Comput. Math. Applic. Vol. 15, No. 4, pp. 259–266, 1988
Printed in Great Britain.

0097-4943/88 $3.00 + 0.00

A PARALLEL ALGORITHM FOR BISECTION WIDTH IN TREES

M. GOLDBERG
Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12181, U.S.A.

Z. MILLER
Department of Mathematics and Statistics, Miami University, Oxford, OH 45056, U.S.A.

Abstract—The bisection width $b(G)$ of a graph G is the number of edges necessary in an edge cut of G so that the two sides of the cut have equal (within one element) size. Our main result is an $O(n^2)$ processor $O(\log^2 n \log\log n)$ time parallel algorithm for determining $b(T)$ when T is a tree.

1. INTRODUCTION

The bisection width of a graph G is the number of edges necessary in an edge cut of G so that the two sides of the cut have equal (within one element) size. The problem of finding the bisection width of a given graph is termed the graph bisection problem (GBP) known to be NP-hard [1]. In particular, it was shown in Ref. [2] that GBP is NP-hard even for the class of graphs with maximum degree 3. Generalizations have been considered in Refs [3, 4].

We consider GBP for trees and call it TBP. In his thesis, MacGregor showed an $O(n^3)$-time algorithm for TBP. With some modification, the running time can be reduced to $O(n^2)$, though we omit the details here. Our main result is a parallel algorithm for TBP which runs in polylog time on $O(n^2)$ number of processors.

Let T be a tree and let $z \in V(T)$. Then, a vector $W(z, T) = [w_1(z, T), w_2(z, T), \ldots, w_{n-1}(z, T)]$ is defined as follows:

> $w_i(z, T)$ is the minimum number of edges cut by a partition $(X_i, \bar{X}_i)$ such that (1) $z \in X_i$; (2) $|\bar{X}_i| = i$ and $|\bar{X}_i| = n - i$; $(i = 1, 2, \ldots, n-1)$.

MacGregor showed that the computation of $W(x, T)$ can be reduced to that of $\{W(x_i; T_i)\}$ $(i = 1, \ldots, k)$, where $\{T_i\}$ are the connected components of the graph $T - x$, $\{x_i\}$ is the set of neighbors of x in T, and $\{x_i \in V(T_i)\}$ $(i = 1, \ldots, k)$. Clearly, the computation of $\{W(x_i; T_i)\}$ can be done in parallel. However, this does not result in an algorithm of polylog running time because, in general, the number of consecutive steps equals the height of the tree and hence could be as large as a linear function.

A natural way to reduce the number of such basic stages of the algorithm is to always use the centroid of the tree. Since the size of each T_i is halved at each stage, the number of the stages is $O(\log n)$. The difficulty that arises is that the centroids of the T_is are not always adjacent to the centroid of T itself, so the reduction used by MacGregor cannot be performed. *In order to overcome this difficulty, we change the set of subtrees used for the reduction and generalize the problem.*

A branch of a vertex z in T is a maximal subtree containing z as a terminal vertex. Thus, a branch of z can be obtained by joining z to a component of $T - z$.

Let A and B be two disjoint subsets of the $V(T)$. For every i with the property $n - |B| \leqslant i \leqslant |A|$, define $w_i(T; A; B)$ as the minimum number of edges cut by a partition $(X_i, \bar{X}_i)$ such that $|X_i| = i$, $A \subset X_i$ and $B \subset \bar{X}_i$. If $i < |A|$ or $n - 1 < |B|$, then set $w_i(T; A; B) = \infty$. Finally, define $W(T; A; B)$ by

$$W(T; A; B) = [w_1(T, A, B), \ldots, w_{n-1}(T, A, B)].$$

The tool for reduction used by MacGregor is an operation called min-convolution. We now define it slightly differently.

Let $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_m)$ be two vectors. For each k, $1 \leqslant k \leqslant n + m - 1$,

let $c_k = \min\ (a_i + b_j\colon i + j = k + 1)$, and define the min-convolution of A and B by $MC(A, B) = (c_1, c_2, \ldots, c_{m+n-1})$. For an arbitrary set $A_1, A_2, \ldots, A_l$ of vectors, define $MC\ (A_1, \ldots, A_l)$ recursively by

(i) if $l = 1$, then $MC(A_1) = A_1$,
(ii) if $l > 1$, then $MC(A_1, \ldots, A_l) = MC(MC(A_1, \ldots, A_r), MC(A_{r+1}, \ldots, A_l))$,
where $r = \left\lfloor \frac{l}{2} \right\rfloor$.

