

**Efficient Parallel Algorithms  
for Some Tree Layout Problems**

J. Díaz	A. Gibbons
G. Pantziou	M.J. Serna
P. Spirakis	J. Torán

Report LSI-92-33-R

 **UPC**  
Facultat d'Informàtica  
de Barcelona - Biblioteca

**12 MARÇ 1993**

# Efficient Parallel Algorithms for some Tree Layout Problems \*

J.Díaz<sup>†</sup>   A. Gibbons<sup>‡</sup>   G. Pantziou<sup>§</sup>   M. Serna<sup>†</sup>   P.Spirakis<sup>§</sup>  
J.Toran<sup>†</sup>

## Abstract

The minimum cut and minimum sum 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 P for trees. We show in this paper that both problems are NC for trees.

## 1 Introduction

Given a graph  $G = (V, E)$  with  $|V| = n$ , a *layout* of  $G$  is a one-to-one mapping  $\varphi$  from  $V$  to the first  $n$  integers  $\{1, 2, \dots, n\}$ . The term layout is also known as *linear arrangement* [Yan83], [Shi79]. Notice that a layout  $\varphi$  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  $\varphi(u) < i \leq \varphi(v)$ . The *cutwidth* of  $\varphi$ , denoted  $\gamma(\varphi)$ , is the maximum cut of  $\varphi$  over all integers from 1 to  $n$ .

Graph layout problems are motivated as simplified mathematical models of VLSI layout. Given a set of modules, the VLSI layout problem consists in placing the modules on a board in a non-overlapping manner, and then wiring together the terminals on the different modules according to a given wiring specification and in such a way that the wires do not interfere among them.

We can model a VLSI circuit by means of a graph, where the edges of the graph represent the wires, and the nodes represent the modules. Of course, this graph is an over-simplified 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. Deterministic exact algorithms are not known for placement and routing problems in the real world, and the techniques used are based on a more or less efficient heuristic algorithms (see the survey by Shing and Hu [Shi79]).

In this paper we shall consider two layout problems. The first problem is called *the minimum sum linear arrangement*, MINLA. Given a graph  $G = (V, E)$ , find the layout  $\varphi$  which minimizes

---

\*This research was supported by the ESPRIT BRA Program of the EC under contract no. 3075, project ALCOM.

<sup>†</sup>Departament de Llenguatges i Sistemes, Universitat Politècnica Catalunya, Pau Gargallo 5, 08028-Barcelona

<sup>‡</sup>Department of computer Science, University of Warwick

<sup>§</sup>Computer Technology institute, University of Patras

$\sum_{\{u,v\} \in E} |\varphi(u) - \varphi(v)|$ . The MINLA is a simplified version of the problem of how to place  $n$  modules in such a way to minimize the total length of the wire interconnecting them.

The MINLA problem is NP-complete for general graphs [GJ76]. Moreover, 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, Harper solved it for the case where the graph is a de Bruijn graph of order four [Har70]. Adolph and Hu gave a  $O(n \log n)$  algorithm for the case the graph is a rooted tree [AH73]. Even and Shiloach proved the problem was also NP-complete for bipartite graphs [ES78]. Finally, Shiloach proved that the MINLA can be solved for the case of an unrooted tree with  $n$  nodes by a deterministic algorithm running in time  $O(n^{2.2})$  [Shi79].

The second problem that we shall consider is *the minimum cut problem*, MINCUT. Given a graph  $G = (V, E)$ , find the layout  $\varphi$  that minimizes the cutwidth  $\gamma(\varphi)$ . A particular and important case of this problem is *the graph bisection problem*; find the partition of  $2n$  vertices into two subsets of size  $n$  in such a way that it minimizes the cut between subsets. Assuming that a board has a number of vertical and horizontal tracks for wiring the modules, the general MINCUT gives us the minimal number of tracks necessary to connect the modules.

The MINCUT problem is NP-complete for general graphs [Gav77]. 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 [Har66]. Chung, Makedon, Sudborough and Turner presented a  $O(n(\log n)^{d-2})$  time algorithm to solve the MINCUT problem on trees, where  $d$  is the maximum degree of any node in the tree [CMST82]. The MINCUT can be solved for the case of an unrooted tree with  $n$  nodes in time  $O(n \log n)$  [Yan83]. The particular case of the *graph bisection problem* is also NP-complete [GJ79].

We present here two algorithms in  $NC$ . The first one solves the MINLA for unrooted trees in  $O(\log^2 n)$  time and  $O(n^2 3^{\log n})$  processors on a CREW PRAM. The second algorithm solves the MINCUT for trees of unrooted trees in time  $O(\log^2 n)$  and  $O(n^2)$  processors in the CREW PAM model.

