4-1993

Efficient Parallel Algorithms for some Tree Layout Problems

J Diaz
Universitat Politecnica Catalunya

A Gibbons
University of Warwick

Grammati E. Pantziou
Dartmouth College

M Serna
Universitat Politecnica Catalunya

Paul G. Spirakis
University of Patras

See next page for additional authors

EFFICIENT PARALLEL ALGORITHMS FOR SOME TREE LAYOUT PROBLEMS

J. Diaz, A. Gibbons, G. Pantziou,
M. Serna, P. Spirakis, J. Toran

Technical Report PCS-TR93-189

4/93

Efficient Parallel Algorithms for some Tree Layout Problems *

J.Díaz† A. Gibbons‡ G. Pantziou§ M. Serna† P.Spirakis¶
J.Toran†

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 in P for trees. We present here two algorithms in NC. The first solves the minimum sum linear arrangement problem for unrooted trees in $O(\log^2 n)$ time and $O(n^2 3^{\log n})$ CREW PRAM processors. The second algorithm solves the minimum cut arrangement for unrooted trees of maximum degree d in $O(d \log^2 n)$ time and $O(n^2/\log n)$ CREW PRAM processors.

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 $\{1, 2, \cdots, n\}$. The term layout is also known as *linear arrangement* [Yan83], [Shi79]. 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 $\varphi(u) < i \leq \varphi(v)$. The *cutwidth* of φ, denoted $\gamma(\varphi)$, is the maximum cut of φ over all integes 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-overlaping manner, and then wiring together the terminals on the differents 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

*This research was supported by the ESPRIT BRA Program of the EC under contract no. 7141, project ALCOM II. The work of the third author is also supported by the NSF grant No. CDA-9211155

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

‡Department of computer Science, University of Warwick

§Dartmouth College and Computer Technology Institute

¶University of Patras and Computer Technology Institute

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 φ which minimizes $\sum_{\{u,v\}\in E} |\varphi(u) - \varphi(v)|$. 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 Brujin 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 is 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 φ 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 is 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 unrooted trees of maximum degree d in time $O(d \log^2 n)$ and $O(n^2/\log n)$ processors in the CREW PRAM model.

2 A parallel algorithm for the MINLA problem on trees

2.1 Basic definitions and theorems

Let φ be a layout of a tree T. The cost of φ is defined by $C[\varphi, T] = \sum_{\{v_i,v_j\}\in E} \mid \varphi(v_i) - \varphi(v_j) \mid$. φ is a minimum sum layout of T if there is no other arrangement with smaller

cost. Let $\overline{\varphi}$ denote the layout obtained by reversing the order of the vertices. Note that $C[\varphi, T] = C[\overline{\varphi}, T]$. In general, $C[T]$ will denote the mimimum 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 mod v. *For each edge (v', v) there is a unique subtree of T mod v, say T', such that $v' \in T'$. The vertex v' is the root of T' mod v.*

Given $v \in V$, let $T_1, \ldots, T_k$ be subtrees of T mod v, $T-(T_1, \ldots, T_k)$ denotes the tree obtained by removing the vertices of $T_1, \ldots, T_k$ and their incident edges. When $T_0, T_1, \ldots, T_k$ are all the subtrees of T mod v we will assume that they are numbered in such a way that $n_0 \geq n_1 \geq \ldots \geq n_k$ where n_i denotes the size of T_i, $i = 0, 1, \ldots, k$.

A *central vertex* is a vertex v_* such that if $T_0, T_1, \ldots, T_k$ are all the subtrees of T mod v_* then $n_i \leq \lfloor n/2 \rfloor$ for $i = 0, 1, \ldots, k$. 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 φ 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, whith $\alpha = 0$ 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_α be the value of the greatest integer p_α satisfying*

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

where $n_* = n - \sum_{i=0}^{2p_\alpha - \alpha} n_i$ and $T_* = T(\alpha) - (T_1, \ldots, T_{2p_\alpha - \alpha})$, then $\mid T_* \mid < n/2$, 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 mod v_*. If $\mid T - (T_0, T_1) \mid > n/2$ 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

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

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 $T_0, \ldots, T_k$ be all the subtrees of T mod v_*. Let β be the first index for which $\mid T - (T_0, \ldots, T_\beta) \mid \leq n/2$ and let p_{α_i} be the value obtained applying lemma 1 to $T - (T_0, \ldots, T_i)$ (anchored at v_* if i is even) and let T^i_* be $T - (T_0, \ldots, T_i, \ldots T_{2p_{\alpha_i} - \alpha_i})$, for each $i = 1, \ldots, \beta$. We consider the following layout

$$A^i_* = (\overrightarrow{T}_{i+1}(v_{i+1}), \ldots, T^i_*, \ldots, \overleftarrow{T}_{i+2}(v_{i+2}))$$