We show in Section 2 that for any A and B, $W(T; A; B)$ can be calculated using the min-convolution of similar vectors associated with the branches of T. To implement this reduction in polylog time, we have to reduce the overhead time which is spent on the assignment of processors to different jobs that are to be done in parallel. The recursive character of the general reduction idea makes the assignment problem difficult because the sizes of the arrays to be processed are not known in advance. Thus, the jobs to be done by processors must be assigned in the course of the algorithm's work. For example, we use repeatedly a procedure that constructs, for a given subtree, its centroid and the centroid's branches. Then, the procedure is applied to each of the branches. In order to execute it in parallel, without overblowing the number of processors required, it is necessary to have a rule which would reassign the set of the processors used for the original tree to the branches so that each branch would get a sufficient number of processors. A redistribution of this kind turned out to be possible for the centroid decomposition.

Our algorithm is executed on CRCW P-RAM parallel computer [5–7]. All graph theoretic notions not defined here, may be found in Ref. [8].

2. CENTROID DECOMPOSITION

Given a tree T and two disjoint subsets A and B of $V(T)$, we define a new tree called $\mathrm{Str}(T; A; B)$ (for structure of T) that expresses a centroid decomposition of T.

If T is the trivial tree K_1, then $\mathrm{Str}(T; A; B) = T$. Otherwise, let $c(T)$ be a centroid vertex of T and $\{T_i, 1 \leqslant i \leqslant k)\}$ be its branches in T. For each $i, 1 \leqslant i \leqslant k$, let

$$A_i = A \cap V(T_i), \quad B_i = B \cap V(T_i), \quad A_i^+ = A_i \cup \{c(T)\}, \quad B_i^+ \cup \{c(T)\}.$$

Set a new node z to be the root of $\mathrm{Str}(T; A; B)$ and make it adjacent to the roots of either

$$\{\mathrm{Str}(T_i; A_i; B_i);\}\ (1 \leqslant i \leqslant k) \quad \text{if} \quad c(T) \in A \cup B,$$

or

$$\{\mathrm{Str}(T_i; A_i^+; B_i); \mathrm{Str}(T_i; A_i; B_i^+)\}\ (1 \leqslant i \leqslant k)$$

otherwise.

In addition, assign each node of $\mathrm{Str}(T; A; B)$ a vector $W(T'; A'; B')$, where T' will be a subtree of T and $A' = A \cap V(T')$; $B' = B \cap V(T')$.

Thus $\mathrm{Str}(T; A; B)$ is a tree whose nodes correspond to the branches of centroids (at various levels), and whose edges, $\alpha\beta$, correspond to pairs of branches such that β is a branch of a centroid in α. Since each $\alpha \in \mathrm{Str}(T)$ may have two centroids, we must choose one of these and we let this choice be artibrary. The roots of $\mathrm{Str}(T_i; A_i^+; B_i)$ and $\mathrm{Str}(T_i; A_i; B_i^+)$ are called *siblings* $(1 \leqslant i \leqslant k)$. Note also that the height of $\mathrm{Str}(T; A; B)$ is $O(\log n)$, where $n = |V(T)|$.

Our objective is to compute the vector $w(T; \emptyset, \emptyset)$, whose entry $w_{n/2}$ is the desired bisection width $b(T)$. We will show that $W(T; A; B)$ can be computed fast from the vectors assigned to the nodes adjacent to the root of $\mathrm{Str}(T; A; B)$.

First we consider the case of $c(T) \in A \cup B$.

