The MINSUMCUT problem

Josep Díaz
Alan M. Gibbons
Michael S. Patterson
Jacobo Torán

Report LSI–91–11

The MINSUMCUT problem*

J. Díaz† A.M. Gibbons‡ M.S. Paterson‡ J. Torán†

February 13, 1991

Abstract

In this paper we first present a sequential linear algorithm for an linear arrangement problem on trees, the MINSUMCUT, and then a $O(\log\ n)$ time parallel algorithm for the MINSUMCUT on trees, which uses $n^2/(\log n)$ processors.

1 Introduction

Linear arrangement problems have played an important role in computer science ([NK72]) and ([AH73]). Let us recall that a *linear arrangement* (or *layout*) of a graph $G(V,E)$ is an enumeration φ from V onto $\{1,2,\ldots,|V|\}$. Some of the well known arrangement problems are; the minimun linear arrangement problem, (this problem is also know as the EDGE-SUM problem ([Har77])); given a graph $G(V,E)$ find the enumeration φ which minimizes $\sum_{\{u,v\}\in E}|\varphi(u)-\varphi(v)|$. The decision version of this problem is NP-complete for general graphs ([GJ76]), and has a $O(n^{2.2})$ solution for unrooted trees ([Shi79]). Another problem is the MIN-CUT problem; given a graph $G(V,E)$ find the enumeration φ such that for all i $(1 \leq i \leq |V|)$ it minimizes the number of edges in the set $cut_\varphi(i) = \{(u,v0 \in E \mid \varphi(u) \leq i \leq \varphi(v)\}$. This problem is also NP-Complete for general graphs ([GJ79]), and has a polynomial time solution for trees ([Yan83]). The BANDWIDTH problem; given a graph $G(V,E)$ find an enumeration φ which minimizes $|\varphi(u)-\varphi(v)|$ over all the edges $\{u,v\} \in E$. This problem is NP-Complete for general graphs ([GJ79]) and also for trees ([GGJK78]).

In this paper, we consider another layout problem, the minimal sumcut problem, MIN-SUMCUT for short.

Given a graph $G(V,E)$ with a layout φ of V, for any integer $i, 1 \leq i \leq |V|$, let S_i be the set defined by $S_i = \{v \in V \mid \varphi(v) \leq i\}$. Then we define

$$\delta(S_i) = |\{w \in V - S_i \mid \exists v \in S_i, (v,w) \in E\}|$$

Notice that given a graph with n nodes, any layout of it determines in a unique way, a nested sequence $S_0, S_1, \cdots, S_n$. Notice that S_0 is the empty set, S_n is the set of all nodes, and for any i, $1 \leq i \leq n$, $|S_i| = i$.

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

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

‡Departament of Computer Science, University of Warwick, Coventry, CV47AL, UK

Definition 1.1. Given a graph $G(V,E)$, $|V| = n$, the MINSUMCUT problem consists in finding a layout which minimizes

$$\sum_{i=1}^{n} \delta(S_i)$$

where $\{S_i\}_{i=1}^{n}$ is the nested sequence determined by the layout

A related problem, the Delta Operator, was studied in ([Dia79]).

In section 2 we present a linear algorithm to solve the problem for an unrooted tree. In the following section we sketch the proof of the optimality of the algorithm and in the remaining section we present a parallel algorithm to solve the MINSUMCUT problem for trees.

2 The MINSUMCUT problems for trees

The enumeration of a tree with n nodes can be considered as a process of n steps; at step k we assign label k to a node that has not been enumerated yet.

Definition 2.1. At a certain stage of the enumeration a node is said to be *taken* if a label has already been assigned to it. A node is said to be *marked* if one or more of its immediate neighbours have been already enumerated.

For a tree T we will denote by $w(T)$ the number of nodes of the tree.

We introduce three basic enumeration which will be used later by the algorithm. These enumerations are defined for rooted trees. The enumeration algorithm works however for trees in which no orientation is given. For the enumeration of T, the algorithm will first give an orientation to T and then recursively enumerate the different subtrees defined by the orientation. Assume that T' is a subtree of T with root r, and from r hang the subtrees $A_1, \ldots, A_m$. Denote by $f(r)$ the father of r in T. (See figure 1 in the annex.) Depending on whether $f(r)$ has been already been enumerated or marked, we will consider three types of enumeration of the subtrees: β-mode, μ-mode and τ-mode.

The numbering in β-mode will apply when $f(r)$ is neither marked nor enumerated. It consists of choosing in a certain way which optimizes the numbering a subtree A_i, enumerating recursively A_i in β-mode, enumerating the other descendants of r in τ-mode and finally enumerating r. This way of numbering can be considered as the way in which the algorithm starts enumerating a tree, when none of its nodes has been taken.

The numbering in τ mode will apply when $f(r)$ is marked. One of the subtrees of T', A_i, is enumerated in β-mode, then the rest of the subtrees except one are recursively enumerated in τ mode, then the root r and finally the last subtree A_j is enumerated in μ mode. The way to choose in which order the subtrees are enumerated will be explained later. This way of numbering corresponds to the middle of the enumeration when some nodes have already been taken.

The numbering in μ-mode applies when $f(r)$ has been enumerated (and therefore r is marked). In the enumeration all the subtrees except one are enumerated in τ mode, then we enumerate r, and finally the last subtree of r is enumerated recursively in μ mode. This way of numbering corresponds to the end of the enumeration when the algorithm is finishing the enumeration of a subtree in which the root is already taken.