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

$$A_i = (\overrightarrow{T}_0(v_0), \overrightarrow{T}_3(v_3), \overrightarrow{T}_4(v_4), \ldots, A^i_*, \ldots, \overrightarrow{T}_6(v_6), \overrightarrow{T}_5(v_5), \overrightarrow{T}_2(v_2), \overrightarrow{T}_1(v_1))$$

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

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

if β is even or

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

if β is odd.

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

$$(n_{i+3} + n_{i+4}) + 2(n_{i+5} + n_{i+6}) + + \cdots + (p^i_0 - 1)(n_{2p^i_0 - 1} + n_{2p^i_0}) + p^i_0 n^i_*$$

when i is even and

$$(n_{i+2} + n_{i+3}) + 2(n_{i+4} + n_{i+5}) + + \cdots + (p^i_1 - 1)(n_{2p^i_1 - 2} + n_{2p^i_1 - 1}) + p^i_1(n^i_* + 1)$$

when i is odd.

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

Then the cost of these layouts can be computed as

$$C(A_i) = \sum_{j=0}^{i-1} C[\overrightarrow{T}_j(v_j)] + \sum_{j=i+1}^{2p^i_\alpha - 2\alpha} C[\overrightarrow{T}_j(v_j)] + C[T^i_*] + Q_i + R_i$$

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

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

For an anchored tree we get the following decomposition

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

$$\mid T - (T_0^0, \ldots, T_0^\gamma) \mid \leq n/2$$

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

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

The cost of layout B_i can be computed as:

$$C(B_i) = \sum_{j=0}^{i} C[\overrightarrow{T_0^j}(v_0^j)] + C[T - (T_0^0, \ldots T_0^i)] + n - n_0^i, \quad \text{for } \; i = 0, \ldots, \gamma$$

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 $\mid T \mid /2$. 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 α 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 $\beta, p_0^0, p_1^1, \ldots, p_1^\beta$. 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 γ. 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 $O(3^{log^n} n) = O(n^{2.6})$ taking into account that the sum of the sizes of the trees obtained in the decomposition of a tree T is at most $3 \mid T \mid$. 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 $w_1, ..., w_d$ be the children of v in T_o. Then, the vertex set of T includes vertices $v^1, ..., v^{d+1}$. 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 $\langle \gamma_1, \eta_1, \gamma_2, \eta_2, ... \rangle$ where γ_1 is the largest cut (in $c(u)$) occuring 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. $\langle\overline{rightcost(u)}\rangle$ is a sequence $\langle\gamma'_1, \eta'_1, \gamma'_2, ...\rangle$ where γ'_1 is the largest cut in $c(u)$ 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, $\gamma_1 \geq \gamma_2 \geq ...$, $\eta_1 \leq \eta_2 \leq ...$, $\gamma'_1 \geq \gamma'_2 \geq ...$, $\eta'_1 \leq \eta'_2 \leq ....$ 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 and b be the two subsequences of a cost-sequence $cost$ (e.g., $cost = \langle a, b\rangle$). 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 $leftcost(u) > rightcost(u)$ 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. Let $heavyside_i = \gamma_{1i}, \eta_{1i}, \gamma_{2i}, \eta_{2i}, \cdots$ and $lightside_i = \gamma'_{1i}, \eta'_{1i}, \gamma'_{2i}, \eta'_{2i}, \cdots$, for $i \in \{1, 2\}$. To compare the cost sequences $cost_1$ and $cost_2$, we construct the sequence $compare_i = \langle\gamma^c_{1i}, \eta^c_{1i}, \gamma^c_{2i}, \eta^c_{2i}, \cdots\rangle$, 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 $heavyside_i$, $lightside_i$ respect., 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 $cost_1 < cost_2$ 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 $cost_1 < cost_2$ while if $compare_1$ is of even length then $cost_1 > cost_2$. (Note that the way we compare the two sides of a cost-sequence is different from the way we compare two "*compare*" sequences.)

