ELSEVIER Discrete Applied Mathematics 87 (1998) 187–201

DISCRETE APPLIED MATHEMATICS

An optimal parallel algorithm for node ranking of cographs

Chuan-Ming Liu, Ming-Shing Yu*

Department of Applied Mathematics, National Chung-Hsing University, Taichung 40227, Taiwan

Received 28 September 1996; received in revised form 5 March 1998; accepted 16 March 1998

Abstract

A *ranking* of a graph G is a mapping, ρ, from the vertices of G to the natural numbers such that for every path between any two vertices u and v, $u \neq v$, with $\rho(u) = \rho(v)$, there exists at least one vertex w on that path with $\rho(w) > \rho(u) = \rho(v)$. The value $\rho(v)$ of a vertex v is the *rank* of vertex v. A ranking is *optimal* if the largest rank assigned is the smallest among all rankings. The optimal ranking problem on a graph G is the problem of finding an optimal ranking on G. We persent a parallel algorithm which needs $O(\log n)$ time and $n/\log n$ processors on the EREW PRAM model for this problem on cographs. © 1998 Elsevier Science B.V. All rights reserved.

1. Introduction

In this paper we propose a parallel algorithm for the node (vertex) ranking problem on cographs. Consider a finite, undirected graph $G = (V, E)$ where V is the vertex set and E is the edge set. A *ranking* of G is a mapping, ρ, from the vertices of G to the natural numbers such that for every path between any two vertices u and v, $u \neq v$, with $\rho(u) = \rho(v)$, there exists at least one vertex w on that path with $\rho(w) > \rho(u) = \rho(v)$. The value $\rho(v)$ of a vertex v is the *rank* of vertex v. A ranking is *optimal* if the largest rank assigned is the smallest among all rankings. And the *ranking number*, $r(G)$, of a graph G is the largest rank assigned in any optimal ranking of G. The optimal ranking problem on a graph G is the problem of finding an optimal ranking on G. Fig. 1 shows a ranking and an optimal ranking on a graph. The constraints for a ranking imply that two adjacent vertices cannot have the same rank. Hence this problem is a restriction of the node coloring problem. Furthermore, it is obvious that there is exactly one vertex with the largest rank in a ranking [8].

The node ranking problem has interesting applications in communication network design, planning efficient assembly of products in manufacturing systems [8, 15, 18, 22] and VLSI Layout [12, 19]. Furthermore, the problem of finding an optimal vertex ranking is equivalent to the problem of finding the minimum height eliminating tree

* Corresponding author. Fax: +886-4-2873028; e-mail: msyu@dragon.nchu.edu.tw.

0166-218X/98/$19.00 © 1998 Elsevier Science B.V. All rights reserved.
PII S0166-218X(98)00056-0

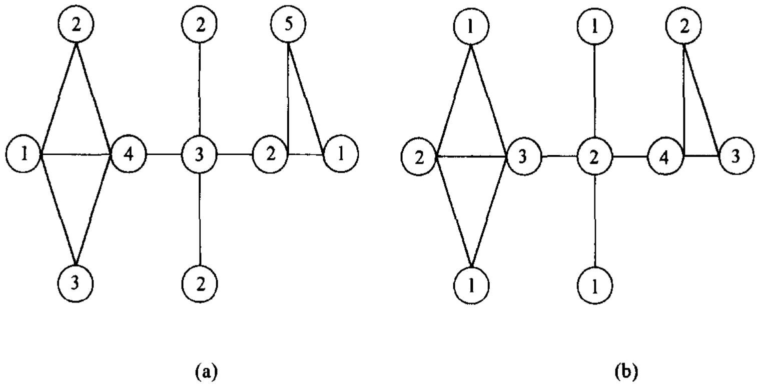


Fig. 1. A ranking (a) and an optimal ranking (b) on a graph.

of a graph [6, 22]. This measure is important for the parallel Cholesky factorization of matrices [2, 7, 14].

The complexity of the optimal ranking problem is still under investigation for many graph classes. This problem is NP-complete for cobipartite graphs [16] and bipartite graphs [1]. On the other hand, there are many polynomial-time sequential algorithms for this problem on several special classes of graphs: circular permutation graphs, interval graphs, circular arc graphs, trapezoid graphs and cocomparability graphs of bounded dimension [6], trees [8, 17], split graphs, cographs [18], and graphs with treewidth at most k [1].

As for parallel algorithms, to the best of our knowledge, there are some parallel algorithms for this problem on trees [13, 20, 21], but they are not optimal with respect to the $O(n)$ algorithms in [17]. In [13], Liang, Dhall and Lakshmivarahan indicated that this problem appeared highly sequential in nature and speculated that it might be P-complete.

We will present an $O(\log n)$ optimal parallel algorithm using $n/\log n$ processors on EREW PRAM using the tree contraction methods in [11, 23] and the Euler tour technique in [4]. Our result can also be applied to the parallel algorithms for the pathwidth and treewidth problem in cographs.

The organization of this paper is as follows. In Section 2 we present some preliminaries and definitions. The parallel algorithm will be given in Section 3. Section 4 contains some concluding remarks.