The cost of such enumerations can be computed in the following way:

Let T' be a subtree of T with root r. If T' is a leaf then $cost(\beta(T')) = 1$, $cost(\tau(T')) = 1$, and $cost(\mu(T')) = 0$. Observe that in the first two cases we have to count node $f(r)$, which by hypothesis is not taken, while in the μ case, this node is already taken.

If T' is not a leaf, and it consists of a root r and subtrees $A_1, \ldots, A_m$ hanging from r, the costs of the enumerations are:

- $cost(\beta(T')) = min_i\{cost(\beta(A_i)) + \sum_{j \neq i} cost(\tau(A_j)) + 1\}$.
- $cost(\tau(T')) = min_{i,j(i \neq j)}\{cost(\beta(A_i)) + cost(\beta(A_j)) + \sum_{k \neq i,j} cost(\tau(A_k)) + w(T')\}$ if $m \geq 2$.
- $cost(\tau(T')) = cost(\beta(A_1)) + w(T')$ if $m = 1$.
- $cost(\mu(T')) = min_i\{cost(\mu(A_i)) + \sum_{j \neq i} cost(\tau(A_j)) + 1\}$.

The following technical lemmas, which will be used later, follow from the definition of the enumerations.

Lemma 2.2. For every tree T, $cost(\beta(T)) = cost(\mu(T)) + 1$.

Lemma 2.3. For every tree T, $cost(\beta(T)) + w(T) - 1 \geq cost(\tau(T))$.

Lemma 2.4. For every tree T, if $w(T) > 1$ then $cost(\tau(T)) > cost(\beta(T))$.

We give now a description of the algorithm, which finds the optimal enumeration in linear time. An example of the algorithm can be followed in figures 1a, 1b and 1c, in the annex.

Algorithm Part 1: Input a tree T. Output Cost(T).

1. For every undirected edge $e = (u, v)$ of T, replace e by two directed edges $e_1 = (u, v)$ and $e_2 = (v, u)$.

   We will associate with every directed edge $e = (u, v)$ the subtree with root u defined when node v is deleted from the tree. We will denote this subtree by T_e.

   For every directed edge e denote by $w(e)$ the number of nodes in the subtree T_e.

2. To every directed edge e we will assign a 5-tuple $[B(e), C(e), F(e), S(e), W(e)]$. The first two parameters are values which denote the cost of a β and a τ enumeration of T_e, respectively. The third and fourth parameters are pointers to the two direct descendants of the root of T_e, T_{e_1} and T_{e_2} which have the smallest value $(C(e_i) - B(e_i))$. These two pointers will be needed later since T_{e_1} will be the first subtree taken in the case of a β or a τ enumeration and the last taken in the case of a μ enumeration, while T_{e_2} will be the last subtree enumerated in the τ case. $W(e)$ will denote the number of nodes of the subtree T_e.

   The values B, C, F, S are recursively computed in the following way:

(a) If $e = (u, v)$ and u is a leaf of T then $B(e) = 1$ $C(e) = 1$, and $F(e)$ and $S(e)$ are the empty string.

(b) If $e = (u, v)$ and u is not a leaf then let $e_1, e_2, \ldots, e_m$ be the edges pointing to u, not considering the inverse edge of e. Assign to $F(e)$, $S(e)$, $W(e)$, $B(e)$ and $C(e)$ the values

$F(e)$ is a pointer to the direct descendant of u, T_{e_i}, such that for every $j \neq i$ $(C(e_i) - B(e_i)) \geq (C(e_j) - B(e_j))$.

$S(e)$ is a pointer to the direct descendant of u, T_{e_j} $(j \neq i)$ such that for every $k \neq i, j$ $(C(e_j) - B(e_j)) \geq (C(e_k) - B(e_k))$.

$W(e) = 1 + \sum_i W(e_i)$ with $1 \leq i \leq m$

$B(e) = B(e_i) + \sum_{j \neq i} C(e_j) + 1$ with $e_i = F(e)$.

$C(e) = B(e_i) + B(e_j) + \sum_{k \neq i,j} C(e_k) + W(e)$ with $e_i = F(e)$ and $e_j = S(e)$.
If $m = 1$ then $C(e) = B(e_i) + w(e)$

3. $\text{Cost}(T) = min_{(u,v) \in E}\{B(u, v) + B(v, u)\} - 1$.

Algorithm Part 2: Input a tree T, and the output of part 1.

Let $e_1 = (u, v)$ the edge selected by the previous part to minimize cost, and let $e_2 = (v, u)$.

1. Recursively do the following:
   (a) enumerate T_{e_1} in β mode.
   (b) enumerate T_{e_2} in μ mode.
2. For a subtree T_e in a β enumeration do:
   (a) enumerate $F(e)$ in β mode.
   (b) enumerate the rest of the subtrees of T_e in τ mode.
   (c) enumerate the root of T_e.
3. For a subtree T_e in a τ enumeration do:
   (a) enumerate $F(e)$ in β mode.
   (b) enumerate the rest of the subtrees of T_e (except $S(e)$) in τ mode.
   (c) enumerate the root of T_e.
   (d) enumerate $S(e)$ is μ mode.
4. For a subtree T_e in a μ enumeration do:
   (a) enumerate all the subtrees of T_e (except $F(e)$) in τ mode.
   (b) enumerate the root of T_e.
   (c) enumerate $F(e)$ is μ mode.