Let T_u be a tree rooted at a node u and $c(u)$ a layout-sequence realizing a layout φ 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 φ' 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 $O(n^2/\log n)$-processor, $O(d\log^2 n)$-time parallel algorithm which finds a minimum cutwidth linear arrangement of a tree T_o (d is the maximum degree of T_o). The algorithm is based on the use of the parallel tree contraction technique ([ADKP89]). 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 $u_1, ..., u_d$ and T_v a tree

rooted at v with children $v_1, ..., v_{d'}$. 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_{fij} 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_{fij} is done using the layout-sequence $c'(f_{ij})$ while every merge operation of f_{ij} with a vertex of $T_{v0} - T_{fij}$ 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:
$c(u) = (x_1, x_2, ..., x_i, *, x'_{i1}, ..., x'_2, x'_1)$, $c(v) = (y_1, y_2, ..., y_j, *, y'_{j1}, ..., y'_2, y'_1)$
$cost(u) = (\gamma_{1u}, \eta_{1u}, ..., \gamma_{ku}, *, \gamma'_{lu}, ..., \eta'_{1u}, \gamma'_{1u})$, $cost(v) = (\gamma_{1v}, \eta_{1v}, ..., \gamma_{mv}, *, \gamma'_{nv}, ..., \eta'_{1v}, \gamma'_{1u})$
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 $c(u)$, $c(v)$ 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):

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

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

The other subcases are similar.

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

Case 2.1. Suppose that $\gamma_v + \eta_{1u} = \gamma_u$ 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 $c(uv)$ cannot be less than $\gamma_u + 1$. For this reason, we try to spead 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 γ_{1v} (resp., γ'_{1v}). We insert $c(v_{\gamma 1})$ ($c(v_{\gamma' 1})$) in the position of $c(u)$ realizing the cut of η_{1u} (resp., η'_{1u}) and the rest of $c(v)$ in the position of $c(u)$ realizing the cut of η_{2u} (see figure 4). (Notice that the resulting layout is of cost 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) $\forall i \leq k, \gamma_{iv} = \gamma'_{iv}$ (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 γ_{iv}, $1 \leq i \leq k$, we examine if the following holds:

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

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

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

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

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

or

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

We call the above condition (C).

In the sequel, for each γ_{iv}, $1 \leq i \leq k$, we create a node 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 node 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-sequence of T_{uv} with cost γ_u. Otherwise, there is one (or

more) layout-sequence of T_{uv} with cost γ_u. In order to find an optimal one do the following: find the largest η_{ju}, $j \geq i$ satisfying

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

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

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

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 $c(v)$ realizing the cut γ_{kv} into η_{kv}, the cut γ'_{kv} into η'_{ku}, $1 \leq k \leq (i-1)$, and the rest of $c(v)$ into η_{ju} (resp., η'_{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. (For a similar construction see case 3 in the appendix.)

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. In the appendix we present only one other case in order to show how to compute efficiently in parallel the cost-sequence $cost(uv)$, given that we have already computed the layout-sequence $c(uv)$.

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 $c(u)$, $c(v)$).*

Proof (sketch): The correctness is providing 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-sequence with cost $\gamma_u + 1$ (which is the best possible) has cost-sequence larger than or equal to the one given above. For the other cases the correctness comes also easily from their description. ∎

Before we describe the merge-operation B we prove the following lemma concerning the merge-operation A:

Lemma 6 *Let $v_1, \cdots, v_k$ be the children of a node u and let $T_1, \cdots, T_k$ be the subtrees rooted at $v_1, \cdots, v_k$ respectively. Let also $c(v_1), \cdots, c(v_k)$ be the optimal layout sequences of the above subtrees. Suppose that $c(u)$ is the layout-sequence resulting from merging (sequentially) the layout sequences $c(v_1), \cdots, c(v_k)$ with the layout sequence of u, using the merge-operation A. The cost-sequence of $c(u)$ is optimal whichever is the order with which we merge the trees T_{v_i}, $i = 1, \cdots, k$, with the layout sequence of u (using the merge-operation A).*

Proof (idea): By induction on the number k of trees and the fact that the merge-operation A computes an optimal layout-sequence provided that the input sequences are optimal (see lemma 5). ∎

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 $v_1, \cdots, v_d$ and T_u a tree rooted at u with children $u_1, \cdots, u_{d'}$. Let also d' be greater than or equal to d. Suppose that $c(u), c(v)$ are optimal layout sequences of T_u, T_v, respectively. Then, the merge-operation B computes an optimal layout sequence $c(uv)$ that realizes a layout 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 subsequence $c(v_i)$, for each $i \in \{1, \cdots, d\}$, of the layout sequence $c(v)$, induced by the vertices of the subtree T_i rooted at v_i, for each $i \in \{1, \cdots, d\}$. Note that each $c(v_i)$, $i = 1, \cdots, d$, is an optimal layout sequence of T_i, $i = 1, \cdots, d$ (if not then there is another layout sequence $c'(v_i)$ with smallest cost-sequence resulting to another layout-sequence $c'(v)$ for T_v which has smaller cost-sequence than $c(v)$; but $c(v)$ is optimal). Then, sequentially use merge-operation A to merge each one of $c(v_i)$, $i = 1, \cdots, d$, with $c(u)$.