## 2 A parallel algorithm for the MINLA problem on trees

### 2.1 Basic definitions and theorems

Let  $\varphi$  be a layout of a tree  $T$ . The cost of  $\varphi$  is defined by  $C[\varphi, T] = \sum_{\{v_i, v_j\} \in E} |\varphi(v_i) - \varphi(v_j)|$ .  $\varphi$  is a minimum sum layout of  $T$  if there is no other arrangement with smaller cost. Let  $\bar{\varphi}$  denote the layout obtained by reversing the order of the vertices. Note that  $C[\varphi, T] = C[\bar{\varphi}, T]$ . In general,  $C[T]$  will denote the minimum cost of a layout for  $T$ .

**Definition 1** 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 of  $T \bmod v$ , say  $T'$  such that  $v' \in T'$ . The vertex  $v'$  is the root of  $T' \bmod v$ .

Given  $v \in V$ , let  $T_1, \dots, T_k$  be subtrees of  $T \bmod v$ ,  $T - (T_1, \dots, T_k)$  denotes the tree obtained by removing the vertices of  $T_1, \dots, T_k$  and their incident edges. When  $T_0, T_1, \dots, T_k$

are all the subtrees of  $T \bmod v$  we will assume that they are numbered in such a way that  $n_0 \geq n_1 \geq \dots \geq n_k$  where  $n_i$  denotes the size of  $T_i$ ,  $i = 0, 1, \dots, k$ .

A *central vertex* is a vertex  $v_*$  such that if  $T_0, T_1, \dots, T_k$  are all the subtrees of  $T \bmod v_*$  then  $n_i \leq \lfloor n/2 \rfloor$  for  $i = 0, 1, \dots, k$ . A way to compute such a vertex is given in [Shi79].

**Definition 2** Let  $T$  be an  $n$ -vertex tree, let  $v \in T$ , and let  $\varphi$  be a layout of  $T$ .  $T$  is called a *right anchored tree at  $v$*  and denoted by  $\overrightarrow{T}(v)$  when its cost is defined by

$$C[\varphi, \overrightarrow{T}(v)] = C[\varphi, T] + n - \varphi(v)$$

It is called a *left anchored tree at  $v$*  and denoted by  $\overleftarrow{T}(v)$  when its cost is defined by

$$C[\varphi, \overleftarrow{T}(v)] = C[\varphi, T] + \varphi(v) - 1$$

**Remark.** Finding a minimum sum 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 cost is unchanged.

In the following we will use  $T(\alpha)$  to denote a tree, with  $\alpha = 0$  for free trees and  $\alpha = 1$  for anchored trees. When we refer to the root of a tree this root will be either the vertex at which the anchor is connected or the central vertex for free trees.

Let first give two technical lemmas related to sizes of subtrees.

**Lemma 1** Let  $T(\alpha)$  be a tree. Let  $p_\alpha$  be the value of the greatest integer  $p_\alpha$  satisfying

$$n_i > \left\lfloor \frac{n_0 + 2}{2} \right\rfloor + \left\lfloor \frac{n_* + 2}{2} \right\rfloor \text{ for } i = 1, 2, \dots, 2p_\alpha - \alpha$$

where  $n_* = n - \sum_{i=0}^{2p_\alpha - \alpha} n_i$  and  $T_* = T(\alpha) - (T_1, \dots, T_{2p_\alpha - \alpha})$ , then  $|T_*| < n/2$ , where  $n$  is the size of  $T(\alpha)$ .

**Lemma 2** Let  $T$  be a free tree with central vertex  $v_*$ , and let  $T_0, T_1$  be the two heaviest subtrees  $\bmod v_*$ . If  $|T - (T_0, T_1)| > n/2$  then  $v_*$  is a central vertex of  $T - (T_0, T_1)$ .

In order to compute the minimum sum layout we consider some special layouts. We will say that a tree  $T(\alpha)$  has an layout of type

$$(T_1(\alpha_1), \dots, T_k(\alpha_k), \dots, T_r(\alpha_r))$$

when the layout is computed from the corresponding subtree layouts adding the number of nodes of the subtrees on the left, and the union of all the subtrees is  $T(\alpha)$ .

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  $\beta$  be the first index for which  $|T - (T_0, \dots, T_\beta)| \leq n/2$  and let  $p_{\alpha_i}$  be the value obtained applying lemma 1 to  $T - (T_0, \dots, T_i)$  (anchored at  $v_*$  if  $i$  is even) and let  $T_*^i$  be  $T - (T_0, \dots, T_i, \dots, T_{2p_{\alpha_i} - \alpha_i})$ , for each  $i = 1, \dots, \beta$ . We consider the following layout

$$A_*^i = (\overrightarrow{T}_{i+1}(v_{i+1}), \dots, T_*^i, \dots, \overleftarrow{T}_{i+2}(v_{i+2}))$$