2. Preliminaries

In this section we give some definitions and preliminary results related to the optimal ranking problem on cographs. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. The *union* of G_1 and G_2 is $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. The *complete interconnection* of G_1 and G_2 is $G_1 \times G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{\{u,v\} \mid u \in V_1 \text{ and } v \in V_2\})$, where $\{u,v\}$

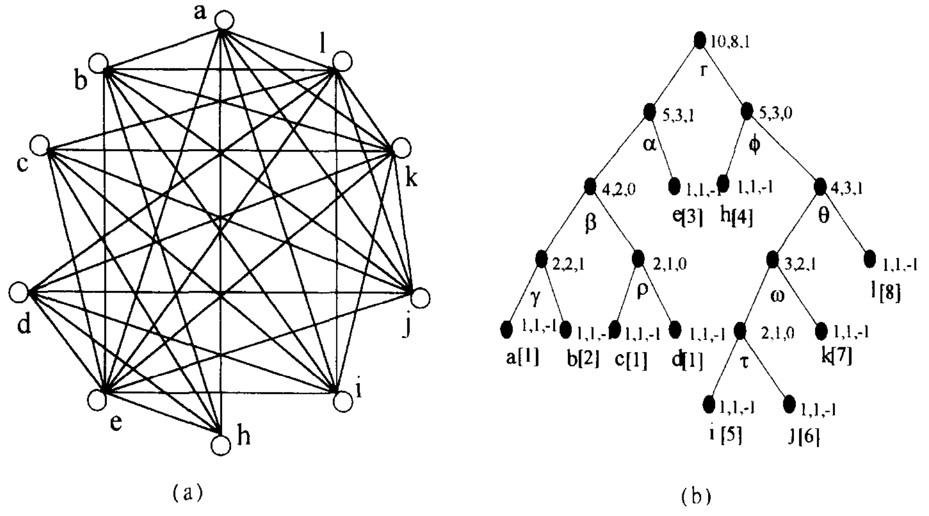


Fig. 2. (a) A cograph. (b) A parse tree of (a) and a ranking on it.

denotes the edge between vertices u and v. The *complement* of a graph $G=(V,E)$ is $\bar{G}=(V,\bar{E})$, where $\bar{E}=\{\{u,v\} \mid u,v \in V,\ \{u,v\} \notin E\}$.

Definition 1. A *cograph* $G=(V,E)$ is defined recursively as follows:
(1) If $|V|=1$, then G is a cograph;
(2) If $G_1, G_2, \ldots, G_k$ are cographs, then $G=G_1 \cup G_2 \cup \cdots \cup G_k$ is a cograph;
(3) If $G_1, G_2, \ldots, G_k$ are cographs, then $G=G_1 \times G_2 \times \cdots \times G_k$ is a cograph.
Rule (3) is equivalent to the following: (3)$'$ If G is a cograph, then so is its complement, $\bar{G}$. Fig. 2(a) shows a cograph.

We can associate a *parse tree* T_G for each cograph $G=(V,E)$. Each vertex of G corresponds to a unique leaf in T_G. Internal nodes of T_G have a label, 0 or 1. The cograph corresponding to a 0-labeled node v (abbreviated as (0)node) in T_G is obtained from the union of the cographs corresponding to the children of v in T_G. The cograph corresponding to a 1-labeled node v (abbreviated as (1)node) in T_G is obtained from the complete interconnection of the cographs corresponding to the children of v in T_G. Note that $\{u,v\} \in E$ if and only if the lowest common ancestor of u and v in T_G is a (1)node. And each internal node will have at least two children. By using a method similar to that in [10], each parse tree T_G can be transformed into an equivalent binary parse tree by rearranging the c ($\geqslant 3$) children of an internal node v as follows. First add $c-2$ internal nodes of the same label as v in a sequence of left children from v; then make the original c children of v the children of v and the newly added $c-2$ internal nodes. We will take a binary parse tree of a cograph as our input. Moreover, there is a normalized form among the parse tree representations for a cograph, called the *cotree* [5], which is unique up to a permutation of the children of the internal nodes.

The sequential algorithm in [18] is based on Lemma 1 below.

Lemma 1 (Scheffler [18]). *Let $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ be two cographs. The following formulas hold*:
(1) $r(G_1\cup G_2)=\max\{r(G_1),r(G_2)\}$,
(2) $r(G_1\times G_2)=\min\{r(G_1)+|V_2|,r(G_2)+|V_1|\}$.

Actually, Lemma 1 can be extended to the case that G_1 and G_2 are two graphs. Here we state the result explicitly and give a proof by analyzing the possible cases.

Theorem 2. *Let $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ be two graphs. The following formulas hold*:
(1) $r(G_1\cup G_2)=\max\{r(G_1),r(G_2)\}$,
(2) $r(G_1\times G_2)=\min\{r(G_1)+|V_2|,r(G_2)+|V_1|\}$.

Proof. Since formula (1) is easy to see, we omit the proof for it. We concentrate on formula (2).

(a) In $G_1\times G_2$, if we rank the vertices in G_1 with ranks $1,2,\ldots,r(G_1)$, we can rank the rest of the vertices in $G_1\times G_2$ (i.e., the vertices in G_2) with ranks $r(G_1)+1,\ldots,r(G_1)+|V_2|$, respectively. So, we have $r(G_1\times G_2)\leqslant r(G_1)+|V_2|$. Similarly, $r(G_1\times G_2)\leqslant r(G_2)+|V_1|$. Hence, $r(G_1\times G_2)\leqslant \min\{r(G_1)+|V_2|,r(G_2)+|V_1|\}$.

(b) Now, we shall prove that $r(G_1\times G_2)\geqslant \min\{r(G_1)+|V_2|,r(G_2)+|V_1|\}$.

(b.1) If $r(G_1)=|V_1|$ and $r(G_2)=|V_2|$, then $r(G_1\times G_2)=|V_1|+|V_2|\geqslant \min\{r(G_1)+|V_2|,r(G_2)+|V_1|\}$.

(b.2) Suppose $r(G_1)<|V_1|$. Then two vertices, say a and b, in V_1 will be ranked the same.

(b.2.1) If the vertices in V_2 all have different ranks in $G_1\times G_2$, then $r(G_1\times G_2)=r(G_1)+|V_2|$ since $\forall v\in G_1$, $\forall u\in G_2$, $\rho(v)\neq\rho(u)$. Hence, $r(G_1\times G_2)=r(G_1)+|V_2|\geqslant \min\{r(G_1)+|V_2|,r(G_2)+|V_1|\}$.

(b.2.2) Otherwise, two of the vertices in V_2, say c and d, will have the same rank in $G_1\times G_2$, which we now show to be impossible. Since $a-c-b$ is a path, $\rho(b)=\rho(a)<\rho(c)$. However, $c-a-d$ is also a path. We will have $\rho(d)=\rho(c)<\rho(a)$. This contradicts the previous result, $\rho(a)<\rho(c)$. So, $r(G_1\times G_2)\geqslant r(G_1)+|V_2|$ for case (b.2).

Similarly, $r(G_1\times G_2)\geqslant r(G_2)+|V_1|$. Combined with the result in case (b.1), we have $r(G_1\times G_2)\geqslant \min\{r(G_1)+|V_2|,r(G_2)+|V_1|\}$ for case (b). □

The above formulas hold for the pathwidth, *pw*, and treewidth, *tw*, of cographs [3, 18]. Pathwidth is defined as follows.

Definition 2. A path-decomposition of a graph $G=(V,E)$ is a pair $(\{X_i\mid i\in I\},I)$, with $\{X_i\mid i\in I\}$ a family of subsets, and there exists $r\in N$: $I=\{1,2,\ldots,r\}$ such that

(1) $\bigcup_{i \in I} X_i = V$;
(2) For all $(v,w) \in E$, there exists $i \in I$, $v \in X_i \wedge w \in X_i$;
(3) For all $v \in V$, there exists $b_v, e_v \in I$, such that for all $i \in I$, $v \in X_i \Leftrightarrow b_v \leqslant i \leqslant e_v$.
The pathwidth of $(\{X_i \mid i \in I\}, I)$ is $\max_{i \in I} |X_i|$. The pathwidth of G is the minimum pathwidth over all possible path-decompositions of G.

The definition of treewidth is similar to that of the pathwidth. It can be found in [3]. Because pathwidth is equal to treewidth in cographs [3], we have the following theorem [18].

Theorem 3 (Scheffler [18]). *For any cograph G, $r(G) = pw(G) + 1 = tw(G) + 1$.* □

Also, Scheffler provided a linear-time algorithm for the vertex ranking problem in cographs [18]. Theorem 2 leads to a method for ranking a parse tree. First, the ranking number of the cograph is computed in a bottom-up fashion. Then a ranking can be given according to how the ranking number is found. An example is shown in Fig. 2(b). The leaves in the parse tree are the vertices of the cograph. The number in the bracket to the right of a leaf shows its rank. Let T_v denote the subtree rooted at v in a parse tree and let G_v represent its corresponding cograph. There are three values associated with each node v in the parse tree. The first is the number of leaves in T_v. The second is the ranking number of G_v. The third is the label of v, with -1 denoting a leaf.

The ranking starts from the root. Since $r(G_r) = 8 = r(G_\alpha) + |G_\phi| = 3 + 5$, we rank the leaves in T_α with ranks 1 to 3 and rank the leaves in T_ϕ with ranks 4 to 8. In this case, $r(G_\alpha)$ is used in computing the ranking number of its parent, $r(G_r)$, and we say that α *contributes its ranking number*. Consider α, which contributes its ranking number. Since $r(G_\alpha) = 3 = r(G_\beta) + |G_e| = 2 + 1$, we assign the ranks 1 or 2 to the leaves in T_β and assign the rank 3 to e. Consider ϕ, which does not contribute its ranking number. Hence we assign each leaf in T_ϕ a different rank in $\{4,5,6,7,8\}$. We choose to assign them in postorder of the parse tree. Such a process continues until all the leaves are assigned a rank.

3. The parallel algorithm

Our parallel algorithm is an implementation of the method in the above section, which is composed of two phases. In Phase 1, we compute the ranking number of the cograph, G, with a binary parse tree as input. Then we find the rank of each node v in Phase 2. Our parallel algorithm uses the idea of tree contraction in [11, 23]. So we describe a basic operation, named *cutting*, first. For a node v in a binary tree, let $par(v)$ and $sib(v)$ be its parent and sibling, respectively. For a leaf v, we number it with the postorder numbering. For a node v, we save the number of leaves in T_v in LEAVES(v). We shall compute for every node v an expression, f_v, which is the ranking number of v in finding $r(G_v)$. At the beginning, we let $f_v = x$ for each internal node v because its value has not been determined yet. After cutting a leaf v, we record

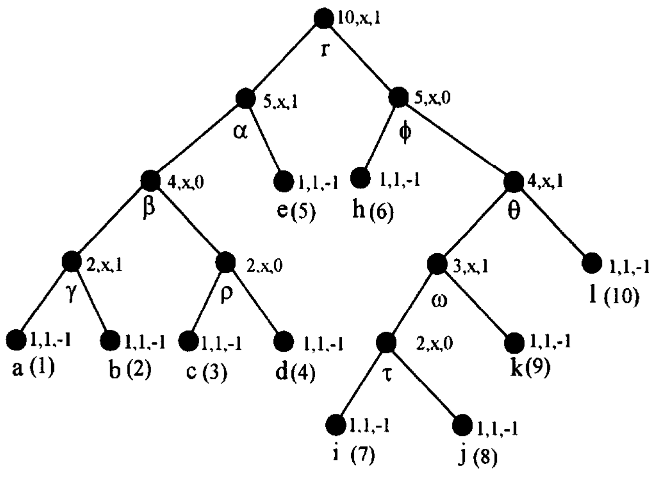


Fig. 3. A trace for applying Phase 1 of Algorithm PACOR to the cograph in Fig. 2. This figure shows the result after Steps 1 and 2. The vector beside each node v shows LEAVES(v), f_v, and the label of v with -1 denoting a leaf. The postorder number for each leaf is shown in the parenthesis to the right of the leaf.

the following five parameters on a stack of $par(v)$: (1) time stamp to perform this cutting, (2) left child of $par(v)$, (3) right child of $par(v)$, (4) the new $f_{par(v)}$, and (5) the new label of $par(v)$. We use the formulas in Theorem 2 to do our computing. And we keep the time stamp in a variable, *StepCounter*, to record the total number of cutting operations.

The *cutting* operation on a node v is given below:

(1) Cut v and $sib(v)$ by combining the expressions of v and $sib(v)$ into one and passing it to their parent as follows:

```
if $par(v)$ is a (1)node then
    $x_{par(v)} \leftarrow \min\{f_v + \text{LEAVES}(sib(v)), f_{sib(v)} + \text{LEAVES}(v)\}$
else/* $par(v)$ is a (0)node */
    $x_{par(v)} \leftarrow \max\{f_v, f_{sib(v)}\}$,
```

(2) Label of $par(v) \leftarrow$ label of $sib(v)$,

(3) if $sib(v)$ is a leaf, then postorder of $par(v) \leftarrow$ postorder of $sib(v)$, and

(4) Record the five parameters onto the stack of $par(v)$.

Our parallel algorithm, Algorithm PACOR, is given below. A trace for applying this algorithm to the cograph in Fig. 2 is shown in Figs. 3–7.

Algorithm PACOR./* Parallel Algorithm for Cograph's Optimal Ranking */

Input: A binary parse tree T of a cograph. And this tree is represented by using adjacency lists. Each node has 4 pointers: parent, left child, left sibling, and right sibling.

Output: An optimal ranking of the cograph. For each leaf v in the parse tree, RANK(v) is the result we want.

Begin

Phase 1:/* Compute the ranking number. */

```
Step 1. Use the Euler tour technique to do the following things:
            (1) For each leaf $v$ in $T$, number it with the postorder numbering.
            (2) For each node $v$ in the tree, LEAVES($v$) $\leftarrow$ the number of leaves in $T_v$.
Step 2. for each node $v$ in $T$, in parallel do
            if $v$ is a leaf then $f_v \leftarrow 1$
            else $f_v \leftarrow$ an indeterminate, $x$
            endif
            Push LEAVES($v$), $f_v$, and the label onto the stack.
            StepCounter $\leftarrow 1$;
        end for
Step 3. repeat until only one node remains
Step 3(a): for each odd-numbered leaf $v$, which is a left child,
           in parallel do cutting
           for each node $v$, in parallel do StepCounter $\leftarrow$ StepCounter $+ 1$;
Step 3(b): for each odd-numbered leaf $v$, which is a right child,
           in parallel do cutting
           for each node $v$, in parallel do StepCounter $\leftarrow$ StepCounter $+ 1$;
Step 3(c): Divide the numbering of the remaining leaves by 2
    end repeat
Phase 2:/* Assign a rank to each node. */
Step 4. Mark the root $r$.
Step 5. repeat Step 5(a) to Step 5(b) until StepCounter $= 0$
  Step 5(a): for each node $v$ that took an action at time stamp $=$ StepCounter
             during Phase 1, in parallel do
               (1) Pop one cutting from the stack of $v$ and recover the pair
                   of cut vertices, $u$ and $w$.
               (2) Compute the ranking numbers of $u$ and $w$, we can
                   use the information on their children if necessary.
               (3) if $v$ is unmarked then unmark $u$ and $w$
                   else /*$v$ is marked. */
                     if $u$ and $w$ do not contribute their ranking numbers then
                       unmark both,
                     else /*$u$ or $w$ contributes its ranking number. */
                       if $v$ is a (1)node then
                         mark the one which contributes its ranking number,
                       else /*$v$ is a (0)node. */
                         mark both
                       end if
                     end if
                   end if
  Step 5(b): StepCounter $\leftarrow$ StepCounter $- 1$;
             end for
  end repeat
```

Step 6. Use the Euler tour technique to compute the rank for each leaf.
The implementation is as follows:

```
for each advance edge, $\langle v,u\rangle$, (i.e., $v$ is the parent of $u$.) in parallel do
  if $v$ is marked and is a (1)node then
    if $u$ is marked then $w(\langle v,u\rangle) \leftarrow -r(G_v)$
    else /* $u$ is unmarked */
      $w(\langle v,u\rangle) \leftarrow r(G_{sib(u)})$
    end if
  else /* $v$ is unmarked or a (0)node */
    $w(\langle v,u\rangle) \leftarrow 0$
  end if
end for
for each retreat edge, $\langle u,v\rangle$, (i.e., $v$ is the parent of $u$.) in parallel do
  if $u$ is an unmarked leaf then $w(\langle u,v\rangle) \leftarrow 1$
  else
    $w(\langle u,v\rangle) \leftarrow 0$
  end if
end for
Use the parallel prefix on the list of directed edges.
```

/⋆ We restrict the Euler tour first to traverse the unmarked part then to traverse the marked part if it meets a marked (1)node. In the case of our example, there are 3 marked (1)nodes: r, α, and γ. Hence $\langle r,\phi\rangle$ is visited before $\langle r,\alpha\rangle$, $\langle \alpha,e\rangle$ is visited before $\langle \alpha,\beta\rangle$, and $\langle \gamma,b\rangle$ is visited before $\langle \gamma,a\rangle$. ⋆/

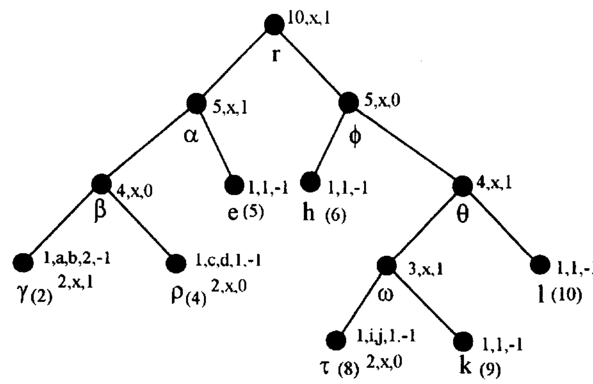
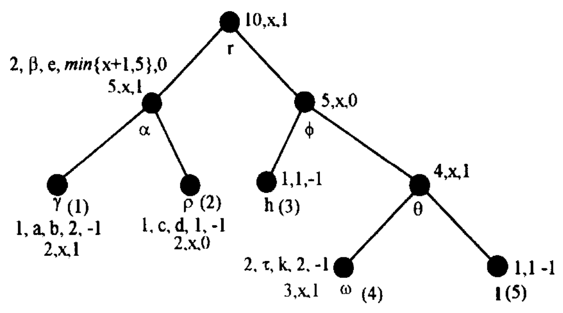
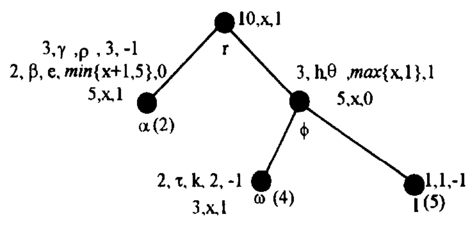


Fig. 4. A trace for applying Phase 1 of Algorithm PACOR to the cograph in Fig. 2. This figure is a trace for Step 3. The information we store is the StepCounter, left child, right child, ranking number, and label of v.

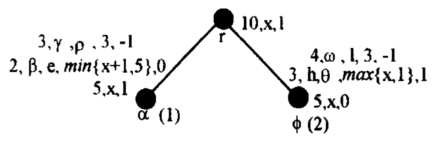
[2]: After Steps 3b) and 3c).



[3]: After Step 3a).



[4]: After Steps 3b) and 3c).



[5]: After Step 3a).

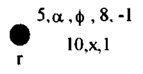


Fig. 4. Continued.

```
Step 7. for each leaf, $v$, in the parse tree in parallel do
            if $v$ is marked then RANK($v$) $\leftarrow$ 1
            else RANK($v$) $\leftarrow$ the value on the retreat edge from $v$
            end if
          end for
end Algorithm PACOR.
```

The correctness for Phase 1 follows the sequential algorithm [18] and Theorem 2. As to Phase 2, we first discuss Step 5. When the vertices are recovered, we can calculate

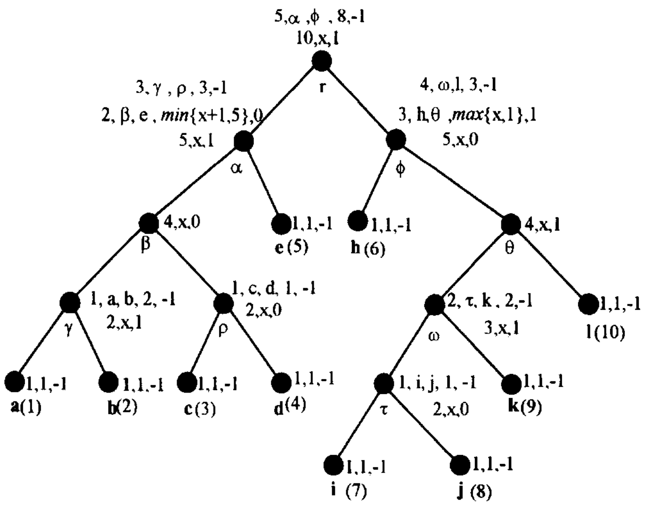


Fig. 5. The contents of the stack at each node after Step 3 of Algorithm PACOR is done. Near each node v, the lowest row contains LEAVES(v), f_v, and label of v, and the other rows contain the StepCounter, left child, right child, the ranking number at that step, and the lable.

their ranking numbers through the ranking numbers and leaf numbers of their children. The nodes contributing ranking numbers can be found during the execution. Hence, after Step 5, the marked nodes are the nodes which contribute their ranking numbers to finding the ranking number of the root.

In Step 6, if v is a marked (1)node, one of its children, say u, contributes $r(G_u)$ and the other child, say w, does not. We rank the leaves of T_w from $r(G_u)+1$ to $r(G_v)$, and the leaves of T_u by the numbers between 1 and $r(G_u)$. Our method first ranks the leaves of T_w, then goes on to rank the leaves of T_u. The purpose of the assignment, $\mathrm{w}(\langle v,w\rangle)\leftarrow r(G_u)$, is that we want to rank the leaves of T_w from $r(G_u)+1$ to $r(G_v)$. As for the assignment $\mathrm{w}(\langle v,u\rangle)\leftarrow -r(G_v)$, it just resets the value on the tour to be 0 and goes on to rank the leaves of T_u from 1 to $r(G_u)$. If v is a (0)node or is unmarked, we will set the weight to be zero and let the tour go on. Our tour goes to the unmarked part first when it meets a marked (1)node. For instance, a tour in our example can be $\langle \mathrm{r},\phi\rangle\rightarrow\langle\phi,\mathrm{h}\rangle\rightarrow\langle\mathrm{h},\phi\rangle\rightarrow\langle\phi,\theta\rangle\rightarrow\langle\theta,\omega\rangle\rightarrow\langle\omega,\tau\rangle\rightarrow\langle\tau,\mathrm{i}\rangle\rightarrow\langle\mathrm{i},\tau\rangle\rightarrow\langle\tau,\mathrm{j}\rangle\rightarrow\langle\mathrm{j},\tau\rangle\rightarrow\langle\tau,\omega\rangle$ $\rightarrow\langle\omega,\mathrm{k}\rangle\rightarrow\langle\mathrm{k},\omega\rangle\rightarrow\langle\omega,\theta\rangle\rightarrow\langle\theta,\mathrm{l}\rangle\rightarrow\langle\mathrm{l},\theta\rangle\rightarrow\langle\theta,\phi\rangle\rightarrow\langle\phi,\mathrm{r}\rangle\rightarrow\langle\mathrm{r},\alpha\rangle\rightarrow\langle\alpha,\mathrm{e}\rangle\rightarrow\langle\mathrm{e},\alpha\rangle\rightarrow$ $\langle\alpha,\beta\rangle\rightarrow\langle\beta,\gamma\rangle\rightarrow\langle\gamma,\mathrm{b}\rangle\rightarrow\langle\mathrm{b},\gamma\rangle\rightarrow\langle\gamma,\mathrm{a}\rangle\rightarrow\langle\mathrm{a},\gamma\rangle\rightarrow\langle\gamma,\beta\rangle\rightarrow\langle\beta,\rho\rangle\rightarrow\langle\rho,\mathrm{c}\rangle\rightarrow\langle\mathrm{c},\rho\rangle\rightarrow\langle\rho,\mathrm{d}\rangle$ $\rightarrow\langle\mathrm{d},\rho\rangle\rightarrow\langle\rho,\beta\rangle\rightarrow\langle\beta,\alpha\rangle\rightarrow\langle\alpha,\mathrm{r}\rangle$. With the correctly marked nodes in Step 5, our method to rank the vertices is correct. An example is shown in Fig. 7.

As to the time complexity, we refer to [11, 23]. Steps 1 and 6 can be done in $O(\log n)$ time with $n/\log n$ processors on an EREW PRAM model [4, 9]. It is easy to see that both Steps 2 and 7 can be done in $O(\log n)$ time with $n/\log n$ processors. Step 5 is just the reverse of Step 3 accompanied with some additional judgments that can be done in constant time for each iteration. We can concentrate on the time complexity of Step 3. Step 3 needs $O(\log n)$ iterations with the same argument in [11, 23]. All we need to verify is whether each iteration is of constant time. We know that the two operators, min and max, are associative and distributive. These properties are useful in the following lemma.

Lemma 4. *Each cutting operation in Step 3 can be done in constant time.*

Proof. We will show that the most general form of each vertex v is $f_v = \min\{\max\{x + c_1, c_2\}, c_3\}$. In initialization, $f_v = c$ if v is a leaf, and $f_v = x$ if v is an internal node. We need to show that the form remains after the operation, *cutting*, is done

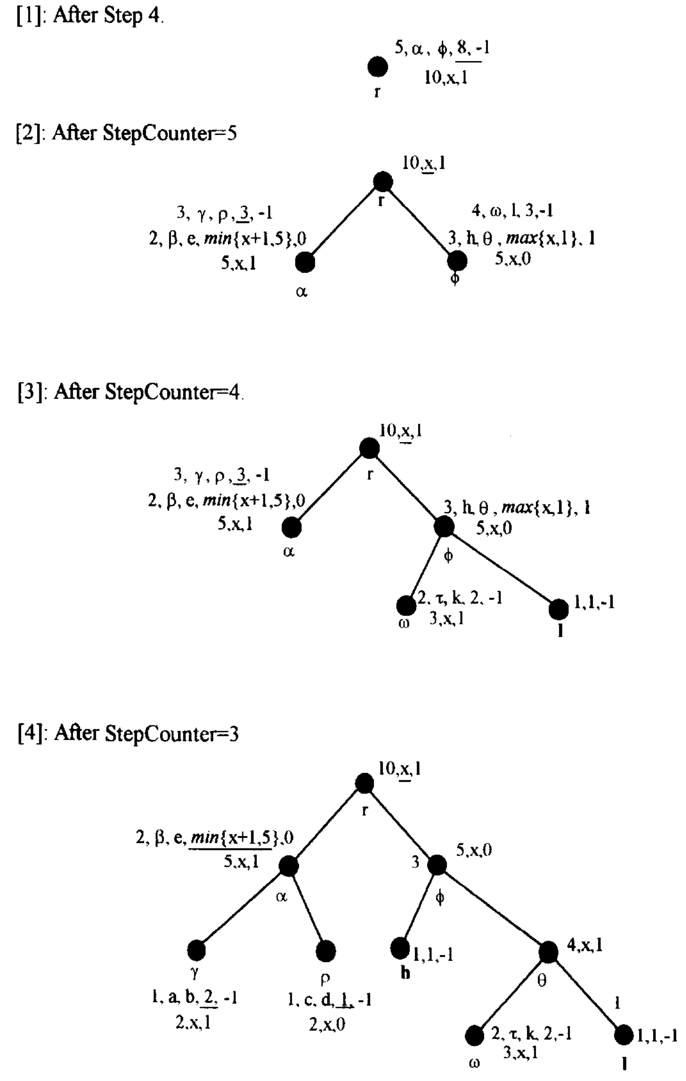
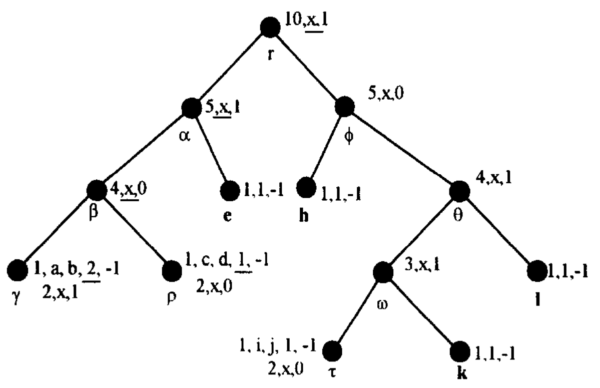


Fig. 6. A trace for applying Phase 2 of Algorithm PACOR to the cograph in Fig. 2. This figure is a trace for Steps 4 and 5. The underline denotes that the node in the tree contributes its ranking number.

[5]: After StepCounter=2



[6]: After StepCounter=1

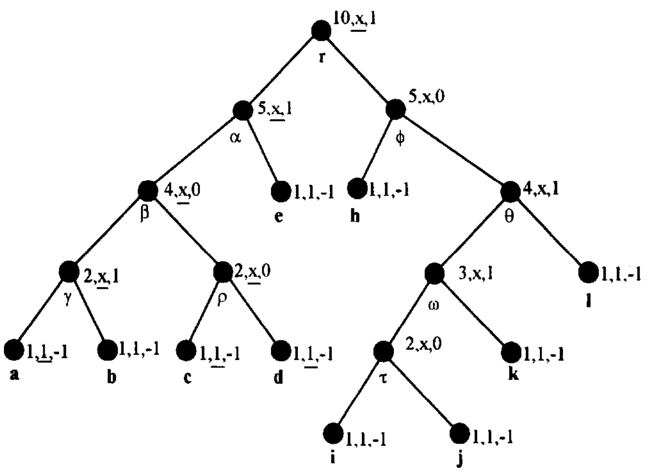


Fig. 6. Continued.

in constant time. Let us consider the case that the node to be cut is v which has a sibling u and a parent w. We will use the properties mentioned in the previous paragraph. Let the expressions in v, u, and w be $f_v = c_1$, $f_u = \min\{\max\{x+c_2, c_3\}, c_4\}$, and $f_w = \min\{\max\{x+c_5, c_6\}, c_7\}$, respectively.

(1) If w is a (1)node then after the cutting operation, the form in w will be

$$f_w = \min\{\max\{\min\{f_v + c_8, f_u + c_9\} + c_5, c_6\}, c_7\}$$

$/\star c_8 = |L(u)|$ and $c_9 = |L(v)| \star/$

$$= \min\{\max\{\min\{c_1 + c_8, f_u + c_9\} + c_5, c_6\}, c_7\}$$

$$= \min\{\max\{\min\{c_{10}, f_u + c_9\} + c_5, c_6\}, c_7\}$$

$$= \min\{\max\{\min\{c_{10} + c_5, f_u + c_9 + c_5\}, c_6\}, c_7\}$$

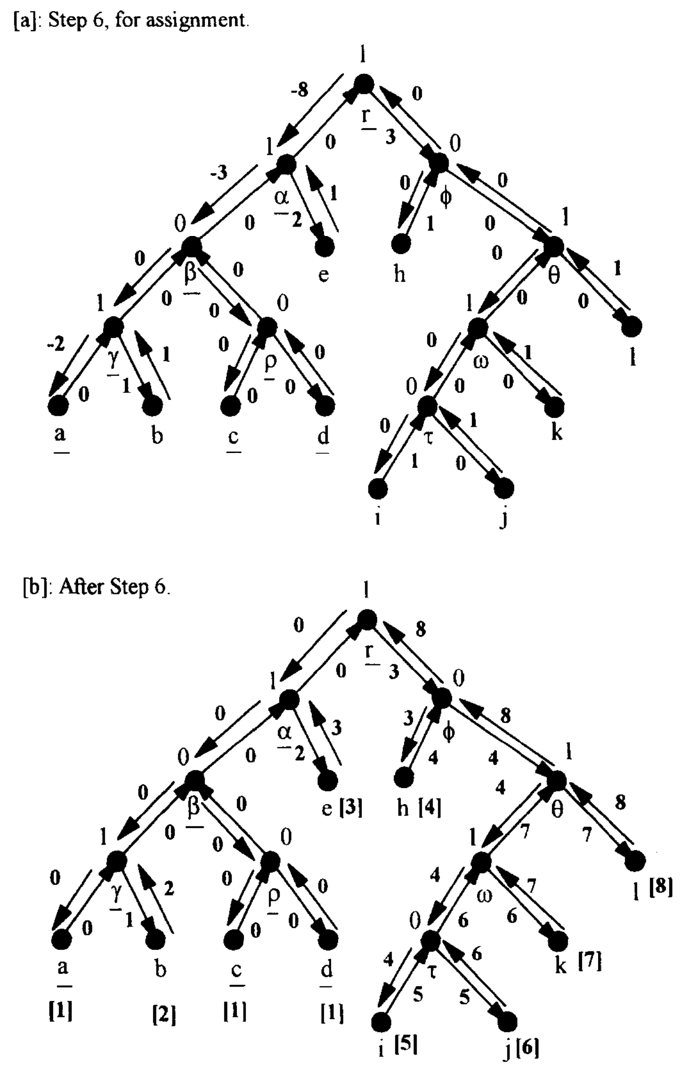


Fig. 7. A trace for applying Phase 2 of Algorithm PACOR to the cograph in Fig. 2, where [#] is the rank for each vertex. This figure shows the result after of Step 6.

$$
\begin{aligned}
&= \min\{\max\{\min\{c_{11}, f_u + c_{12}\}, c_6\}, c_7\}\\
&= \min\{\min\{\max\{c_{11}, c_6\}, \max\{f_u + c_{12}, c_6\}\}, c_7\}\\
&= \min\{\min\{c_{13}, \max\{f_u + c_{12}, c_6\}\}, c_7\}\\
&= \min\{\max\{f_u + c_{12}, c_6\}, \min\{c_{13}, c_7\}\}\\
&= \min\{\max\{f_u + c_{12}, c_6\}, c_{14}\}\\
&= \min\{\max\{\min\{\max\{x + c_2, c_3\}, c_4\} + c_{12}, c_6\}, c_{14}\}\\
&= \min\{\max\{\min\{\max\{x + c_2 + c_{12}, c_3 + c_{12}\}, c_4 + c_{12}\}, c_6\}, c_{14}\}
\end{aligned}
$$

$$
\begin{aligned}
&= \min\{\max\{\min\{\max\{x+c_{15},c_{16}\},c_{17}\},c_6\},c_{14}\}\\
&= \min\{\max\{\max\{\min\{x+c_{15},c_{17}\},\min\{c_{16},c_{17}\}\},c_6\},c_{14}\}\\
&= \min\{\max\{\max\{\min\{x+c_{15},c_{17}\},c_{18}\},c_6\},c_{14}\}\\
&= \min\{\max\{\min\{x+c_{15},c_{17}\},\max\{c_{18},c_6\}\},c_{14}\}\\
&= \min\{\max\{\min\{x+c_{15},c_{17}\},c_{19}\},c_{14}\}\\
&= \min\{\min\{\max\{x+c_{15},c_{19}\},\max\{c_{17},c_{19}\}\},c_{14}\}\\
&= \min\{\min\{\max\{x+c_{15},c_{19}\},c_{20}\},c_{14}\}\\
&= \min\{\max\{x+c_{15},c_{19}\},\min\{c_{20},c_{14}\}\}\\
&= \min\{\max\{x+c_{15},c_{19}\},c_{21}\}.
\end{aligned}
$$

(2) If w is a (0)node, with a derivation similar to (1), the form will remain after the *cutting* operation in constant time.

Hence we complete this lemma. □

With Lemma 4, see that each iteration needs only constant time to complete. Note the general form in the proof of Lemma 4. The general form also can be $\max\{\min\{x+c_1, c_2\},c_3\}$. But it is equivalent to the one we used above since $\max\{\min\{x+c_1,c_2\},c_3\} = \min\{\max\{x+c_1,c_3\},\max\{c_2,c_3\}\} = \min\{\max\{x+c_1,c_3\},c_4\}$. For consistency, we take $\min\{\max\{x+c_1,c_2\},c_3\}$ as our general form. From the above complexity analysis and the correctness shown in Theorem 2, we can get the following theorem.

Theorem 5. *An optimal ranking on a cograph represented by its binary parse tree can be found in* $O(\log n)$ *time with* $n/\log n$ *processors on the EREW PRAM model.* □

4. Concluding remarks

In this paper, we solve the optimal ranking problem on a cograph represented by a binary parse tree with a parallel algorithm which, furthermore, can be applied to the pathwidth and treewidth problems in cographs. There already are many polynomial-time sequential algorithms for this problem on several special graph classes. We feel that the exploration of parallel algorithms for other graph classes is still open and interesting.

References

[1] H.L. Bodlaender, J.S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, Zs. Tuza, Rankings of graphs, Lecture Notes in Computer Science, vol. 903, Springer, Berlin, 1996, pp. 292–304.

[2] H.L. Bodlaender, J.R. Gilbert, H. Hafsteinsson, T. Kloks, Approximating treewidth, pathwidth, and minimum elimination tree height, Proceedings of the 17th International Workshop on Graph-Theoretic Concepts in Computer Science, WG'91, in: Springer-Verlag, Lecture Notes in Computer Science, vol. 570, Springer, Berlin, 1992, pp. 1–12.

[3] H.L. Bodlaender, R.H. Möhring, The pathwidth and treewidth of cographs, SIAM J. Discrete Math. 6 (2) (1993) 181–188.

[4] R. Cole, U. Vishkin, Approximate parallel scheduling. Part I: the basic technique with applications to optimal parallel list ranking in logarithmic time, SIAM J. Comput. 17 (1988) 128–142.

[5] D.G. Corneil, H. Lerchs, L.S. Burlingham, Complement reducible graphs, Discrete Appl. Math. 3 (1981) 163–174.

[6] J.S. Deogun, T. Kloks, D. Kratsch, H. Müller, On vertex ranking for permutation and other graphs, Lecture Notes in Computer Science, vol. 775, Springer, Berlin, 1994, pp. 747–758.

[7] I.S. Duff, J.K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations, ACM Trans. Math. Software 9 (1983) 302–325.

[8] A.V. Iyer, H.D. Ratliff, G. Vijayan, Optimal node ranking of trees, Inform. Process. Lett. 28 (1988) 225–229.

[9] J. JáJá, An Introduction to Parallel Algorithms, Addison-Wesley, Reading, MA, 1992.

[10] D. Knuth, The Art of Computer Programming, vol. 1, Fundamental Algorithms, Addison-Wesley, Reading, MA, 1968.

[11] S.R. Kosaraju, A.L. Delcher, Optimal parallel evaluation of tree-structured computations by raking, Proceedings of the 3rd Aegean Workshop on Computing, AWOC88, 1988, pp. 101–110.

[12] C.E. Leiserson, Area-efficient graph layouts (for VLSI), Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science, 1980, pp. 270–281.

[13] Y. Liang, S.K. Dhall, S. Lakshmivarahan, Parallel algorithms for ranking of trees, Proceedings of the 2nd Annual IEEE Symposium on Parallel and Distributed Computing, 1990, pp. 26–31.

[14] J.W.H. Liu, The role of elimination trees in sparse factorization, SIAM J. Matrix Anal. Appl. 11 (1990) 134–172.

[15] J. Nevins, D. Whitney (Eds.), Concurrent Design of Products and Processes, McGraw-Hill, New York, 1989.

[16] A. Pothen, The complexity of optimal elimination trees, Technical Report CS-88-13, Pennsylvania State University, USA, 1988.

[17] A.A. Schäffer, Optimal node ranking of trees in linear time, Inform. Process. Lett. 33 (1989/90) 91–96.

[18] P. Scheffler, Node ranking and searching on cographs (Abstract), in: U. Faigle and C. Hoede (Eds.), 3rd Twente Workshop on Graph and Combinatorial Optimization, 1993.

[19] A. Sen, H. Deng, S. Guha, On a graph partition problem with application to VLSI layout, Inform. Process. Lett. 43 (1992) 87–94.

[20] P. de la Torre, R. Greenlaw, Super critical tree numbering and optimal tree ranking are in NC, Proceedings of the 3rd Annual IEEE Symposium on Parallel and Distributed Computing, 1991, pp. 767–773.

[21] P. de la Torre, R. Greenlaw, T.M. Przytycka, Optimal tree ranking is in NC, Parallel Process. Lett. 2 (1992) 31–41.

[22] P. de la Torre, R. Greenlaw, A.A. Schäffer, Optimal ranking of trees in polynomial time, Proceedings of the 4th ACM Symposium on Discrete Algorithms, Austin, Texas, 1993, pp. 138–144.

[23] M.S. Yu, L.Y. Tseng, J.H. Lin, Optimal parallel algorithms for some problems on trees, Proceedings of the International Conference on Parallel Processing, 1992, pp. III160–III163.