3 Optimality of the enumeration

In this section we shall prove that the algorithm given in the previous section produces an enumeration which has optimal cost. For this we need the following definitions:

Definition 3.1. Let T be a tree and $\{a_1, \ldots, a_i\}$ and $\{b_1, \ldots, b_j\}$ be two disjoint sets of nodes of T. $neigh(a_1, \ldots, a_i)$ denotes the set of nodes which are neighbours in $T - \{a_1, \ldots, a_i\}$ of $\{a_1, \ldots, a_i\}$. $cost(a_1, \ldots, a_i|_{b_1,\ldots,b_j})$ denotes the cost of enumerating $\{a_1, \ldots, a_i\}$ in this order, suposing that $\{b_1, \ldots, b_j\}$ have been already enumerated. If $\{a_1, \ldots, a_i\} = T$ then $cost(a_1, \ldots, a_i)$ denotes the cost of enumerating T in this order.

Theorem 3.2. Let T be an undirected tree. Let s be an interior node of T, and $T_1, \ldots, T_i$ be the set of subtrees defined by deleting s. Let φ be an enumeration that does not start and finish enumerating T by the same subtree T_i. There is an enumeration φ' of cost smaller or equal than φ, and such that for every T_j φ' enumerates all the nodes from T_j consecutively, before enumerating any node from a different subtree.

Proof. Let φ be an enumeration of T. We will show that for any subtree T_s defined by deleting node s in T, if $\varphi(1) \notin T_s$ or $\varphi(|T|) \notin T_s$, then there is an enumeration φ' satisfying the following properties:

- φ' enumerates all the nodes of T_s consecutively,
- $cost(\varphi') \leq cost(\varphi)$, and
- for every pair of nodes u, v which belong to the same subtree of T (defined by deleting node s), if φ enumerates u and v consecutively, then φ' also enumerates u and v consecutively.

Theorem 3.2 follows from this fact.

We proceed by induction on the size of T_s. The induction basis (T_s has just one node) is trivial. For the induction step, suppose that T_s is composed of a root r (hanging from s) and subtrees $A_1, \ldots, A_m$ hanging from r. Let φ be an enumeration of T, and suppose that $\varphi(1) \notin T_s$ or $\varphi(|T|) \notin T_s$. Since all the subtrees A_i have size smaller than T_s, by induction hypothese there is an enumeration φ' which for every i enumerates all the nodes of A_i consecutively, and has cost smaller than or equal to the cost of φ.

We show now that there is another enumeration which takes all the nodes of T_s consecutively. The proof is divided into different claims which study all the possible ways in which φ can enumerate T_s. In this version of the paper the proofs of the claims are omitted. They are not hard to prove using a case analysis of the costs of the different enumerations considered.

Claim 1. The enumeration φ defined by

$$a_1, \ldots, a_i, A_1, \ldots, A_j, b_1, \ldots, b_k, A_{j+1}, c_1, \ldots, c_l$$

with $\{a_1, \ldots, a_i\} \cap T_n = \emptyset$ and $\{b_1, \ldots, b_k\} \cap T_n = \emptyset$ has a cost greater than or equal to an enumeration φ' in which $A_1, \ldots, A_j, A_{j+1}$ are enumerated consecutively. Also, for every pair of nodes u, v which belong to the same subtree of T (defined by deleting node s), if the first emuneration takes u and v consecutively, then so does φ'.

Claim 2. The enumeration φ defined by

$$a_1, \ldots, a_i, r, b_1, \ldots, b_k, A_j, c_1, \ldots, c_l$$

with $\{b_1, \ldots, b_k\} \cap T_n = \emptyset$ has a cost greater than an enumeration φ' in which A_j is enumerated directly after r. Also, for every pair of nodes u, v which belong to the same subtree of T (defined by deleting node s), if the first emuneration takes u and v consecutively, then so does the φ'.

Claim 3. The enumeration $a_1, \ldots, a_i, r, A_j, b_1, \ldots, b_k$ where A_j is not the last tree enumerated from T_s has a cost greater than the enumeration $a_1, \ldots, a_i, A_j, r, b_1, \ldots, b_k$. Also, for every pair of nodes u, v $(u, v \neq r)$ which belong to the same subtree of T (defined by deleting node s), if the first emuneration takes u and v consecutively, then so does the second.

Claim 4. The enumeration given by

$$a_1, \ldots, a_i, A_1, \ldots, A_{m-1}, b_1, \ldots, b_k, r, A_m, c_1, \ldots, c_l$$

where $A_1, \ldots, A_m$ are all the subtrees in T_s hanging from the root, and $\{a_1, \ldots, a_i, c_1, \ldots, c_l\} \neq \emptyset$ has a cost greater than or equal to an enumeration in which $A_1, \ldots, A_{m-1}, r, A_m$ are enumerated consecutively in this order. Also, for every pair of nodes u, v which belong to the same subtree of T (defined by deleting node s), if the first emuneration takes u and v consecutively, then so does the second.

Claim 5. The enumeration $a_1, \ldots, a_i, A_1, \ldots, A_m, b_1, \ldots, b_k, r, c_1, \ldots, c_l$ where $A_1, \ldots, A_m$ are all the subtrees in T_s hanging from the root, has a cost greater than or equal to an enumeration φ' in which $A_1, \ldots, A_m, r$ are enumerated consecutively in this order. Also, for every pair of nodes u, v which belong to the same subtree of T (defined by deleting node s), if the first emuneration takes u and v consecutively, then so does the φ'.