**Lemma 3** A minimum sum layout for a free tree  $T$  can be computed as the minimum of the following layouts:

$$A_i = (\vec{T}_0(v_0), \vec{T}_3(v_3), \vec{T}_4(v_4), \dots, A_\star^i, \dots, \vec{T}_6(v_6), \vec{T}_5(v_5), \vec{T}_2(v_2), \vec{T}_1(v_1))$$

for  $i = 0, \dots, \beta$ , and

$$A_{\beta+1} = (\vec{T}_0(v_0), \vec{T}_3(v_3), \vec{T}_4(v_4), \dots, T - (T_0, \dots, T_\beta), \dots, \vec{T}_6(v_6), \vec{T}_5(v_5), \vec{T}_2(v_2), \vec{T}_1(v_1))$$

if  $\beta$  is even or

$$A_{\beta+1} = (\vec{T}_0(v_0), \vec{T}_3(v_3), \vec{T}_4(v_4), \dots, \overleftarrow{T - (T_0, \dots, T_\beta)}, \dots, \vec{T}_6(v_6), \vec{T}_5(v_5), \vec{T}_2(v_2), \vec{T}_1(v_1))$$

if  $\beta$  is odd.

In order to get an expression to compute the cost of the previous layouts we consider the following. Let  $Q_i$  be

$$(n_{i+3} + n_{i+4}) + 2(n_{i+5} + n_{i+6}) + \dots + (p_0^i - 1)(n_{2p_0^i-1} + n_{2p_0^i}) + p_0^i n_\star^i$$

when  $i$  is even and

$$(n_{i+2} + n_{i+3}) + 2(n_{i+4} + n_{i+5}) + \dots + (p_1^i - 1)(n_{2p_1^i-2} + n_{2p_1^i-1}) + p_1^i (n_\star^i + 1)$$

when  $i$  is odd.

Let  $R_i$  be  $\frac{i}{2}n - \frac{i}{2}(n_0 + n_1) - (\frac{i}{2} - 1)(n_2 + n_3) - \dots - (n_{i-2} + n_{i-1})$  when  $i$  is even and  $R_{i-1}$  when  $i$  is odd.

Then the cost of these layouts can be computed as

$$C(A_i) = \sum_{j=0}^{i-1} C[\vec{T}_j(v_j)] + \sum_{j=i+1}^{2p_\alpha^i-2\alpha} C[\vec{T}_j(v_j)] + C[T_\star^i] + Q_i + R_i$$

$$C(A_{\beta+1}) = \sum_{j=0}^{\beta} C[\vec{T}_j(v_j)] + C[\overrightarrow{T - (T_0, \dots, T_\beta)}(v_\star)] + R_\beta + 1 \text{ if } i \text{ is even}$$

$$C(A_{\beta+1}) = \sum_{j=0}^{\beta} C[\vec{T}_j(v_j)] + C[T - (T_0, \dots, T_\beta)(v_\star)] + R_\beta + n - n_0 - \dots - n_\beta \text{ if } i \text{ is odd}$$

For an anchored tree we get the following decomposition

**Lemma 4** Let  $\vec{T}(v_\star)$  be an anchored tree and let  $\gamma$  be the first index for which

$$|T - (T_0^0, \dots, T_0^\gamma)| \leq n/2$$

a minimum layout for  $T$  can be computed as the minimum of the following layouts:

$$B_i = (\vec{T}_0(v_0), \dots, \vec{T}_0^i(v_0^i), T - (T_0^0, \dots, T_0^i)) \quad \text{for } i = 0, \dots, \gamma$$

The cost of layout  $B_i$  can be computed as:

$$C(B_i) = \sum_{j=0}^i C[\vec{T}_0^j(v_0^j)] + C[T - (T_0^0, \dots, T_0^i)] + n - n_0^i, \text{ for } i = 0, \dots, \gamma$$

Note that from lemmas 1 and 2 all the trees for which we have to compute an layout in the decomposition given for free or anchored trees have size less than  $|T|/2$ . Also in both cases we obtain less than  $2n$  trees of size less than  $n/2$ , furthermore the sum of the tree sizes is at most  $3n$ .

## 2.2 The parallel algorithm

Our algorithm will be divided into two stages. In the first one we decompose the tree according to lemmas 3 and 4 until all the trees have size 1. At the same time we record the expressions that will allow us to compute the layout from the smallest trees. In the second stage we reconstruct the layout, until we get a minimum layout for the whole tree. The complexity bounds for the algorithm are the following

**Theorem 1** *There is an NC algorithm to compute a minimum sum layout for an undirected tree. The algorithm uses  $O(n^{3.6})$  processors and  $O(\log^2 n)$  time, where  $n$  is the number of vertices in the tree.*

We comment the main points on the parallel algorithm to compute the cost of a minimum sum layout for an undirected tree, it is easy to add the modifications to get also the layout.

### Tree representation

We represent each tree using a mask where the nodes that form part of the corresponding tree are recorded. Every time that we obtain a new tree we add a new node to the original graph, connected only to the corresponding nodes, the mask will be the same putting 1 in the position corresponding to the new node and 0 in the position corresponding to the old one. To compute the corresponding tree we run a connected components algorithm, those nodes that are in the connected component of the added node will be the ones that will remain in the mask.

To distinguish between free and anchored trees we keep the parameter  $\alpha$  and the corresponding root. This root will be either the vertex at which the anchor is connected or the central vertex.

### Free tree decomposition

In order to compute a central vertex we compute for each edge in the tree the sizes of the two corresponding subtrees. Then we compute the minimum of the difference of sizes over all edges. We take as central vertex the root of the heaviest subtree corresponding to an edge of minimum difference.

Once we have the central vertex, we have to compute subtree sizes (now the tree is rooted), and order subtrees by size. From the tree sizes using suffix sums we compute  $\beta, p_0^0, p_1^1, \dots, p_1^\beta$ . Consider the decomposition given in lemma 3 the cost of each layout can be computed as the sum of the costs of the corresponding subtrees, adding an expression that depends only on the sizes. The last amount can be computed now and recorded in a matrix together with pointers to the corresponding subtree masks.

### Anchored tree decomposition

Now we have a rooted tree, we first compute subtree sizes using the standard Euler's Tour technique [KR90] 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  $\gamma$ . Consider the decomposition given in lemma 4. The cost of each layout can be computed the sum of the costs of the corresponding subtrees, adding an expression that depends only on the sizes.

### Complexity

Note that the number of decomposition phases in the first stage is  $O(\log n)$ . Thus the maximum number of trees in a decomposition phase is  $O(3^{\log n}) = O(n^{2.6})$  taking into account that the sum of the sizes of the trees obtained in the decomposition of a tree  $T$  is at most  $3 \cdot |T|$ . Furthermore the time used in each phase in the first and second stage is  $O(\log n)$ . Thus the total requirements of the algorithm are  $O(n^{3.6})$  processors and  $O(\log^2 n)$  time.

## 3 A parallel algorithm for the MINCUT problem on trees

### 3.1 Preliminaries

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

With each node  $u \in T$ , we associate two pieces of information: i) a layout-sequence,  $c(u)$ , realizing the layout of the subtree rooted at  $u$  and  $u$ 's position in this layout and ii) a cost-sequence,  $cost(u)$ , of the layout  $c(u)$  defined as follows:  $cost(u) = (leftcost(u), *, rightcost(u))$ . (The "\*" denotes the position of  $u$ .)

$\langle leftcost(u) \rangle$  is a sequence  $\langle \gamma_1, \eta_1, \gamma_2, \eta_2, \dots \rangle$  where  $\gamma_1$  is the largest cut (in  $c(u)$ ) occurring on the left side of  $u$ . Let  $w_1$  be the point where the cut of  $\gamma_1$  occurs. If  $w_1$  is immediately to the left of  $u$  then  $\langle leftcost(u) \rangle = \langle \gamma_1 \rangle$ . Otherwise, let  $\eta_1$  be the smallest cut between  $w_1$  and  $u$  and let  $w_2$  be the point closest to  $u$  where  $\eta_1$  occurs. Suppose that  $\gamma_2$  is the maximum cut between  $w_2$  and  $u$  and  $w_3$  is the point closest to  $u$  where  $\gamma_2$  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  $\langle \gamma'_1, \eta'_1, \gamma'_2, \dots \rangle$  where  $\gamma'_1$  is the largest cut in  $c(u)$  occurring 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,  $\gamma_1 \geq \gamma_2 \geq \dots$ ,  $\eta_1 \leq \eta_2 \leq \dots$ ,  $\gamma'_1 \geq \gamma'_2 \geq \dots$ ,  $\eta'_1 \leq \eta'_2 \leq \dots$ . Also,  $\gamma_1 \geq \eta_1$ ,  $\gamma_2 \geq \eta_2$ , etc., and  $\gamma'_1 \geq \eta'_1$ ,  $\gamma'_2 \geq \eta'_2$ , etc. (Our cost-sequences definitions are motivated by the fundamental work of Yannakakis in [Yan83].)

If  $\gamma_1 = \gamma'_1$  then we say that the layout is *balanced*; otherwise, it is *unbalanced*. We define the *cost* of the layout-sequence  $c(u)$  as the quantity  $\gamma_u = \max\{\gamma_1, \gamma'_1\}$ .

Let  $a = (a_1, a_2, \dots)$  and  $b = (b_1, b_2, \dots)$  be two sequences. If  $a \neq b$ , then  $a > b$  iff  $a$  is lexicographically larger than  $b$ . If  $\gamma_1 \geq \gamma'_1$  and  $leftcost(u) > rightcost(u)$  then we call the left side of  $c(u)$  (with respect to the position of  $u$ ) *heavy side* and the right one *light*. Let  $cost_1 = (heavyside_1, *, lightside_1)$ ,  $cost_2 = (heavyside_2, *, lightside_2)$  be two cost-

sequences corresponding to two layout-sequences.  $cost_1 = cost_2$  iff  $heavyside_1 = heavyside_2$  and  $lightside_1 = lightside_2$ .  $cost_1 > cost_2$  iff  $(heavyside_1 > heavyside_2)$  or  $(heavyside_1 = heavyside_2$  and  $lightside_1 > lightside_2)$ .

Let  $T_u$  be a tree rooted at a node  $u$  and  $c(u)$  a layout-sequence realizing a layout  $\varphi$  of  $T_u$ . Let  $cost(u)$  be the cost-sequence of  $c(u)$ . We say that  $c(u)$  is *optimal* iff there is no other layout  $\varphi'$  of  $T_u$  with layout and cost-sequence  $c'(u)$  and  $cost'(u)$  respectively, such that  $cost'(u) < cost(u)$ .

### 3.2 The Algorithm

We give an  $O(n^2)$ -processor,  $O(\log^2 n)$ -time parallel algorithm which finds a minimum cutwidth linear arrangement of a tree  $T_o$ . The algorithm is based on the use of the parallel tree contraction technique. The *shunt* operation uses two merge-operations on the layout sequences:

*Merge-operation A:* Let  $T_{uv}$  be a tree rooted at a node  $u$ .  $T_{uv}$  consists of two trees  $T_u, T_v$  (rooted at  $u, v$  respectively) and the edge  $\{u, v\}$ . Suppose that  $c(u), c(v)$  are optimal layout sequences of  $T_u, T_v$ , respectively. The merge-operation *A* computes an optimal layout sequence  $c(uv)$  realizing the layout of  $T_{uv}$  and the cost sequence of  $c(uv)$ .

*Merge-operation B:* Let  $T_u$  be a tree rooted at  $u$  with children  $u_1, \dots, u_d$  and  $T_v$  a tree rooted at  $v$  with children  $v_1, \dots, v_{d'}$ . Suppose that we are given the optimal layout and cost-sequences of  $T_u, T_v$ . The merge-operation *B* computes an optimal layout-sequence  $c(uv)$  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 define now the *shunt* operation of the tree contraction technique:

Suppose that  $l_i$  is the leaf which is ready to perform the *shunt* operation 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 we are given the layout-sequences  $c(l_i), c(f_i), c(f_j), c(p(f_i))$  and the cost-sequences  $cost(l_i), cost(f_i), cost(f_j), cost(p(f_i))$  of  $l_i, f_i, f_j, p(f_i)$  respectively. Initially, all the sequences are equal to  $(0, *, 0)$ . In the sequel,  $f_{ij}$  will be the node which is the result of the *shunt* operation. We distinguish among two cases:

**Case 1.**  $l_i$  is the left leaf of  $f_i$ .

**Fact 1.** The nodes  $l_i, f_j$  are not of the *same label*.

We distinguish the following subcases:

**Case 1.a.**  $f_i, f_j$  are of the *same label*. In this case, first we merge the sequences  $c(l_i)$  and  $c(f_i)$  using the merge-operation *A* into the sequence  $c(f_i l_i)$ . Afterwards, we join the sequence  $c(f_i l_i)$  and  $c(f_j)$  into the sequence  $c(f_{ij})$  using the merge-operation *B*. Note that the new vertex  $f_{ij}$  is supposed to be of the *same label* with vertices  $f_i, f_j$ .

**Case 1.b.**  $f_i, f_j$  are not of the *same label*. We apply the merge-operation *A* first to merge  $c(l_i)$  and  $c(f_i)$  and then to merge  $c(f_i l_i)$  and  $c(f_j)$ . (Notice that  $f_{ij}$  and  $f_j$  are of the *same label*).

**Case 2.**  $l_i$  is the right leaf of  $f_i$ . (From fact 1,  $f_i, f_j$  cannot be of the *same label*)

**Case 2.a.**  $l_i, f_i$  are of the *same label*.

*Subcase 2.a.a.*  $f_i, p(f_i)$  are also of the *same label*. We apply the merge-operation *B* to merge  $c(l_i)$  and  $c(f_i)$  and after that to merge  $c(f_i l_i)$  and  $c(p(f_i))$ . The resulting sequence constitutes the new sequence of  $p(f_i)$  while  $c(f_{ij}) = c(f_j)$  ( $f_{ij}$  coincides with  $f_j$ ).

*Subcase 2.a.b.*  $f_i, p(f_i)$  are not of the *same label*. We use the merge-operation *B* to merge  $c(l_i)$  and  $c(f_i)$ , and the merge operation *A* to merge  $c(f_i l_i)$  and  $c(f_j)$ . Suppose that the resulting sequence  $c(f_{ij})$  is as in the figure 2, 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 figure 2). Let  $T_{f_{ij}}$  be

the subtree - of the current  $T_{v_0}$  - 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  $T_{v_0} - T_{f_{ij}}$  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  $A$  and  $B$  is a nontrivial task and is done by distinguishing a number of cases. Suppose that:

$c(u) = (x_1, x_2, \dots, x_i, *, x'_{i1}, \dots, x'_2, x'_1)$ ,  $c(v) = (y_1, y_2, \dots, y_j, *, y'_{j1}, \dots, y'_2, y'_1)$   
 $cost(u) = (\gamma_{1u}, \eta_{1u}, \dots, \gamma_{ku}, *, \gamma'_{lu}, \dots, \eta'_{1u}, \gamma'_{1u})$ ,  $cost(v) = (\gamma_{1v}, \eta_{1v}, \dots, \gamma_{mv}, *, \gamma'_{nv}, \dots, \eta'_{1v}, \gamma'_{1v})$   
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  $c(u), c(v)$  balanced. In this case it is clear that the cost of an optimal layout-sequence  $c(uv)$  cannot be less than  $\gamma_u + 1$ .

**Case 1.1.** Suppose that  $leftcost(u) \leq rightcost(u)$  and  $rightcost(v) \leq leftcost(v)$  and  $n_{1u} \neq 1, n'_{1v} \neq 1$ . Then we construct  $c(uv)$  from  $c(u), c(v)$  as follows (see figure 3):

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

and the cost-sequence of  $c(uv)$  is:  $cost(uv) = (\gamma_{1u} + 1, \eta_{1u} + 1, \dots, \gamma_{ku} + 1, *, \gamma'_{lu}, \dots, \eta'_{1u}, \gamma'_{1u})$ .

The other subcases are similar.

**Case 2:**  $\gamma_u > \gamma_v$  and  $c(u), c(v)$  balanced.

**Case 2.1.** Suppose that  $\gamma_v + \gamma_{1u} = \gamma_u$  and  $\eta'_{1u} = \eta_{1u}$ . If we insert  $T_v$  between the vertices where the cut  $\eta_{1u}$  (or  $\eta'_{1u}$ ) is realized then the cost of  $c(uv)$  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  $\gamma_{2u} + \eta_{1v} \leq \gamma_u$  and  $\gamma'_{2u} + \eta'_{1v} \leq \gamma_u$ . Also suppose that  $\min\{\gamma_{2v} + \eta_{2u}, \gamma'_{2v} + \eta_{2u}\} \leq \gamma_u - 1$  and  $\max\{\gamma_{2v} + \eta_{2u}, \gamma'_{2v} + \eta_{2u}\} \leq \gamma_u$ . Let  $c(v_{\gamma_1})$  ( $c(v_{\gamma'_1})$ ) be the part of the left (resp., right) sequence of  $c(v)$  - with respect to the position of  $v$  - from the beginning until the point realizing the cut  $\gamma_{1v}$  (resp.,  $\gamma'_{1v}$ ). We insert  $c(v_{\gamma_1})$  ( $c(v_{\gamma'_1})$ ) in the position of  $c(u)$  realizing the cut of  $\eta_{1u}$  (resp.,  $\eta'_{1u}$ ) and the rest of  $c(v)$  in the position of  $c(u)$  realizing the cut of  $\eta_{2u}$  (see figure 4). (Notice that the resulting layout is of cost  $\gamma_u$  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)  $\forall i \leq k, \gamma_{iv} = \gamma'_{iv}$  (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  $cost(u)$ ).

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

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

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

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

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

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

or

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

We call the above condition (C).

In the sequel, for each  $\gamma_{iv}$ ,  $1 \leq i \leq k$ , we create a node with label 1 if  $\gamma_{iv}$  satisfies condition (A), label 0 if  $\gamma_{iv}$  does not satisfy (A) but satisfies (B) and *null* nor (B). Also, we create a node for  $\gamma_{(k+1)v}$  with label 1 if it satisfies (C) and label 0 if it does not. From each  $\gamma_{iv}$  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  $\gamma_{1v}$  the first  $\gamma_{iv}$  with label 1. If such a  $\gamma_{iv}$  does not exist, there is no optimal layout-sequence of  $T_{uv}$  with cost  $\gamma_u$ . Otherwise, there is one (or more) layout-sequence of  $T_{uv}$  with cost  $\gamma_u$ . In order to find an optimal one do the following: find the largest  $\eta_{ju}$ ,  $j \geq i$  satisfying

$$\min\{\gamma_{iv} + \eta_{ju}, \gamma'_{iv} + \eta_{ju}\} \leq \gamma_u - 1 \text{ and } \max\{\gamma_{iv} + \eta_{ju}, \gamma'_{iv} + \eta_{ju}\} \leq \gamma_u$$

and the largest  $\eta'_{ju}$ ,  $j \geq i$  satisfying

$$\min\{\gamma_{iv} + \eta'_{ju}, \gamma'_{iv} + \eta'_{ju}\} \leq \gamma_u - 1 \text{ and } \max\{\gamma_{iv} + \eta'_{ju}, \gamma'_{iv} + \eta'_{ju}\} \leq \gamma_u$$

Let  $c_1(uv)$  ( $c_2(uv)$ ) be the layout-sequence which we take if we insert the part of  $c(v)$  realizing the cut  $\gamma_{kv}$  into  $\eta_{kv}$ , the cut  $\gamma'_{kv}$  into  $\eta'_{kv}$ ,  $1 \leq k \leq (i-1)$ , and the rest of  $c(v)$  into  $\eta_{ju}$  (resp.,  $\eta'_{ju}$ ). Then,  $c(uv)$  is this one from  $c_1(uv)$ ,  $c_2(uv)$ , with the smallest cost-sequence.

Once we have the layout-sequence  $c(uv)$  we can construct its cost-sequence. The construction will not be described here.

The other subcases of case 2 as well as the remaining cases are similar to the above ones and will not be considered in this extended abstract.

**Lemma 5** *Let  $T_{uv}$  be a tree rooted at a node  $u$  that consists of the trees  $T_u$ ,  $T_v$  rooted at  $u$ ,  $v$  respectively, and the edge  $\{u, v\}$ . Let also  $c(u)$ ,  $c(v)$  be optimal layout-sequences of  $T_u$  and  $T_v$  respectively. Then, the merge-operation A correctly computes an optimal layout-sequence  $c(uv)$  of  $T_{uv}$  in  $O(\log n)$  time using  $O(n)$  CREW PRAM processors ( $n$  is the maximum of the lengths of  $c(u)$ ,  $c(v)$ ).*

**Merge-operation B:**

The parallel implementation of merge-operation B is done in a way similar to that of case 2.1 of the merge-operation A, and will not be described here.

**Lemma 6** *Let  $T_u$  be a tree rooted at  $u$  with children  $u_1, \dots, u_d$  and  $T_v$  a tree rooted at  $v$  with children  $v_1, \dots, v_d$ . Let also  $c(u)$ ,  $c(v)$  be optimal layout-sequences of  $T_u$ ,  $T_v$  respectively. Then, the merge-operation B correctly computes an optimal layout-sequence  $c(uv)$  realizing a layout of the tree  $T_{uv}$  which is rooted at  $u$  and has as children the children of both  $T_u$  and  $T_v$ , in  $O(\log n)$  time using  $O(n)$  CREW PRAM processors ( $n$  is the maximum of the lengths of  $c(u)$ ,  $c(v)$ ).*

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

### ALGORITHM *MINCUT*( $T$ )

for all  $v \in T$  pardo

    Convert the tree rooted at  $v$  into a binary one  $T_{v0}$

    Apply the parallel tree contraction technique to  $T_{v0}$  to compute an optimal layout-sequence  $c(v)$  of  $T_v$  and its cost-sequence  $cost(v)$ .

odpar

$cost(T) = \min\{cost(v) \mid v \in T\}$

$MINCUT(T) = c(T) \{ * c(T) \text{ is a layout-sequence with cost-sequence } cost(T) * \}$

**Theorem 2** *Given a tree  $T$  with  $n$  vertices, the algorithm *MINCUT* constructs an optimal layout of  $T$  in time  $O(\log^2 n)$  using  $O(n^2)$  CREW PRAM processors.*

## References

- [AH73] D. Adolphson and T.C. Hu. Optimal linear ordering. *SIAM J. on Applied Mathematics*, 25(3):403–423, Nov 1973.
- [CMST82] M. Chung, F. Makedon, I.H. Sudborough, and J. Turner. Polynomial time algorithms for the min cut problem on degree restricted trees. In *FOCS*, volume 23, pages 262–271, Chicago, Nov 1982.
- [ES78] S. Even and Y. Shiloach. NP-completeness of several arrangements problems. Technical report, TR-43 The Technion, Haifa, 1978.
- [Gav77] F. Gavril. Some NP-complete problems on graphs. In *Proc. 11th. Conf. on Information Sciences and Systems*, pages 91–95, John Hopkins Univ., Baltimore, 1977.
- [GJ76] M.R. Garey and D.S. Johnson. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.
- [GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
- [Har66] L.H. Harper. Optimal numberings and isoperimetric problems on graphs. *Journal of Combinatorial Theory*, 1(3):385–393, 1966.
- [Har70] L.H. Harper. Chassis layout and isoperimetric problems. Technical Report SPS 37–66, vol II, Jet Propulsion Laboratory, Sept. 1970.
- [KR90] R. Karp and V. Ramachandran. Parallel algorithms for shared memory machines. In Jan van Leewen, editor, *Handbook of Theoretical Computer Science, Vol. A*, pages 869–942. Elsevier Science Publishers, 1990.
- [Shi79] Yossi Shiloach. A minimum linear arrangement algorithm for undirected trees. *SIAM J. on Computing*, 8(1):15–31, February 1979.
- [Yan83] Mihalis Yannakakis. A polynomial algorithm for the min cut linear arrangement of trees. In *IEEE Symp. on Found. of Comp. Sci.*, volume 24, pages 274–281, Providence RI, Nov. 1983.

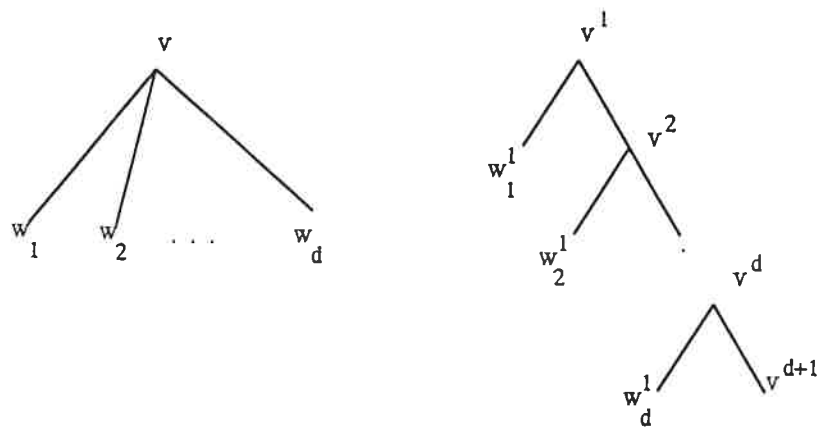


Figure 1

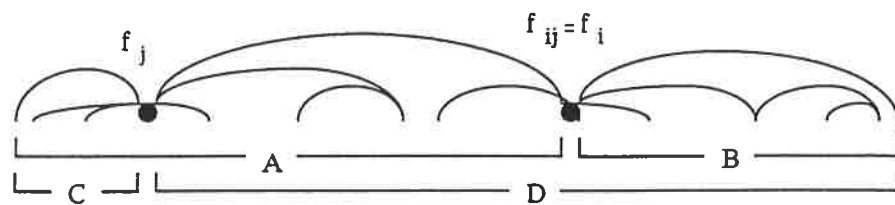


Figure 2

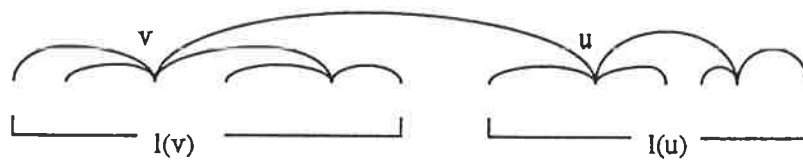


Figure 3

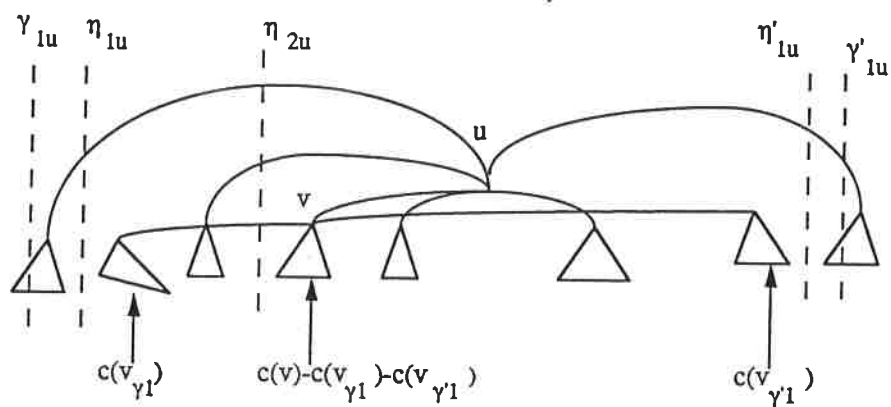


Figure 4

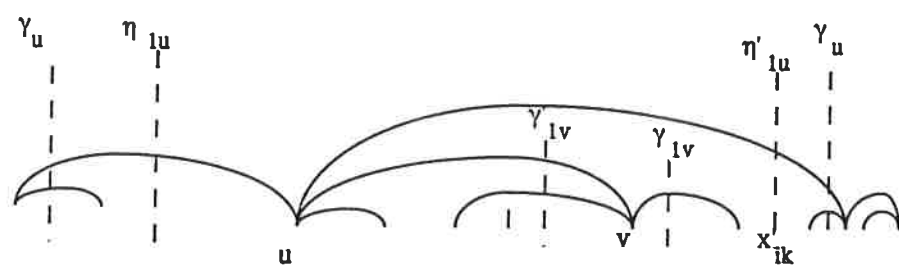


Figure 5