree layout problems, in: *Proc. Computing and Combinatorics Conf. (COCOON)*, Lecture Notes in Computer Science (Springer, Berlin, 1995).

[6] S. Even and Y. Shiloach, NP-completeness of several arrangements problems, Tech. Report, TR-43, The Technion, Haifa, 1978.

[7] F. Gavril, Some NP-complete problems on graphs, in: *Proc. 11th. Conf. on Information Sciences and Systems* (1977) 91–95.

[8] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of* NP-*Completeness* (Freeman, San Fransisco, 1979).

[9] L.H. Harper, Optimal numberings and isoperimetric problems on graphs, *J. Combin. Theory* **1** (1966) 385–393.

[10] R. Karp and V. Ramachandran, Parallel algorithms for shared memory machines, in: J. van Leewen, ed., *Handbook of Theoretical Computer Science, Vol. A* (Elsevier, Amsterdam, 1990) 869–942.

[11] B. Monien and I.H. Sudborough, Min cut is NP-complete for edge weighted trees, in: *Proc. 13th Coll. on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science (Springer, Berlin, 1986) 265–274.

[12] M.T. Shing and T.C. Hu, Computational complexity of layout problems, in: T. Ohtsuki, ed., *Layout Design and Verification* (North Holland, Amsterdam, 1986) 267–294.

[13] Y. Shiloach, A minimum linear arrangement algorithm for undirected trees, *SIAM J. Comput.* **8** (1979) 15–31.

[14] M. Yannakakis, A polynomial algorithm for the min cut linear arrangement of trees, *J. ACM* **32** (1985) 950–988.