Theorem 2.3 follows from the previous claims, since they consider all possible ways of enumerating T. After any one of the claims is applied we obtain a new enumeration of cost smaller or equal than the previous one, and in which some new set of nodes of T_s is enumerated consecutively, maintaining a consecutive enumeration of nodes of the same subtree which were consecutively taken by the first enumeration. The fact that the first and last nodes taken by the first enumeration do not belong to the same subtree is used in claim 4 (the fact implies $\{a_1, \ldots, a_i, c_1, \ldots, c_l\} \neq \emptyset$). □

Lemma 3.3. Let φ be an enumeration of T and let s be an interior node of the tree such that φ does not start and finish enumerating T in the same subtree defined by deleting s. There is an enumeration of T, φ' of cost less than or equal to φ such that for every subtree T_j (defined by deleting s), all the nodes of T_j are consecutively enumerated by φ' in the following way:

1. If in the stage in which the first node of T_j is enumerated, s has not been marked nor enumerated then T_j is enumerated in β mode.

2. If in the stage in which the first node of T_j is enumerated, s has been marked but has not been enumerated then T_j is enumerated in τ mode.

3. If in the stage in which the first node of T_j is enumerated, s has been taken then T_j is enumerated in μ mode.

Proof. Let s be a node of T. By theorem 3.2, we know that there is an enumeration which for every subtree T_j defined by deleting s, all the nodes of T_j are consecutively enumerated by it.

Let T_s be one of these subtrees. We argue by induction on the size of the subtree T_s. The induction basis ($|T_s| = 1$) is a trivial case, since there is just one possible enumeration of T_s. For the induction step, suppose that T_s is composed by a root r (hanging from s) and subtrees $A_1, \ldots, A_m$ hanging from r. Since the nodes of T_s are consecutively enumerated all the possible enumerations of T have the form $A_1, A_2, \ldots, A_i, r, A_{i+1}, \ldots, A_m$.

1. If s is neither marked nor taken, by induction hypothesis the cost of such an emumeration is

$$cost(\beta(A_1)) + \sum_{k=2}^{i} cost(\tau(A_k)) + 1 + (m-i) + \\ + \sum_{k=i+1}^{m} [cost(\mu(A_k)) + (m+1-k) \times w(A_k)]$$

   Using lemmas 2.2 to 2.4, it follows that the minimum value for this expression is for the case $i = m$, i.e, an enumeration in which the root is the last node of T which is taken. This is the case of a β enumeration.

2. If s is marked but not taken, the cost of such an emumeration is

$$cost(\beta(A_1)) + w(A_1) + \sum_{k=2}^{i} [cost(\tau(A_k)) + w(A_k)] + 1 + (m-i) + \\ + \sum_{k=i+1}^{m} [cost(\mu(A_k)) + (m+1-k) \times w(A_k)]$$

   The minimum value for this expression is for the case $i = m-1$ i.e. when the root is the node taken before the last one, (τ enumeration).

3. If s is taken, the cost of the emumeration is

$$\sum_{k=1}^{i} cost(\tau(A_k)) + (m-i) + \sum_{k=i+1}^{m} [cost(\mu(A_k)) + (m-k) \times w(A_k)]$$

   Again the minimum value for this expression is for the case $i = m$, i.e, an enumeration in which the root is taken just before the last subtree This is the case of a μ enumeration.

□

Definition 3.4. An edge e of T is called a bridge for an enumeration φ, if φ starts at T_{e_1} and finishes at T_{e_2} where T_{e_1} and T_{e_2} are the two subtrees generated when e is deleted from T.

Lemma 3.5. Let T be an undirected tree and let φ be an optimal enumeration of T. Let e be a bridge for φ. Let s be the root of T_{e_1}. There is an optimal enumeration φ' in which s is enumerated by φ' after all its subtrees but T_{e_2} have been enumerated.

Proof. Let $T_1, T_2, \ldots T_m$ be the set of subtrees of T defined by deleting s. (Observe that T_{e_2} is one of the subtrees of this list). Since e is a bridge for φ, and s is one of the nodes of e, by lemma 3.3 there is an optimal enumeration φ' of T enumerating every subtree T_i either in β, τ or μ mode. We can suppose that φ starts the enumeration by T_{e_1} and finishes in T_{e_2} and therefore, the last subtree enumerated by φ' is T_{e_2}. φ' can enumerate T in order $T_1, T_2, \ldots, T_i, n, T_{i+1}, \ldots, T_{m-1}, T_{e_2}$, and the cost of the enumeration is

$$cost(\beta(A_1)) + w(A_1) + \sum_{k=2}^{i} [cost(\tau(A_k)) + w(A_k)] + (m-i) +$$

$$+ \sum_{k=i+1}^{m-1} [cost(\mu(A_k)) + (m-k) \times w(A_k)] + \mu(T_{e_2})$$

Using lemmas 2.2 to 2.4, it follows that the minimum value for this expression is for the case $i = m-1$, i.e, an enumeration in which s is taken after all the subtrees but T_{e_2} have been enumerated. Observe that in this case the cost of the enumeration is exactly $cost(\beta(T_{e_1})) + cost(\mu(T_{e_2}))$.

□

Lemma 3.6. Let T be an undirected tree. An enumeration φ of T is optimal iff

$$cost(\varphi) = \min_{(u,v) \in E} \{cost(\beta(T_{(u,v)})) + cost(\beta(T_{(v,u)}))\} - 1$$

Proof. If φ is optimal and e is a bridge for φ, then by lemma 2.2, $cost(\varphi) = cost(\beta(T_{e_1})) + cost(\mu(T_{e_2})) = cost(\beta(T_{e_1})) + cost(\beta(T_{e_2})) - 1$.

If there is an edge $a \in E$ with $cost(\beta(T_{a_1})) + cost(\beta(T_{a_2})) < cost(\varphi) + 1 = cost(\beta(T_{e_1})) + cost(\beta(T_{e_2}))$, enumerating T_{a_1} in β mode and T_{a_2} in μ mode, we would obtain an enumeration φ' starting at some leaf of T_{a_1} and finishing at T_{a_2} with cost $cost(\beta(T_{a_1})) + cost(\beta(T_{a_2})) - 1 < cost(\varphi)$ contradicting the fact that φ is an optimal enumeration. □

4 A parallel algorithm for the MINSUMCUT problem on trees

Throughout this section we assume the CREW PRAM model of parallel computation (see [GR88] for example). Theorem 4.3 places the MINSUMCUT problem on trees in the complexity class NC^1 using $n^2/(\log n)$ processors. Throughout this section we use the terminology employed in the description of the sequential algorithm.

Theorem 4.1. For any tree, given the edge e_1 and values $F(e)$, $S(e)$ and $W(e)$ for all edges e of the tree as determined by the preprocessing step (step 1) of the sequential algorithm, we can find an optimal enumeration of the tree in $O(\log n)$ parallel time using n processors.

Proof. As for part 2 of the sequential algorithm, the optimal enumeration requires us to enumerate T_{e1} in β mode followed by an enumeration of T_{e2} in μ mode. It is sufficient to show that given any rooted tree T (with k nodes), then we can find a φ enumeration of T (where φ is β, τ or μ) in $O(\log k)$ time using k processors.

The algorithm starts on the basis of the following observation. Given any subtree $T_{(u,v)}$ (rooted at u and where $y = f(u)$) suppose that the sons of u are $(s_1, s_2, \ldots, s_m)$, $F(u,v) = s_1$ and $S(u,v) = s_m$. In a "stand-alone" evaluation of $T_{(u,v)}$:

(a1) if the evaluation is in β mode, then u will have the enumeration $(\sum_{i=1}^{m} W(s_i)) + 1)$

(b1) if the evaluation is in τ mode, then u will have the enumeration $(\sum_{i=1}^{m-1} W(s_i)) + 1)$

(c1) if the evaluation is in μ mode, then u will have the enumeration $(\sum_{i=2}^{m} W(s_i)) + 1)$

When enumerating T, suppose that $T_{(u,v)}$ is enumerated in β mode. Then u will have the absolute enumeration of $(\sum_{i=1}^{m} W(s_i)) + 1 + NB(u)$ where $NB(u)$ denotes the number of nodes of T enumerated before $T_{(u,v)}$ is enumerated. At the start of the algorithm we do not know in which mode $T_{(u,v)}$ is required to be enumerated for an enumeration of T and, moreover, we do not know the value of $NB(u)$. By the end of the algorithm, for a given mode of enumeration of T, both these unknowns will have been determined. The algoritm starts with three copies of each subtree of T which is of height 1 (one copy for each possible mode of enumeration) and proceeds to join trees together to form larger trees, these larger trees will generally be of twice the height of those joined together.

After at most $\lceil \log \text{ (depth of } T) \rceil = O(\log n)$ parallel joining phases, T will be reconstructed. At each stage of the process, for each of the three copies of any constructed tree, the following three invariants hold:

(a2) For every non-leaf node v, $E(v)$ is the order in which v is visited in a "stand alone" enumeration of the constructed tree.

(b2) Each leaf l of the constructed tree knows $N(l)$, the number of nodes of this tree which would have been enumerated before the subtree rooted at l in T would be enumerated.

(c2) Each leaf l knows what mode of enumeration would be required (along with the two values F and S) for the subtree rooted at l in T.

The construction process starts with the subtrees of height 1. Then initialising the $E(v)$ to be the values given appropriately by (a1), (b1) and (c1) will mean invariant (a2) holds at the outset. In order that invariant (b2) holds for the trees of height 1, we need to assign to the $N(l)$ values obtained from an appropiate prefix sum of the Ws of the sons in these trees. Which prefix sum is appropriate will depend (in an obvious way) on the prevailing mode of enumeration and the local F and S. Initially invariant (c2) obviously holds.

Before describing the overall pattern of rejoining trees to ensure that $O(\log k)$ parallel joining steps are all that is required, we need to describe how just two (copies of) trees are joined together to ensure that the invariants (a2), (b2) and (c2) will hold. Suppose that a leaf of T_1 is identified with the root of T_2 to produce T_3. For a copy of T_1 corresponding to one enumeration, the leaf of T_1 identified with the root of T_2 will determine which copy of T_2 is required in order that invariant (c2) holds in T_3. In order that invariant (a2) continues to hold, we add to each $E(v)$ in T_3 that used to belong to T_2 the value of $N(z)$ where z is the leaf of T_1 identified with the root of T_2. Similarly, adding $N(z)$ to each $N(u)$, where u are those leaves of T_3 which used to be leaves of T_2, will ensure that invariant (b2) continues to hold. Notice that thesse additions require concurrent reads of $N(z)$.

We have described what operations are necessary when joining two trees. Consider now what trees are joined together in one parallel step of tree joining. First determine at what level of T each internal vertex resides. We assume that the root is at level 1. Let $S(i)$ denote the set of all roots of constructed trees that are to be identified with leaves of the other trees in the ith step of parallel tree joining. We specify that $S(i)$ contains all those roots at depth d in T such that $(d - 2^{i-1})$ is exactly divisible by 2^i. Thus $S(1)$ contains roots at depth $(2, 4, 6, 8, \ldots)$, $S(2)$ those at depths $(3, 7, 11, 15, \ldots)$, $S(3)$ those at depths $(5, 13, 21, 29, \ldots)$ and so on. This construction of one copy of T is schematically shown in figure 3. Because of invariant (a2), when each copy of T has been constructed, we have each of the three possible enumerations of T. A similar logarithmic reconstruction of a tree was employed in ([GR90]).

The construction of the initial trees of height 1 may be achieved in constant time with $O(k)$ processors and this is easily simulated in $O(\log k)$ time using $k/(\log k)$ processors. Their initialisation is only complicated by the need to perform some prefix sums, but there are well known optimal logarithmic-time algorithms (see [GR88], for example) for this type of computation. Each parallel tree joining phase takes constant time with $O(k)$ processors ($O(k)$ additions are all that is essentially required) and since there are $O(\log k)$ phases, the overall reconstruction can be achieved in $O(\log k)$ time using k processors. □

Theorem 4.2. For a given directed edge e, the values of $B(e)$, $C(e)$, $F(e)$, $S(e)$ and $W(e)$ required for a φ enumeration of T_e can be found in $O(\log k)$ time using $k/(\log k)$ processors, where T_e has k nodes.

Let us give an sketch of the proof.

Proof. The value of $W(e)$ is the number of descendents of u, where $e = (u, v)$. It is well known (for example using Euler tour technique, see [GR88]) that the number of descendants of all nodes of a rooted tree with k nodes may be found in $O(\log k)$ using $k/(\log k)$ processors. If T_e is a binary tree then we note that $B(e)$, $C(e)$, $F(e)$ and $S(e)$ can be found by employing a straightforward variation of the logarithmic-time, optimal expression evaluation algorithm of ([GR89]). In general T_e is not a binary tree. This complication may be handled by local reconstruction of the tree (to produce a binary version) and by employing a more complicated version of ([GR89]). The local reconstruction is indicated in figure 4. In the algorithm of ([GR89]), one functional form (which we call $f_v(x)$) is associated with each internal node v of the tree which is modified during so-called "leaves-cutting" phases. The $f_v(x)$, when evaluated, is the current value that v must pass to its father if the value of the subtree rooted at v is x. In the more complicated version of the algorithm required here, when v needs to be

re-formulated in a leaves-cutting phase, there may be more than one $f_v(x)$ associated with v (although only one in each phase) and more rules for re-formulation. The complications arise because $f_v(x)$ may, for example, need reformulation as v is associated with a leaf whose father is coalescent with v in the original tree or whose father is actually the father of v in the original tree. The complications, although straightforward, are technically intrincate and so we omit the details. With carefully arranged book-keeping, each leaves-cutting operation still takes constant time and so, just like in ([GR89]), the algorithm takes $O(\log k)$ time using $k/(\log k)$ processors. □

Theorem 4.3. The MINSUMCUT problem for any tree T, with n nodes, can be solved in $O(\log n)$ time using $n^2/(\log n)$ processors.

Proof. In order to solve the MINSUMCUT problems for trees, we need first to identify an edge (u,v) which provides a minimum value of the expression $(B(u,v)+B(v,u))$, as stated in part 1 of the sequential algorithm. There are $O(n)$ edges in T so that these expressions may be evaluated after $O(\log n)$ time using $n^2/(\log n)$ processors by applying the algorithm of theorem 4.2 to all the edges in parallel. Finding the minimum is easily achieved in $O(\log n)$ using $n/(\log n)$ processors by standard algorithms. Having found an appropriate (u,v) and using F, S and W found by the algorithm of theorem 4.2, we can employ the algorithm of theorem 4.1 to find the enumeration. □

Although we have demostrated that there is a logarithmic time algorithm for the MINSUMCUT problem on trees, we would like to reduce the number of processors used. The bottle-neck in this regard is the evaluation of $(B(u,v)+B(v,u))$ for all edges of T. In fact, in $O(\log n)$ time using $n/(\log n)$ processors it is possible to find the Bs for half of the directed edges of T. This task is accomplished by a further modification of the algorithm in theorem 4.2. The modification is equivalent to that required to the algorithm used for expression evaluation ([GR89]) when there is a need to evaluate all the subexpressions represented by the internal nodes of the expression tree. After the root has been evaluated, it is possible to reconstruct the tree by the process which is the reverse of leaves-cutting. As internal nodes re-appear in the tree, prior book-keeping can ensure that the associated expressions are evaluated and overhall this requires the same time and processor requirements as leaves-cutting. The implication is that, if T is rooted arbitrarily, the Bs may be found for all edges directed towards the root by one application of the modified algorithm. The problem, however, is that an (arbitrary) tree would still have to be rooted $O(n)$ times to capture all the edges.

References

[AH73] D. Adolphson and T.C. Hu. Optimal linear ordering. *SIAM J. Apply Mathematics*, 25(3):403–423, Nov 1973.

[Dia79] J. Diaz. The δ-operator. In L. Budach, editor, *Fundamentals of Computation Theory*, pages 105–111. Akademie-Verlag, 1979.

[GGJK78] M.R. Garey, R.L. Graham, D.S. Johnson, and D. Knuth. Complexity results for bandwith minimization. *SIAM J on Applied Mathematics*, 34:477–495, Sept. 1978.

[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.

[GR88] A. Gibbons and W. Rytter. *Efficient Parallel Algorithms.* Cambridge University Press, Cambridge, 1988.

[GR89] A. Gibbons and W. Rytter. Optimal parallel algorithms for dynamic expression evaluation and context free recognition. *Information and Computation*, 81(1):32–45, April 1989.

[GR90] A. Gibbons and W. Rytter. Optimal edge-colouring outerplanar graphs is in NC. *Theoretical Computer Science*, 71:401–411, 1990.

[Har77] L.H. Harper. Stabilization and the edgesum problem. *ARS Combinatoria*, 4:225–270, Dec. 1977.

[NK72] M. Nanan and M. Kurtzberg. A review of theplacement and quadratic assignment problems. *SIAM Review*, 14(2):324–341, April 1972.

[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 *FOCS*, volume 24, pages 274–281, Providence RI, Nov. 1983.

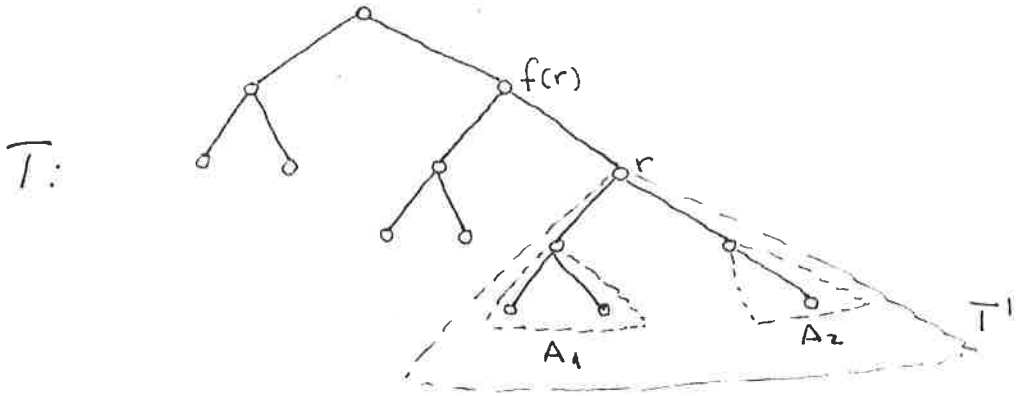

Figure 1. Example of rooted tree T with subtree T'. T' has itself two subtrees A_1 and A_2 hanging from its root r.

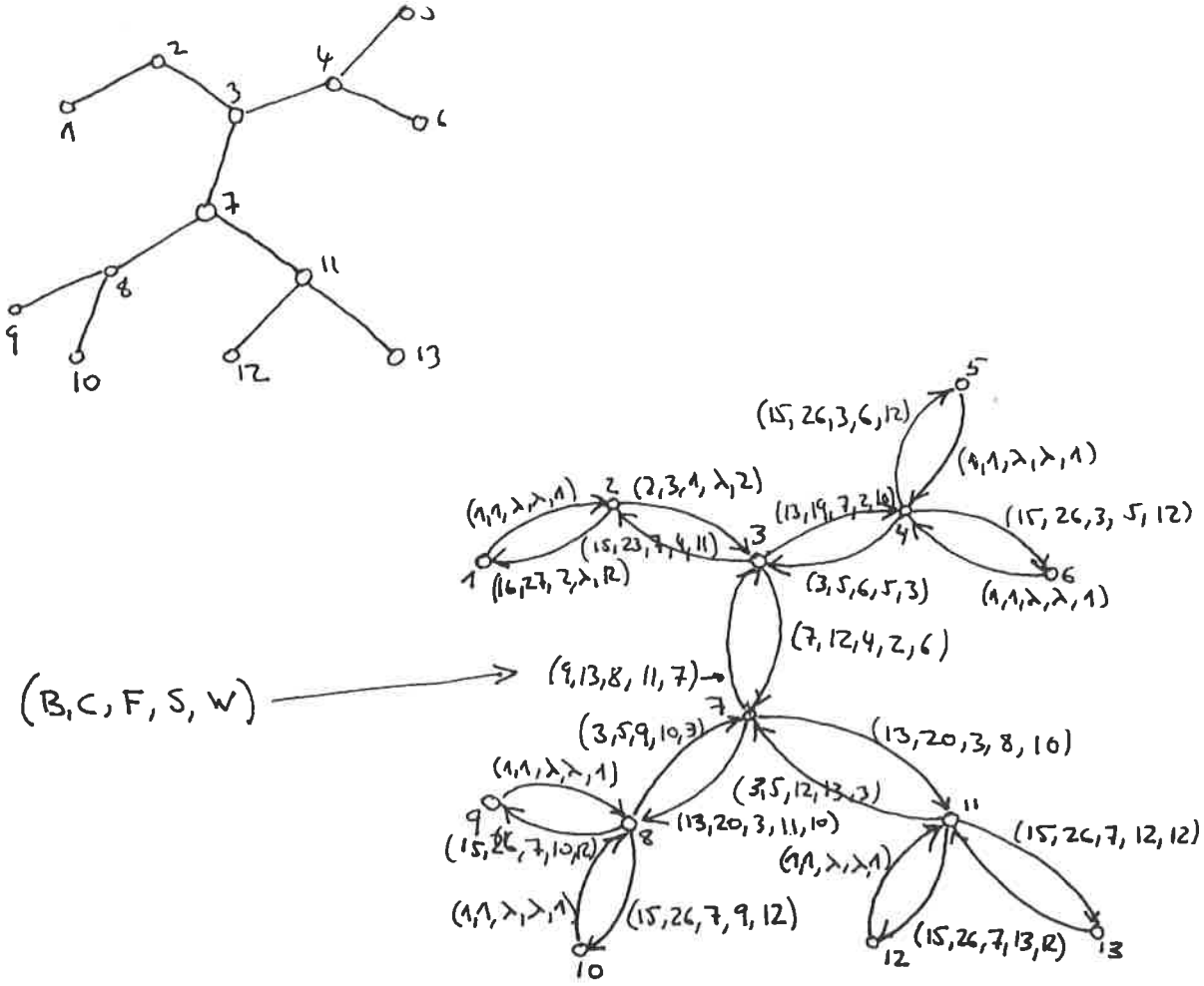

Figure 2a. An unrooted tree T, and the result of applying part 1 of the sequential algorithm.

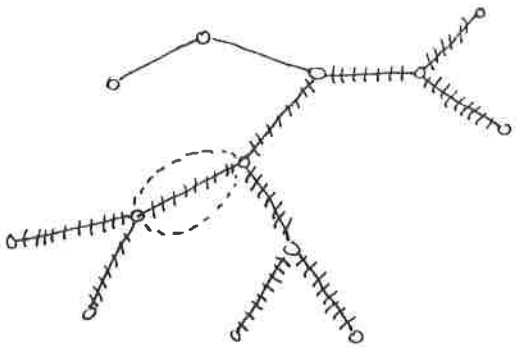

Figure 2b. In shadow the edges which minimize $\{B(u,v)+B(v,u)\}$. The algorithm selects one of them, for example the one surounded with doted line.

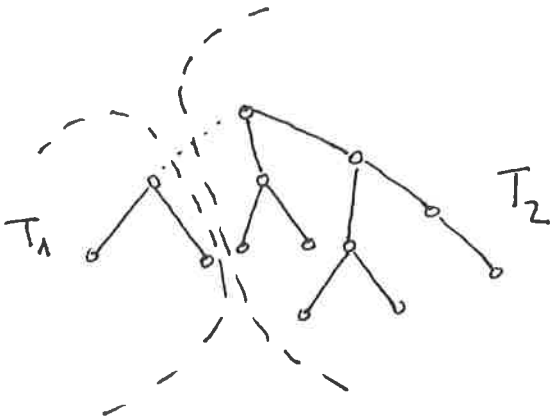

Figure 2c. Enumerate T_1 in β mood and T_2 in μ mood. The resulting enumeration is: 9,10,8,12,11,13,7,1,2,3,5,4,6, and the cost is 16.

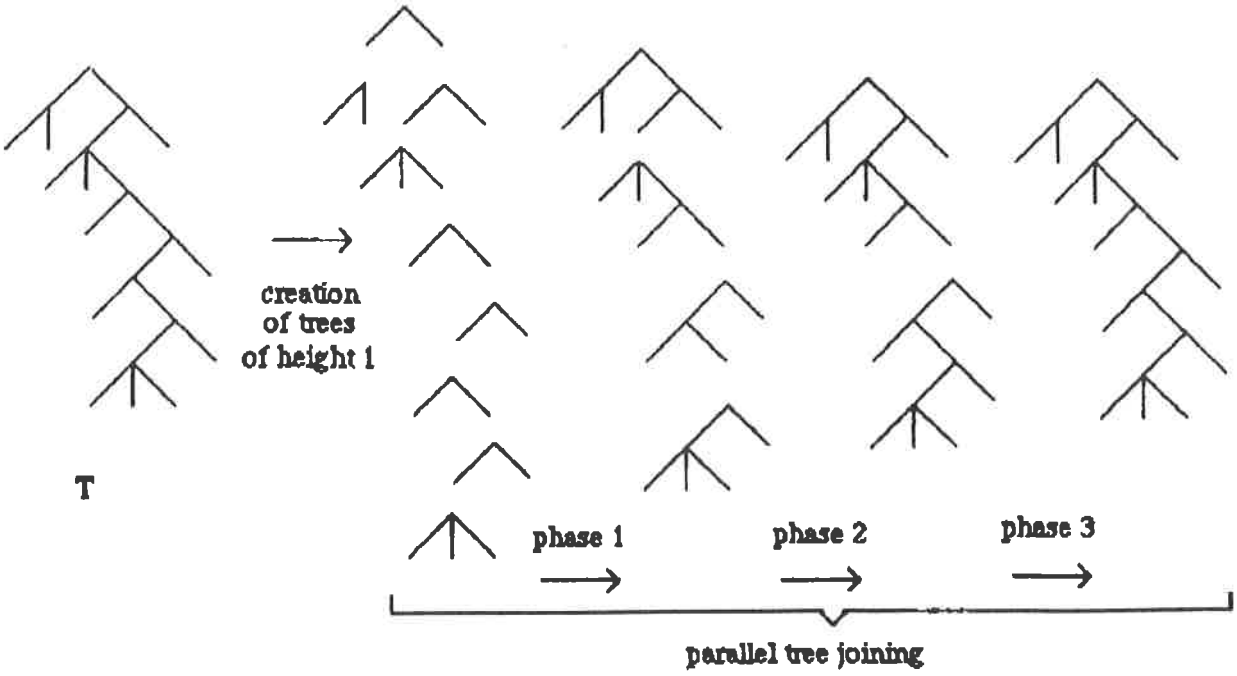

Figure 3

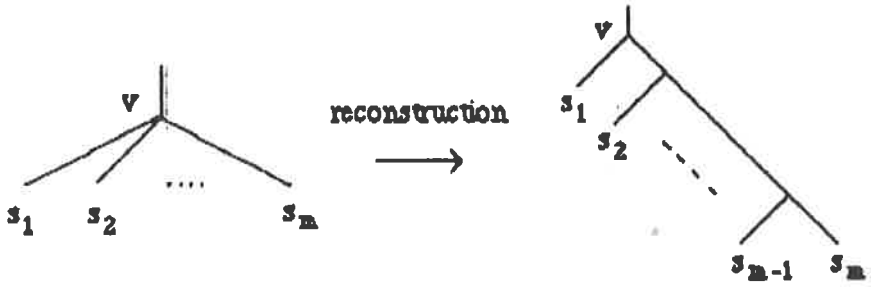

Figure 4