Lemma 7 *Let T_u be a tree rooted at u, T_v a tree rooted at v and $c(u)$, $c(v)$ 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 $O(d \log n)$ 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 lengths of $c(u)$, $c(v)$.*

Proof: From the description of the merge-operation B, and lemmas 6 and 5. ∎

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
$cost(T) = min\{cost(v) \mid v \in T\}$
$MINCUT(T) = c(T)$ {* $c(T)$ 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 $O(d \log^2 n)$ using $O(n^2/\log n)$ CREW PRAM processors, where d is the maximum degree of T.*

Proof: The correctness of the algorithm comes from the correctness of the *shunt* operation, which in turns comes from lemmas 5,6,7 and the fact that the two merge-operations are combined correctly (see the description of the *shunt* operation). The time/processor bounds come from the bounds of the parallel tree contraction technique and lemmas 5 and 7. ∎

References

[ADKP89] K. Abrahamson, N. Dadoun, D. Kirkpatrick and T. Przytycka. A simple parallel tree contraction algorithm. *Journal of Algorithms* 10:287-302, 1989.

[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.

APPENDIX A1

Case 3: $\gamma_u > \gamma_v$ and $c(u)$ balanced and $c(v)$ unbalanced. Let H_u be the heavy side of u and L_u the light one. W.l.o.g. let $H_u = leftcost(u)$ and $L_u = rightcost(u)$. The procedure which gives the cost γ_{uv} of an optimal layout-sequence $c(uv)$ is as follows:

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

In order to find the position between the vertices of 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 $\eta_{iu} + \gamma_v \leq \gamma_{iu} - 1$. Let j be the index of the largest η'_{ju} in L_u for which $\eta'_{ju} + \gamma_v \leq \gamma'_{iu}$. Suppose now that $c(h_{uv})$ $(c(l_{uv}))$ is the layout-sequence we take if we insert $c(v)$ between the vertices - in $c(u)$ - where the cut of η_{iu} (resp., η'_{ju}) occurs. Then, $c(uv)$ 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 $c(v)$ and want to compute the cost-sequence of the layout-sequence $c(uv)$. We consider only the case where $\eta_{1u} + \gamma_v \geq \gamma_u$ and $\eta'_{1u} + \gamma_v \leq \gamma_u$ (see figure 5) (the other cases are similar). Let x'_{ik} be the point in the layout of T_u where the cut η'_{1u} occurs. Then,

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

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

else $cost(uv) = (\gamma_{1u}, \eta_{1u}, \cdots, *, \cdots, \eta'_{2u} + 1, \gamma'_{2u} + 1, \eta'_{1u} + 1, \gamma_v, \eta'_{1u}, \gamma'_{1u})$

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

If $\gamma_{max} = \gamma'_{1v} + 1$ then $cost(uv) = (\gamma_{1u}, \eta_{1u}, \cdots, *, \cdots, \eta'_{2u} + 1, \gamma'_{2u} + 1, \eta_{1u} + 1, \gamma'_{1v} + 1, \eta_{1v}, \gamma_{1u})$.

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 $(\gamma_{mv}, \eta_{(m-1)v}, \cdots, \eta_{2v}, \gamma_{2v})$ 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 $\gamma_{l_1 v} > \gamma'_{1v} + 1$ and $\eta_{l_1 v} > \eta'_{1v} + 1$ (again the other cases are similar). Then, $cost(uv) = (\gamma_{1u}, \eta_{1u}, \cdots, *, \cdots, \eta'_{2u} + 1, \gamma'_{2u} + 1, \eta_{1u} + 1, \gamma'_{1v} + 1, \eta'_{1v} + 1, \gamma_{l_1 v}, \eta_{(l_1-1)v}, \cdots, \gamma_{2v}, \eta_{1v}, \gamma_{1u})$.