Proposition 1

Let z be the root of $\mathrm{Str}(T; A; B)$, $\{z_i\}$ be the nodes adjacent to $z (i = 1, \ldots, k)$, and $c(T) \in A \cup B$. If W_i is the vector assigned to z_i $(i = 1, \ldots, k)$, then

$$W(T; A; B) = MC(W_1, \ldots, W_k).$$

Proof. Every partition $\sigma = (X, Y)$ such that $A \subset X$ and $B \subset Y$, induces partition $\sigma_i = (X \cap V(T_i), Y \cap V(T_i))$ of $V(T_i)$ such that

$$A_i \subset X_i \text{ and } B_i \subset Y_i \quad (i = 1, \ldots, k). \tag{1}$$

Obviously, if σ is minimal, then for every $i = 1, \ldots, k$, $\sigma_i = (X_i, Y_i)$ is a minimal partition of $V(T_i)$ such that condition (1) is satisfied. Moreover, the minimality of σ will be preserved if any σ_i is substituted for another minimal partition $\tau_i = (P_i, Q_i)$ of $V(T_i)$, for which the conditions $|P_i| = |X_i|, |Q_i| = |Y_i|$, and condition (1) hold true. The result follows. ■

To handle the case $c(T) \in A \cup B$, we need one further notation. Given two vectors $U = (u_1, \ldots, u_m)$ and $V = (v_1, \ldots, v_m)$ of the same length, define $W = \min(U, V) = (w_1, \ldots, w_m)$ by

$$w_i = \min(u_i, v_i), \quad i = 1, \ldots, m.$$

Proposition 2

Let $c(T) \in A \cup B$, let z be the root of $\mathrm{Str}(T; A; B)$ and $\{z_i^+, z_i^-\}$ $(i = 1, \ldots, k)$, be the set of siblings adjacent to z. If W_i^+ (resp. W_i^-) is the vector assigned to z_i^+ (resp. z_i^-) $(i = 1, \ldots, k)$, and $W^+ = MC(W_1^+, \ldots, W_k^+)$, $W^- = MC(W_1^-, \ldots, W_k^-)$, then

$$W(T; A; B) = \min(W^+, W^-).$$

Proof. Consider two auxiliary problems P and Q given respectively by the triples $(T; A \cup c(T); B)$ and $(T; A; \cup c(T))$. Obviously, for each of these problems, statement 1 is applied, which generates two vectors W^+ and W^- respectively. For each $j = 1, \ldots, n$ $(n = |V(T)|)$, the minimal size of a partition $(X, \bar{X})$ with $|X| = j$ is $\min(w_j^+, w_j^-)$. ■

3. DATA STRUCTURE

The statements of the preceeding section give the principal idea of our algorithm for TBS. To show that the algorithm runs in polylog time on $O(n^2)$ processors, we have to define our data structure in more precise terms. For this purpose, we use terminology adopted in PASCAL [9, 10] with two major changes.

The first change is concerned with the lengths of the arrays. The specification of the array's lengths is allowed to be made at the moment preceding the first use of the array. For example, the type *node* is a record containing, in particular, three items called *cuts*, *sign* and *listing*. All three are defined to be of the same type, which is a sequence of integers. However, the lengths of these sequences are different; they can also be different from those of other nodes. Certainly, such practice is usual for mathematical definitions.

The second novelty reflects the parallel architecture of the computer which is used for solving the problem. Every variable of type node is supplied by a set of processors and a block in common memory; the sets for different nodes are disjoint.

A node represents a triple $(R; A; B)$ where R is a subtree of the original tree T and A and B are disjoint subsets of $V(S)$. A sequence called *listing* gives the vertex set of R. A sequence called *cuts* contains the vector $W(R; A; B)$. Subsets A and B are described by the sequence *sign*. For every $v \in A$ (resp. $v \in B$), sign $(v) = 1$ (resp. sign $(v) = -1$); for all other v, sign $(v) = 0$. A sequence of pointers, called *children*, points from the node to other nodes, which will also be called children. The subtrees, associated with the children, are the result of the centroid decomposition of R, i.e. they are the branches of a centroid of R.

Every node contains two pointers, called *parent* and *sibling*. The corresponding nodes are also called parent and sibling. Obviously, the root of $\mathrm{Str}(T; A; B)$ has neither parent nor sibling; the corresponding pointers are *nil*. The parent of any other node is not nil, but the sibling can be nil. For any node N, if node M is the sibling of N, then the sibling of M is N. Both M and N have the same parent; both represent the same tree, which is a branch of the tree represented by the parent. The difference between the siblings is in the sets (A_1, B_1) and (A_2, B_2) associated with them. For some vertex c, which is a centroid of the parent's tree, the following holds true:

$$A_2 \cup \{c\} = A_1; \quad B_1 \cup \{c\} = B_2.$$

A node N which represents a triple $(S; A; B)$ with $|V(S)| = 1$ is called *terminal*.

For every node, a pair is attached consisting of a set of processors and an area in common memory; this pair is called the *allowance* of the node. We assume that the cells of the memory are linearly ordered and the processors are numbered by integers from 1 to $L = C_1(n-1)^2$ (here n is the vertex number of the tree and C_1 is a constant). We also assume that both items of the allowance are intervals; thus they are given by two pairs of integers (μ_1, μ_2) and (π_1, π_2). Finally, an item called *time* shows the distance from the root to the node. It is used for the synchronization of the actions executed by processors assigned to the node with those of other processors.

A formal, PASCAL-like definition of the data types is as follows:

```
TYPE  arrow = @node;
      series = array [1..length] of integer;
      block = record
                listing, sign, cuts:
                          series;
              end;
      interval = record
                   first, last: integer;
                 end;
      pair = record
               interval.1, interval.2: interval;
             end;
      fan = array [1..length] of arrow;
      node = record
               content: block;
               children: fan;
               parent, partner: arrow;
               allowance: pair;
               time: integer;
             end;
      tree = arrow;
```

Thus, a tree is defined by a pointer to a node, which is the root of the tree. If a node N represents a triple $(S; A; B)$, then we use $\mathrm{Str}(N)$ instead of $\mathrm{Str}(S; A; B)$.

Note that when a procedure of the algorithm is applied to node N with allowance $[\Pi, \Omega]$, then the execution is performed by the processors in Π using space in Ω.

4. TOOLS

Two basic tools of our algorithm are procedures MinCon, which calculates the min-convolution of a set of vectors, and DECOMPOSE, which constructs the set of children of a given node.

The input of MinCon is a sequence of integer vectors $A_1, A_2, \ldots, A_l$. The output is the vector $MC(A_1, A_2, \ldots, A_l)$, as it is defined in the introduction.

We first consider the case of $l = 2$. Denote $A_1 = A = (a_1, \ldots, a_n)$, $A_2 = B = (b_1, \ldots, b_m)$. Also, define $B(k)$ by

$$B(k) = \begin{cases} k, & \text{if } k \leqslant \min(n, m), \\ \min(n, m), & \text{otherwise.} \end{cases}$$

MinCon works on A and B as follows:

```
begin
  in parallel {(*first part*)
    for k:=1 to n+m-1 do
      in parallel {
        for i:=1 to n do
          for j:=1 to m do
```

```
                if i+j=k+1 then
                  processor P_ij computes a_i+b_j
                  and writes the result into the tapesquare k^2+i
                  of common memory}}
        in parallel {(*second part*)
          for k:=1 to n+m-1 do
          compute the minimum of the numbers written
          on tape squares k^2 through k^2+B(k)
          of common memory
                }
end
```

Time Estimation

(1) Assuming $O(1)$ time for addition and multiplication, we see that the first part of the procedure is executed in a constant time by $O(n^2)$ processors. The amount of space used is also $O(n^2)$. Note that the number of pairs (i,j) such that $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$ and $i+j=k+1$ is given by the function $B(k)$. Since $x^2+B(k)<(k+1)^2$, no write conflict occurs.

(2) By minimization algorithm of Refs [11, 12], the running time of the second part is $O(\log\log(n+m))$ and the space needed is $O(nm)$. We can also arrange, at no further cost in time or processors, to have the output $MC(A,B)$ written into some specified interval of common memory, say, square t through $t+n+m-1$, for some $t \geqslant 1$.

The procedure for an arbitrary number of vectors is now straightforward.

```
Procedure MinCon (MC; A_1, A_2, ..., A_l);
begin
    r:=⌊l/2⌋;
    in parallel {
      A*:=MinCon (A_1, ..., A_r);
      B*:=MinCon (A_{1+r}, ... A_l);
              }
    MC:=MinCon (A*, B*)
end;
```

Proposition 3

Let $n_1, n_2, \ldots, n_l$ $(l \geqslant 2)$ be the lengths of the vectors $A_1, A_2, \ldots, A_l$ respectively and let

$$P = \sum_{i=1}^{l} n_i; \quad Q = \sum_{i \neq j} n_i n_j.$$

Then, MinCon can be executed in $O(\log\ l \times \log\log P)$ time using $O(Q)$ processors and $O(Q)$ space.

Proof. Evidently, there are $O(\log l)$ stages of MinCon; at each stage, the min-convolutions of a number of pairs of vectors with a combined length $\leqslant P$ is calculated in parallel. This implies a running time of $O(\log l \times \log\log\ P)$.

Assume now that calculating A^* and B^* can be done using $Q_1 = O(\Sigma n_i n_j)$ and $Q_2 = O(\Sigma n_i n_j)$ space respectively, where the first (resp. the second) sum is taken over all $1 \leqslant i \neq j \leqslant r$ (resp. $r+1 \leqslant i \neq j \leqslant l$). Since the additional space needed for calculating MinCon (A^*, B^*) is $O(\Sigma n_i n_j)$, where $1 \leqslant i \leqslant r$ and $r+1 \leqslant j \leqslant 11$, the statement is easily proved. ■

Now we explain the work of DECOMPOSE. The input to this procedure is a node N and the output is the set of children-node of N. Let $[\Pi, \Omega]$ be the allowance of N and assume that $|\Pi| = C_1(k-1)^2$ and $|\Omega| = C_2(k-1)^2$, where C_1 and C_2 are constants and k is the number of vertices of the subtree S represented by the node N.

DECOMPOSE uses the algorithm constructed by Tarjan and Vishkin in Ref. [13] which we call procedure T–V. It accepts a pair $(S; x)$, where S is a subtree of T and $x \in V(S)$ and

(i) lists the vertices in depth-first order (with respect to x);
(ii) calculates, for every $y \in V(S)$, the size of the subtree of $R(y)$ which grows from y.

T–V requires $O(n)$ processors and runs in $O(\log n)$ time.

Let node N represent a triple $(S; A; B)$. If N is terminal, then DECOMPOSE defines pointers children as nil; no new node is constructed. For a nonterminal node, DECOMPOSE starts by listing all the vertices of S in depth-first order with respect to the vertex of S with the smallest index (this is done by T–V). Then for every node, DECOMPOSE calculates in parallel the size of its largest branch. This allows the procedure to find a centroid, $c(S)$, of S. After, that T–V is repeated with respect to the same tree, but with $c(S)$ as the root. Using the information supplied by T–V, DECOMPOSE constructs the branches of $c(S)$, and for each of them computes the corresponding sets A' and B'.

If $c(S) \in A \cup B$, then for every branch S_i, a node N_i is created which represents triple $[S_i; A \cap V(S_i); B \cap V(S_i)]$; in the case $c(S) \in A \cup B$, for each branch S_i, DECOMPOSE constructs two siblings N_i^+ and N_i^- $(1 \leqslant i \leqslant k)$. In both cases, items listing and sign representing the vertices and the associated subsets are computed according to the definition in Section 2. The value of time for every node is, evidently, one more than that of the parent. The item cuts which represents the W-vector is not computed at the moment. When DECOMPOSE is applied, there is no need to know the value of this vector. Thus, DECOMPOSE defines all entries of the sequence cut as -1.

Each children-node is given its own set of processors, which is a subset of Π, and an area in Ω. Let $\bar{N}_1, \bar{N}_2, \ldots, \bar{N}_p$ be the children-nodes constructed by DECOMPOSE. Obviously, $r \leqslant p \leqslant 2r$, where r is the number of branches of the centroid $c(S)$. Let $\bar{n}_i = |V(S_i)|$, where S_i is the tree represented by node $\bar{N}_i$, $(i = 1, \ldots, p)$. We construct partitions

$$\Pi = \Pi_1 \cup \Pi_2 \cup \cdots \Pi_p \cup \bar{\Pi}; \quad \Omega = \Omega_1 \cup \Omega_2 \cup \cdots \Omega_p \cup \Omega; \tag{2}$$

so that

$$|\Pi_i| = C_1(n_i - 1)^2$$

and

$$|\Omega_i| = C_2(n_i - 1)^2,$$
$$(i = 1, \ldots, p).$$

The redistribution of the processors and memory is then done by setting the allowance for node $\bar{N}_i$ to be $[\Pi_i \Omega_i]$ $(i = 1, \ldots, p)$.

To show that the partitions above exist, we prove the next result.

Proposition 4

Let $n, n_1, n_2, \ldots, n_k$ be such that

$$1 + \sum_{i=1}^{k} (n_i - 1) = n \quad \text{and} \quad \frac{n+1}{2} \geqslant n_1 \geqslant n_2 \geqslant \cdots \geqslant n_k. \tag{3}$$

Then

$$2\left(\sum_{i=1}^{k} (n_i - 1)^2\right) \leqslant (n-1)^2. \tag{4}$$

Proof. The statement is trivially correct for $n = 2$ or 3. Consider

$$X = \sum_{i=1}^{k} (n_i - 1)^2,$$

where $\{n_1, \ldots, n_k\}$ satisfy condition (3). It is easy to prove that the maximum of X is achieved when $k = 2$ and

$$n_1 = n_2 = \frac{n+1}{2}.$$

The statement follows. ■

Now we show that partitions (2) can be constructed in parallel in polylog time. Without loss of generality, we can assume that $C_1(n-1)^2$ processors assigned to node N are numbered consecutively, starting with 1. Let $t_i = (\bar{n}_i - 1)^2$ $(i = 1, \ldots, p)$ and assume that $t_1 \geqslant t_2 \geqslant \cdots \geqslant t_p$. From condition (4) it follows that

$$C_1\left(\sum_{i=1}^{p} t_i\right) \leqslant C_1(n-1)^2.$$

Our aim is to compute

$$l_k = C_1 \sum_{i=1}^{k} t_i$$

for every $k = 1, \ldots, p$. Having done that, we will assign to node N_i processors with indexes in the interval $[l_{i-1}, l_i - 1]$, where $l_0 = 0$. An algorithm that solves this problem is as follows:

Solve recursively the problem for inputs $\{t_1, \ldots, t_q\}$ and $\{t_{q+1}, \ldots, t_p\}$, where

$$q = \left\lceil \frac{p+1}{2} \right\rceil.$$

If $\{f_1, \ldots, f_q\}$ and $\{g_{q+1}, \ldots, g_p\}$ are the outputs, then $l_i = f_i$ for all $i = 1, \ldots, q$, and $l_j = f_q + g_j$, for all $j = q+1, \ldots, p$.

Assume there are $O(p)$ processors executing this procedure and denote $\lambda(p)$ as the running time. Evidently, $\lambda(p) \leqslant \lambda(q) + 1$, implying $\lambda(p) = O(\log p)$.

Now, the time estimation for DECOMPOSE is simple. The procedure runs in $O(n)$ time using $O(n^2)$ processors and a proportional amount of space. ■

5. ALGORITHM

A top level description of the algorithm is as follows:

```
ALGORITHM TreeCut;
begin
  INITIALIZE(N0);
  DISPLAY(N0, STR);
  COMPUTE;
end.
```

INITIALIZE accepts a triple $(T; \emptyset; \emptyset)$ and defines the node N_0 representing this triple. N_0 is the root of the tree STR.

DISPLAY is a recursive procedure which accepts a node N and constructs tree STR(N) with N as its root. When applied to N_0, it produces tree STR used by COMPUTE as the input.

Starting from the bottom of STR, COMPUTE calculates item cuts for every node of the tree. Below, we describe the procedures in detail.

```
Procedure DISPLAY(N: node; var Str: tree);
begin
    if N is terminal then
        Str: = @N
                        (*Str is a pointer to N*)
    else
     begin
      DECOMPOSE(N; N1, N2, ..., Np);
      in parallel{
        for i: = 1 to k do
          DISPLAY(Ni);
                }
```

```
        end
    end
end;
```

COMPUTE starts after STR is constructed. The running time of DECOMPOSE, when applied to a tree with $\leqslant n$ vertices, is $O(\log n)$. It allows us to set, for the purpose of synchronization, the same time, $C_3 \log n$, for every execution of this procedure. Thus, the total time for execution of DISPLAY equals $C_3 t_{\max} \log n$, where $t_{\max}$ is the maximal *time* of a node in STR. We then set the start of COMPUTE at the moment $T_0 = 1 + C_3 t_{\max} \log n$.

COMPUTE works in $t_{\max}$ stages. During ith-stage, COMPUTE calculates vectors cuts for the nodes with $time = t_{\max} - i + 1$. If a node is terminal (for example, when $i = 1$), the determination of cuts is trivial. Let N be a nonterminal node, $N_1^+, \ldots, N_p^+$ and $N_1^-, \ldots, N_p^-$ be the children-nodes of N (N_i^+ and N_i^- are siblings), and let $n_1, n_2, \ldots, n_p$ be the lengths of the vectors cuts corresponding to them. COMPUTE calculates in parallel the min-convolution of the cuts of the $(+)$ nodes, and that of the $(-)$ nodes. If W^+ and W^- are the results, then vector cuts of node N is $\min(W^+, W^-)$. Calculation of each vector, W^+ and W^-, is done in $O(\log n \log\log n)$ time and requires

$$\leqslant 2C_4 \sum_{i \neq j} n_i n_j$$

processors and space by Proposition 3. Since

$$2 \sum_{i \neq j} n_i n_j \leqslant \left(\sum_i n_i \right)^2,$$

we see that all computations are done using not more than the allowance of the node.

Proposition 5

The running time of *TreeCut* is $O(\log^2 n \log\log n)$ on $O(n^2)$ processors and the amount of space used is $O(n^2)$.

Proof. Clearly, DISPLAY runs in $O(t_{\max} \times \log n)$ time and COMPUTE runs in $O(t_{\max} \times \log n \times \log\log n)$ time. Since $t_{\max} = O(\log n)$, the statement follows. ■

Acknowledgements—Supported in part by National Science Foundation under Grant DCR-8520872 (Goldberg) and by ONR under Grant N00014-85-K-0621 (Miller).

REFERENCES

1. M. R. Garey, D. S. Johnson and L. Stockmeyer, Some simplified NP-complete graph problems, *Theor. Comput. Sci.* **1,** 237–267 (1976).
2. R. M. MacGregor, On partitioning a graph: a theoretical and empiric study. Ph.D. Thesis, Univ. of California, Berkeley (1978).
3. E. R. Barnes and A. J. Hoffman, Partitioning, spectra and linear programming. In *Progress in Combinatorial Optimization* (Ed. W. Pulleyblank). Academic Press, New York (984).
4. T. H. Bui, S. Chaudhuri, T. Leighton and M. Sipser, *Graph Bisection Algorithms with Good Average Case Behavior*, FOCS (1984).
5. S. A. Cook, A taxonomy of problems with fast parallel algorithms. *Inf. Control* **64,** 2–22 (1985).
6. N. Pippenger, On simultaneous resource bounds (preliminary version). *Proc. 20th IEEE FOCS* pp. 307–311 (1979).
7. U. Vishkin, Synchronous parallel computation—a survey. Preprint, Courant Institute, New York University (1983).
8. F. Harary, *Graph Theory*. Addison–Wesley, Reading, Mass. (1969).
9. A. V. Aho, J. E. Hopcroft and J. D. Ullman, *Data Structure and Algorithms*. Addison–Wesley, Reading, Mass. (1982).
10. P. Grogono, *Programming in Pascal*. Addison–Wesley, Reading, Mass. (1982).
11. L. G. Valiant, Parallelism in comparison problems. *SIAM Jl Comput.* **4** 348–355 (1975).
12. Y. Shiloach and U. Vishkin, Finding the maximum, merging and sorting in a parallel computational model. *J. Algorithms* **2,** 88–102 (1981).
13. R. E. Tarjan and U. Vishkin, Finding bicomponents and computing tree functions in logarithmic parallel time, *Proc. 25th Symp. of Computer Science*, pp. 12–20 (1984).