III. The other subcases are not more complicated and will not be described.

APPENDIX A2

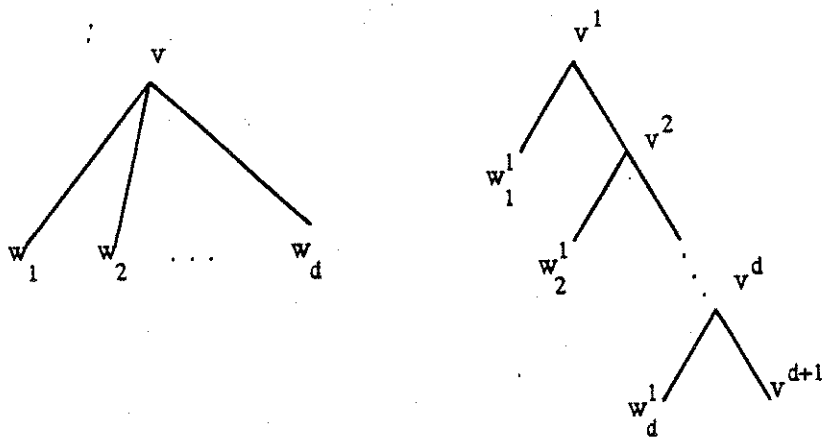

Figure 1:

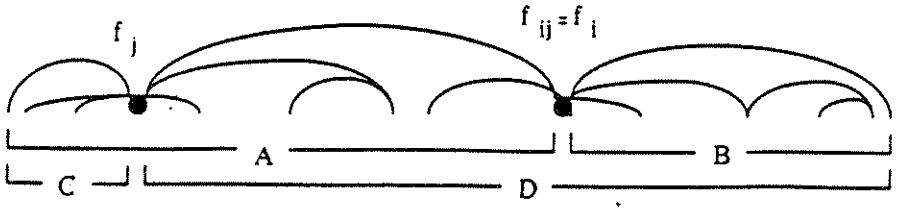

Figure 2:

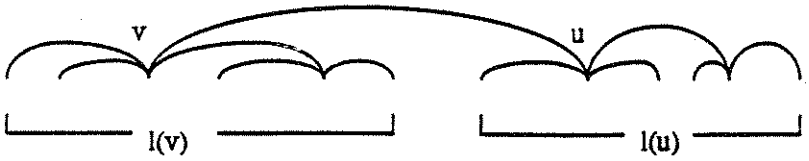

Figure 3:

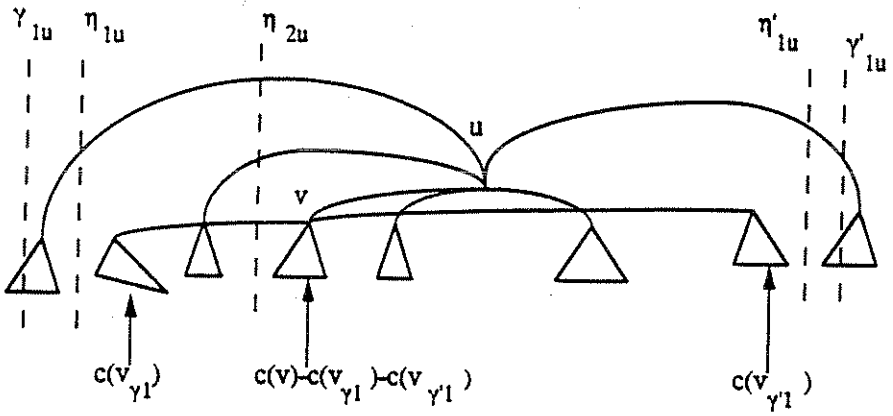

Figure 4:

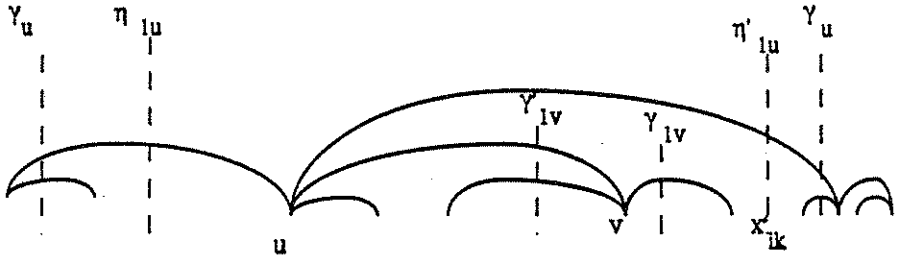

Figure 5: