INFORMATION SCIENCES 77, 179–193 (1994)

A Theorem on Permutation Graphs with Applications

CHANG-WU YU

and

GEN-HUEY CHEN*

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

Communicated by Tosiyasu L. Kunii

ABSTRACT

In this paper, a theorem that characterizes a relation between cographs and permutation graphs is introduced. This theorem has an application to the recognition of a cograph. By the aid of this theorem, sequential and parallel algorithms to determine if a given permutation graph is a cograph can be derived.

1. INTRODUCTION

An undirected graph G with vertices $\{v_1, v_2, \ldots, v_n\}$ is called a *permutation graph* [10] if there exists a permutation π on $N = \{1, 2, \ldots, n\}$ such that for all $i, j \in N$,

$$(i-j)(\pi^{-1}(i) - \pi^{-1}(j)) < 0$$

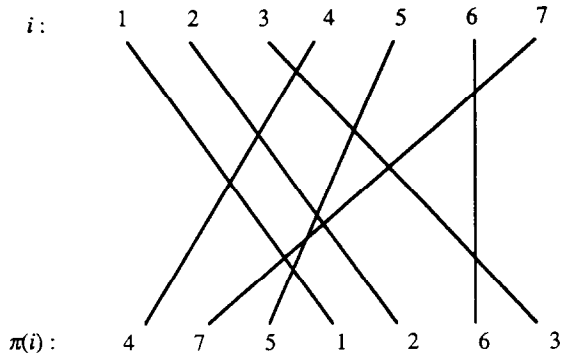
if and only if v_i and v_j are joined by an edge in G. Pictorially, draw the vertices $v_1, v_2, \ldots, v_n$ in order on a line, and $v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)}$ on a parallel line such that for each $i \in N$, v_i is directly above $v_{\pi(i)}$. Next, for each $i \in N$, draw a line segment from v_i on the upper line to v_i on the lower line. Then, there is an edge (v_i, v_j) in G if and only if the line segment for v_i intersects the line segment for v_j. As an illustrative

*Corresponding author.

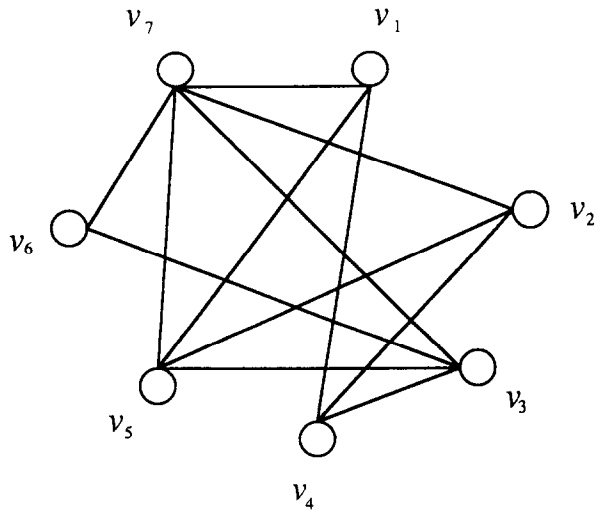
655 Avenue of the Americas, New York, NY 10010 0020-0255/94/$7.00

example, Figure 1 shows a permutation $\pi=(4, 7, 5, 1, 2, 6, 3)$ and its corresponding permutation graph.

Much research [3], [4], [8], [10], [18], [21]–[26] has been devoted to the study of permutation graphs. In [18], Pnueli, Lempel, and Even proposed an $O(n^3)$ time algorithm that recognizes a permutation graph of n vertices by applying the transitively orientable graph test. Later, Spinrad [21]



(a)



(b)

Fig. 1. An example. (a) A permutation π. (b) The corresponding permutation graph.

improved their result by presenting an $O(n^2)$ time recognition algorithm. Besides, Spinrad also gave an algorithm that determines if two permutation graphs are isomorphic in the same time complexity. In [22], Spinrad, Brandstadt, and Stewart showed that a bipartite permutation graph can be recognized in linear time by some algorithmic properties. In [12] and [15], the problem of recognizing a permutation graph was shown to be in the NC class. In [23], Supowit solved the coloring problem, the maximum clique problem, the cliques cover problem, and the maximum independent set problem, all in $O(n \log n)$ time. In [8], using dynamic programming, Farber and Keil solved the weighted domination problem and the weighted independent domination problem in $O(n^3)$ time. In [4], Brandstadt and Kratsch presented an $O(n^2)$ time algorithm for the weighted independent domination problem. In [3], Atallah, Manacher, and Urrutia solved the independent domination set problem in $O(n \log^2 n)$ time. In [24], Tsai and Hsu solved the domination problem and the weighted domination problem in $O(n \log\log n)$ time and $O(n^2 \log^2 n)$ time, respectively. In [25] and [26], Yu and Chen presented various NC algorithms for permutation graphs.

Cographs (or *complement reducible graphs*) [6] are defined as the class of graphs formed from a single vertex under the closure of the operations of union and complement. More detailed description about cographs can be found in [6]. Cographs were independently discovered under various names and were shown by Corneil, Lerchs, and Burlingham [6] to have the following two remarkable properties. First, they are P_4-restricted graphs [6]. In a P_4-restricted graph, there exists no path of length 3 (four nodes in this path) such that no edge exists between two nodes that are not adjacent in this path. Second, a cograph has a unique tree representation called *cotree*. The leaves of a cotree represent the vertices of its corresponding cograph, and its internal nodes are labeled alternately with 0 and 1 along every path starting from the root. The root is labeled with 1 if the cograph is connected, and 0 otherwise. Two vertices x and y in a cograph are adjacent if and only if the lowest common ancestor of their corresponding nodes in the cotree is a 1-node. An example of a cograph and its cotree is shown in Figure 2. The cotree forms the basis of many fast polynomial time algorithms for cographs. In [13], it was shown that a cograph must be a permutation graph, but the reverse is not always true. In this paper, like [7], we consider the cotree unordered. If the cotree is considered ordered, then a cograph may have many cotrees.

In this paper, we first introduce a theorem about permutation graphs, in which three equivalent statements are shown. This theorem characterizes a relation between cographs and permutation graphs. More concretely, it states that a permutation graph is a cograph if and only if either of two predefined conditions is satisfied. Then, we show an application of this

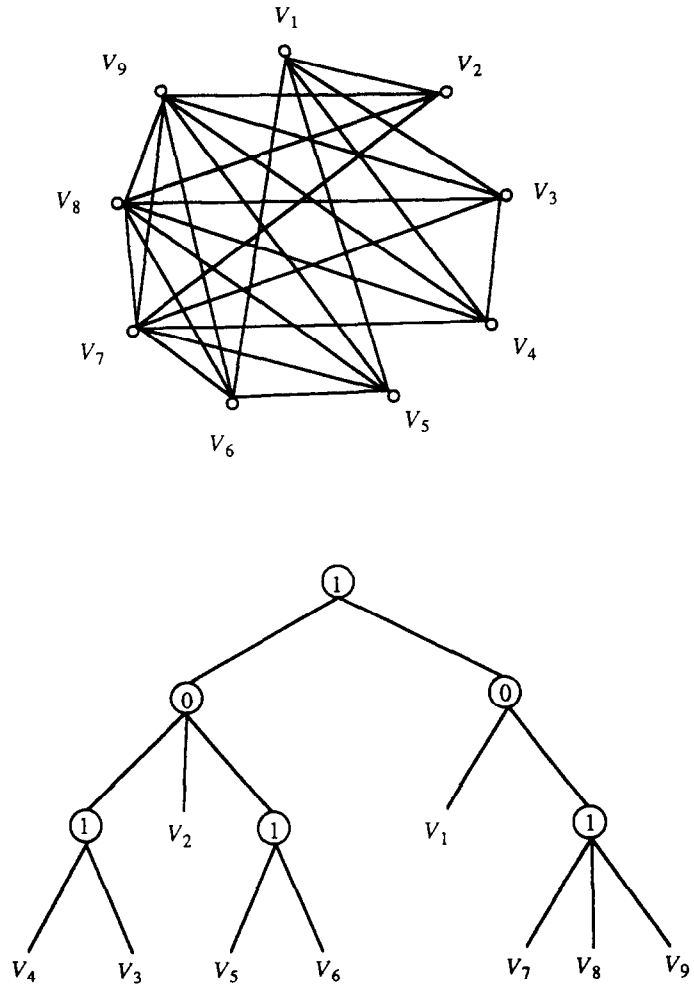


Fig. 2. A cograph and its cotree.

theorem to the recognition of a cograph. By the aid of this theorem, sequential and parallel algorithms to determine if a given permutation graph is a cograph can be derived.

The rest of this paper is organized as follows. In the next section, we introduce some necessary definitions. In Section 3, we give a theorem on permutation graphs. Then, based on this theorem, we present sequential and parallel algorithms for recognizing a cograph in Section 4. In Section 5, we conclude this paper with some final remarks.

2. DEFINITIONS

Given a set S of plane points, $a=(x_1, y_1)\in S$ is said to be *dominated* by $b=(x_2, y_2)\in S$ (or B *dominates* a) if $x_1<x_2$ and $y_1<y_2$. Further, a is said to be *directly dominated* by b (or b *directly dominates* a) if a is dominated by b and there exists no other point c in S such that a is dominated by c and c is dominated by b. The *minimal set* of S is defined as the set of points in S that dominate no point in S.

Let U and V be two sequences of integers. We define $U\cdot V$ as the concatenation of U and V, and $U<V$ if each integer in U is smaller than each integer in V. For example, if $U=(1,3,2,4)$ and $V=(5,6,9,8,7)$, then $U<V$ and $U\cdot V=(1,3,2,4,5,6,9,8,7)$. In the following, we define parenthesizable sequences.

DEFINITION 1. A sequence is *parenthesizable* if it can be constructed according to the following two rules.

(1) An integer is parenthesizable.

(2) If $V_1, V_2, \ldots, V_k$ are parenthesizable sequences satisfying $V_1<V_2<\cdots<V_k$ or $V_1>V_2>\cdots>V_k$, and $C=V_1\cdot V_2\cdot\cdots\cdot V_k$ contains consecutive integers, then C is also parenthesizable.

For example, by rule 1, each integer in the sequence $\pi=(6,5,7,9,8,1,4,3,2)$ is parenthesizable. Then, by rule 2, (6,5), (9,8), and (4,3,2) are all parenthesizable. Again, by rule 2, (6,5,7,9,8) and (1,4,3,2) are parenthesizable. Finally, we have (6,5,7,9,8,1,4,3,2) parenthesizable. Conveniently, we can express the above construction as ((((6)(5))(7)((9)(8)))((1)((4)(3)(2)))). Note that it is possible that a parenthesizable sequence may be parenthesized in more than one way. For example, ((((6)(5))(7)((9)(8)))((1)((4)(3)(2)))) and ((((6)(5))(7)((9)(8))))((1)(((4)(3))(2)))) are another two parenthesizations of π. In the rest of this paper, we use (π) to denote the parenthesization of π with the least number of pairs of parentheses, which is unique. In our example, (π) denotes ((((6)(5))(7)((9)(8)))((1)((4)(3)(2)))). Note that no two pairs of parentheses intersect in a parenthesization; that is, for only two pairs of parentheses, one appears inside or outside the other.

The *size* of a pair of parentheses is defined as the number of integers it contains. For example, the size of the pair of parentheses that consists of the second left parenthesis and the eighth right parenthesis from the left in ((((6)(5))(7)((9)(8)))((1)((4)(3)(2)))) is 5.

Given a graph $G=(V,E)$, where V denotes the vertex set and E denotes the edge set, the *induced subgraph* of G by $V'\subseteq V$ is the subgraph of G whose vertex set is V' and whose edge set contains those edges in E having both end vertices in V'. We denote the subgraph by $G_{V'}$.

3. A THEOREM

In this section, we introduce a theorem that characterizes a relation between cographs and permutation graphs. It will be seen in the next section that this theorem has an application to the recognition of a cograph.

LEMMA 1. *Let π be a permutation on $N=\{1,2,\ldots,n\}$. If π is parenthesizable and there exist $a,b,c\in N$ satisfying $a<b<c$ and $\pi^{-1}(c)<\pi^{-1}(a)<\pi^{-1}(b)$, then there exists a pair of parentheses in (π) that contains a,b, but does not contain c.*

Proof. Because $\pi^{-1}(c)<\pi^{-1}(a)<\pi^{-1}(b)$, we assume $\pi=(\ldots,c,\ldots,a,\ldots,b,\ldots)$. There are three possible ways to include c,a,b in a pair of parentheses. The first way is to include c,a,b directly by a pair of parentheses, i.e.,$\ldots(\ldots,c,\ldots,a,\ldots,b,\ldots)\ldots$, which is impossible by rule 2. The second way is to include c,a first by a pair of parentheses, and then include b by another pair of parentheses, i.e., $\ldots(\ldots(\ldots,c,\ldots,a,\ldots)\ldots b,\ldots)\ldots$, which is again impossible by rule 2. Therefore, the only one way to include c,a,b in a pair of parentheses is like the form $\ldots(\ldots c\ldots(\ldots,a,\ldots,b,\ldots)\ldots)\ldots$. This completes the proof. □

The following lemmas can be proved similarly.

LEMMA 2. *Let π be a permutation on $N=\{1,2,\ldots,n\}$. If π is parenthesizable and there exist $a,b,c\in N$ satisfying $a<b<c$ and $\pi^{-1}(b)<\pi^{-1}(a)<\pi^{-1}(c)$, then there exists a pair of parentheses in (π) that contains b,a, but does not contain c.*

LEMMA 3. *Let π be a permutation on $N=\{1,2,\ldots,n\}$. If π is parenthesizable and there exist $a,b,c\in N$ satisfying $a<b<c$ and $\pi^{-1}(a)<\pi^{-1}(c)<\pi^{-1}(b)$, then there exists a pair of parentheses in (π) that contains c,b, but does not contain a.*

LEMMA 4. *Let π be a permutation on $N=\{1,2,\ldots,n\}$. If π is parenthesizable and there exist $a,b,c\in N$ satisfying $a<b<c$ and $\pi^{-1}(b)<\pi^{-1}(c)<\pi^{-1}(a)$, then there exists a pair of parentheses in (π) that contains b,c, but does not contain a.*

LEMMA 5 [6]. *G is a cograph if and only if G does not contain a P_4 as an induced subgraph, where P_4 denotes a path of length 3 (P_4 contains four vertices and three edges).*

Now we show the main result of this section.

THEOREM 1. *Let G be a permutation graph with $\pi=(\pi(1),\pi(2),\ldots,\pi(n))$ (a permutation of $\{1,2,\ldots,n\}$). The following statements are equivalent.*

(1) *G is a cograph.*

(2) π *is a parenthesizable sequence.*

(3) *There do not exist* $a, b, c, d \in \{1, 2, \ldots, n\}$ *satisfying* $a < b < c < d$ *and*

$$(\pi^{-1}(c) < \pi^{-1}(a) < \pi^{-1}(d) < \pi^{-1}(b) \text{ or}$$
$$\pi^{-1}(b) < \pi^{-1}(d) < \pi^{-1}(a) < \pi^{-1}(c)).$$

Proof. We prove this theorem by showing that (1) implies (2), (2) implies (3), and (3) implies (1).

(1) $\rightarrow$ (2): Let T denote the cotree of G, and T_r the subtree of T whose root is r. First, we show that all permutations π that can represent G can be obtained from T. For the convenience of discussion, we suppose that the root node of T has two children $s1$, $s2$, and $q_{s1} = k$, $q_{s2} = n - k$, where q_{si} is the number of leaf nodes in T_{si}, $i = 1, 2$. The tree T may be of either of two shapes shown in Figure 3. The extension to three or more children is rather straightforward. If the root node of T is a 1-node, then every leaf node in T_{s1} is adjacent to every leaf node in T_{s2}. Thus, the corresponding permutation π of G is in either of the two forms: $\pi = (\pi(1), \pi(2), \ldots, \pi(k), \pi(k+1), \pi(k+2), \ldots, \pi(n))$, where $(\pi(1), \pi(2), \ldots, \pi(k))$ is a permutation of $\{n-k+1, k+2, \ldots, n\}$ and $\{\pi(k+1), \pi(k+2), \ldots, \pi(n)\}$ is a permutation of $\{1, 2, \ldots, n-k\}$, and $\pi = (\pi(1), \pi(2), \ldots, \pi(n-k), \pi(n-k+1), \pi(k+2), \ldots, \pi(n))$, where $(\pi(1), \pi(2), \ldots, \pi(n-k))$ is a permutation of $\{k+1, k+2, \ldots, n\}$ and $\{\pi(n-k+1), \pi(k+2), \ldots, \pi(n)\}$ is a permutation of $\{1, 2, \ldots, k\}$.

Similarly, if the root node of T is a 0-node, then every leaf node in T_{s1} is not adjacent to any leaf node in T_{s2}. Thus, the corresponding permutation π of G is in either of the two forms: $\pi = (\pi(1), \pi(2), \ldots, \pi(k), \pi(k+1), \pi(k+2), \ldots, \pi(n))$, where $(\pi(1), \pi(2), \ldots, \pi(k))$ is a permutation of $\{1, 2, \ldots, k\}$ and $\{\pi(k+1), \pi(k+2), \ldots, \pi(n)\}$ is a permutation of $\{k+1, k+2, \ldots, n\}$, and $\pi = (\pi(1), \pi(2), \ldots, \pi(n-k), \pi(n-k+1), \pi(k+2), \ldots, \pi(n))$, where $(\pi(1), \pi(2), \ldots, \pi(n-k))$ is a permutation of $\{1, 2, \ldots, n-k\}$ and $\{\pi(n-k+1), \pi(n-k+2), \ldots, \pi(n)\}$ is a permutation of $\{n-k+1, n-k+2, \ldots, n\}$.

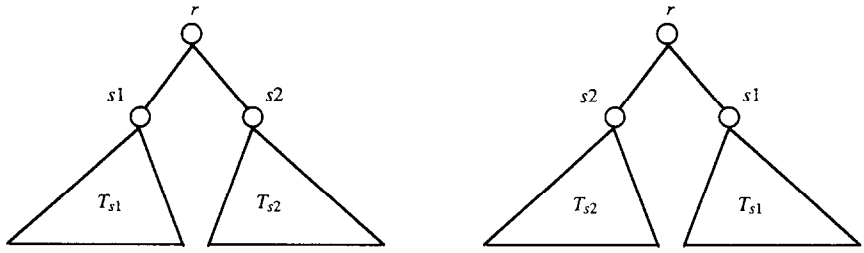


Fig. 3. Two possible shapes of T.

If we repeat the above analysis for nodes $s1, s2$ and all other internal nodes of T, we can generate all corresponding permutations π of G. More concretely, we have the following generation method for each possible shape of T.

What we have to do is to label leaf nodes of T with integers $1, 2, \ldots, n$. First, we associate each internal node r of T with an interval, which represents the set of integers that are used to label leaf nodes of T_r. The root node of T is assigned with the interval $[1, n]$, and the other nodes are assigned with intervals as follows. Let $s1, s2, \ldots, sk$ denote the children from left to right of r. If r is a 1-node and has been assigned with an interval $[b, c]$, then $s1, s2, \ldots, sk$ are assigned with intervals $[c - q_{s1} + 1, c], [c - (q_{s1} + q_{s2}) + 1, c - q_{s1}], \ldots, [c - \sum_{1 \leqslant i \leqslant k} q_{si} + 1, c - \sum_{1 \leqslant i \leqslant k-1} q_{si}]$, respectively, where q_{si} is the number of leaves in T_{si}. On the other hand, if r is a 0-node, then $s1, s2, \ldots, sk$ are assigned with intervals $[b, b + q_{s1} - 1], [b + q_{s1}, b + (q_{s1} + q_{s2}) - 1], \ldots, [b + \sum_{1 \leqslant i \leqslant k-1} q_{si}, b + \sum_{1 \leqslant i \leqslant k} q_{si} - 1]$, respectively. In this way, each leaf node will be assigned with an interval that contains only one integer. An example is shown in Figure 4.

Let $v_1, v_2, \ldots, v_n$ denote the leaf nodes from left to right of T, and $l(v_i)$ the integer assigned to v_i. Now, $\pi_0 = (l(v_1), l(v_2), \ldots, l(v_n))$ is the permutation obtained from T by the above labeling procedure.

As a result of the labeling procedure, we claim that for $i < j$, the lowest common ancestor of v_i and v_j in T is a 1-node if and only if $l(v_i) > l(v_j)$. The proof is easy and therefore left as an exercise. By the claim and the

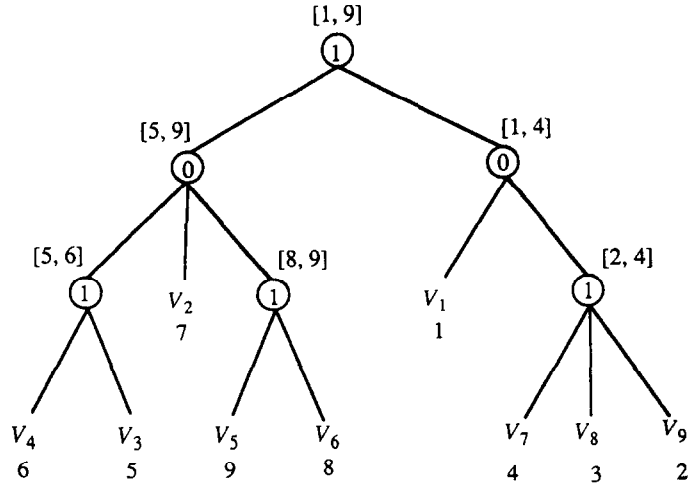


Fig. 4. A labeled cotree.

fact that G is both a permutation graph and a cograph, it is easy to see that nodes v_i and v_j in G are adjacent if and only if $(l(v_i) - l(v_j))(\pi_0^{-1}(l(v_i)) - \pi_0^{-1}(l(v_j))) = (l(v_i) - l(v_j))(i-j) < 0$. So, π_0 can represent G exactly. Besides, π_0 is parenthesizable as a result of the labeling procedure.

Because all corresponding permutations of G can be generated by executing the labeling procedure for all possible shapes of T, π, like π_0, is parenthesizable.

(2) $\rightarrow$ (3): Suppose there exist $a, b, c, d \in \{1, 2, \ldots, n\}$ satisfying $a < b < c < d$ and $\pi^{-1}(c) < \pi^{-1}(a) < \pi^{-1}(d) < \pi^{-1}(b)$. Because $b < c < d$ and $\pi^{-1}(c) < \pi^{-1}(d) < \pi^{-1}(b)$, by Lemma 4 there is a pair of parentheses in (π) that contains c, d, but does not contain b. On the other hand, because $a < b < c$ and $\pi^{-1}(c) < \pi^{-1}(a) < \pi^{-1}(b)$, by Lemma 1 there is a pair of parentheses in (π) that contains a, b, but does not contain c. So, these two pairs of parentheses intersect, which is a contradiction.

Similarly, by Lemmas 2 and 3, we can prove that there do not exist $a, b, c, d \in \{1, 2, \ldots, n\}$ satisfying $a < b < c < d$ and $\pi^{-1}(b) < \pi^{-1}(d) < \pi^{-1}(a) < \pi^{-1}(c)$.

(3) $\rightarrow$ (1): Suppose G is not a cograph. Then, by Lemma 5, G contains a P_4 as an induced subgraph. Let W denote the set of four vertices in P_4 (that is, $G_W = P_4$), and a, b, c, d denote the four integers in π that represent the four vertices in P_4. Without loss of generality, we assume $a < b < c < d$. Then, $\pi^{-1}(a)$ ($\pi^{-1}(d)$) is not the minimal (maximal) value of $\{\pi^{-1}(a), \pi^{-1}(b), \pi^{-1}(c), \pi^{-1}(d)\}$, because otherwise G_W is not connected. On the other hand, $\pi^{-1}(a)$ ($\pi^{-1}(d)$) is not the maximal (minimal) value of $\{\pi^{-1}(a), \pi^{-1}(b), \pi^{-1}(c), \pi^{-1}(d)\}$, because otherwise the maximal degree of G_W is equal to 3. Hence, the maximal value and the minimal value of $\{\pi^{-1}(a), \pi^{-1}(b), \pi^{-1}(c), \pi^{-1}(d)\}$ are in $\{\pi^{-1}(b), \pi^{-1}(c)\}$.

If $\pi^{-1}(c) < \pi^{-1}(b)$, then $\pi^{-1}(a) < \pi^{-1}(d)$, because otherwise $\pi^{-1}(c) < \pi^{-1}(d) < \pi^{-1}(a) < \pi^{-1}(b)$ and G_W forms a cycle. If $\pi^{-1}(b) < \pi^{-1}(c)$, then $\pi^{-1}(d) < \pi^{-1}(a)$, because otherwise $\pi^{-1}(b) < \pi^{-1}(a) < \pi^{-1}(d) < \pi^{-1}(c)$ and G_W is not connected.

Therefore, we have $\pi^{-1}(c) < \pi^{-1}(a) < \pi^{-1}(d) < \pi^{-1}(b)$ or $\pi^{-1}(b) < \pi^{-1}(d) < \pi^{-1}(a) < \pi^{-1}(c)$. This completes the proof. □

4. APPLICATIONS

In this section, we show an application of Theorem 1 to the problem of recognizing a cograph. First, we survey previous work on this problem. Throughout this section, we let n and m denote the numbers of vertices and edges, respectively.

The fastest sequential algorithm, which was proposed by Corneil, Perl, and Stewart [7], takes $O(n+m)$ time. As for parallel algorithms, various results have been proposed on different computation models. In [1], Adhar and Peng presented an $O(\log^2 n)$ time algorithm on CRCW PRAM (concurrent read, concurrent write parallel random access machine), which requires $O(nm)$ processors. In [14], Kirkpatrick and Przytycka presented an $O(\log^2 n)$ time algorithm on CREW (concurrent read, exclusive write) PRAM, using $O(n^3/\log^2 n)$ processors. In [17] and [20], an $O(\log n)$ time algorithm, which uses $O(n^3)$ processors on CRCW PRAM, is obtained by Novick and Shyu, independently. In [16], Lin and Olariu gave an $O(\log n)$ time algorithm, which uses $O((n^2+nm)/\log n)$ processors on EREW (exclusive read, exclusive write) PRAM. In [11], He presented an $O(\log^2 n)$ time algorithm, which uses $O(n+m)$ processors on CRCW PRAM.

All the algorithms above have arbitrary graphs as their input. In the remainder of this section, as an application of Theorem 1, we show that time complexity and/or processor complexity can be reduced, provided the input graph is restricted to a permutation graph.

First, we present a sequential algorithm to recognize a cograph, assuming the input is a permutation graph. It is known by Theorem 1 that given a permutation graph G with its corresponding permutation π, G is a cograph if and only if $\pi=(\pi(1),\pi(2),\ldots,\pi(n))$ is a parenthesizable sequence. So, the problem of recognizing a cograph is transformed into the problem of recognizing a parenthesizable sequence π. In the following, an algorithm for the latter is presented.

In the algorithm, a stack is used to store parenthesizable subsequences that are identified during the execution of the algorithm. The algorithm scans the input π from left to right and tries to merge each $\pi(i)$, $i=1,2,\ldots,n$, with previously found parenthesizable subsequences (stored in the stack). The resulting subsequence is parenthesizable and is again pushed into the stack. After finishing the scanning of π, we can easily determine whether π is parenthesizable or not. If only one subsequence remains in the stack, then π is parenthesizable. Otherwise, π is not parenthesizable.

ALGORITHM 1.

Step 1. Scan the input $\pi=(\pi(1),\pi(2),\ldots,\pi(n))$ from left to right and perform the following for each $\pi(i)$, $i=1,2,\ldots,n$.

1.1. Push $\pi(i)$ into the stack.

1.2. If the top two subsequences in the stack can be merged into a longer parenthesizable subsequence (by rule 2 of Definition 1), then

pop the two subsequences from the stack,

merge them into a new parenthesizable subsequence, push the resulting subsequence into the stack, and go to Step 1.2.

Step 2. If there is only one parenthesizable subsequence in the stack, then π is a parenthesizable sequence. Otherwise π is not a parenthesizable sequence.

The time complexity of Algorithm 1 is dominated by Step 1.2, which is executed n times. Each of the pop, merge, and push operations takes constant time, and they occur whenever the top two subsequences in the stack satisfy rule 2 of Definition 1. Because each merge operation will result in a longer subsequence, the merge operation is executed at most n times. So, the time complexity of Algorithm 1 is proportional to n. Also, the space complexity of Algorithm 1 is proportional to n.

It is easy to construct (π) by modifying Algorithm 1 at two places. First, before a new parenthesizable subsequence is pushed into the stack, we add a pair of parentheses to enclose the subsequence. Second, while merging two smaller parenthesizable subsequences into a longer one, the two pairs of parentheses that enclose the two smaller subsequences may become redundant, and therefore need to be eliminated. The redundant pairs of parentheses can be identified by rule 2 of Definition 1.

The cotree T can be constructed from (π) as follows. We simply let each integer in (π) be a leaf node of T and all the pairs of parentheses that contain two or more integers be the internal nodes of T. The parent of a pair of parentheses is the pair of parentheses that encloses it and has the smallest size. It is not difficult to check that the time complexity and space complexity of constructing the cotree are still $O(n)$.

Next, we present a parallel algorithm, as a consequence of statement (3) of Theorem 1. The following algorithm determines if there exist $a, b, c, d \in \{1, 2, \ldots, n\}$ satisfying $a < b < c < d$ and $(\pi^{-1}(c) < \pi^{-1}(a) < \pi^{-1}(d) < \pi^{-1}(b)$ or $\pi^{-1}(b) < \pi^{-1}(d) < \pi^{-1}(a) < \pi^{-1}(c))$.

ALGORITHM 2.

Step 1. Determine if there exist $a, b, c, d \in \{1, 2, \ldots, n\}$ satisfying $a < b < c < d$ and $\pi^{-1}(c) < \pi^{-1}(a) < \pi^{-1}(d) < \pi^{-1}(b)$.

1.1. Generate all pairs $(\pi(i), \pi(j))$ satisfying $i < j$ and $\pi(i) < \pi(j)$.

1.2. For each pair $(\pi(i), \pi(j))$ created in Step 1.1, determine the maximal value in the set $\{\pi(i+1), \pi(i+2), \ldots, \pi(j-1)\}$. We denote the maximal value by $k_{i,j}$.

1.3. For each pair $(\pi(i), \pi(j))$ created in Step 1.1 satisfy-

ing $k_{i,j} > \pi(j)$, determine if there exists a value, denoted by $z_{i,j}$, in $\{\pi(1), \pi(2), \ldots, \pi(i-1)\}$ satisfying $\pi(j) < z_{i,j} < k_{i,j}$. If $z_{i,j}$ exists for at least one pair $(\pi(i), \pi(j))$, then the answer is *yes*. Otherwise, the answer is *no*.

Step 2. Determine if there exist $a, b, c, d \in \{1, 2, \ldots, n\}$ satisfying $a < b < c < d$ and $\pi^{-1}(b) < \pi^{-1}(d) < \pi^{-1}(a) < \pi^{-1}(c)$.

2.1. Generate all pairs $(\pi(i), \pi(j))$ satisfying $i < j$ and $\pi(i) > \pi(j)$.

2.2. For each pair $(\pi(i), \pi(j))$ created in Step 1.1, determine the maximal value in the set $\{\pi(i+1), \pi(i+2), \ldots, \pi(j-1)\}$. We denote the maximal value by $k_{i,j}$.

2.3. For each pair $(\pi(i), \pi(j))$ created in Step 1.1 satisfying $k_{i,j} > \pi(i)$, determine if there exists a value, denoted by $z_{i,j}$, in $\{\pi(j+1), \pi(j+2), \ldots, \pi(n)\}$ satisfying $\pi(i) < z_{i,j} < k_{i,j}$. If $z_{i,j}$ exists for at least one pair $(\pi(i), \pi(j))$, then the answer is *yes*. Otherwise, the answer is *no*.

Step 3. If the answers obtained in Steps 1 and 2 are all *no*, then the input graph G is a cograph. Otherwise, G is not a cograph.

The correctness of Algorithm 2 can be assured by letting $a = \pi(i)$, $b = \pi(j)$, $c = z_{i,j}$, and $d = k_{i,j}$ in Step 1 and $a = \pi(j)$, $b = \pi(i)$, $c = z_{i,j}$, and $d = k_{i,j}$ in Step 2.

Algorithm 2 can be executed in $O(\log n)$ time using $O(n^2)$ processors on CREW PRAM. Clearly, Steps 1.1 and 2.1 take $O(1)$ time if $O(n^2)$ processors are used. Steps 1.2 and 2.2 can be executed in $O(\log n)$ time using $O(n^2)$ processors if the well-known doubling technique [9] is applied. We simply assign $n-i$ processors to set $\{\pi(i+1), \pi(i+2), \ldots, \pi(n)\}$ for computing $k_{i,i+2}, k_{i,i+3}, \ldots, k_{i,n}$, $i = 1, 2, \ldots, n-2$. Step 1.3 can be executed in $O\log n)$ time using $O(n^2)$ processors if we first sort each set $\{\pi(1), \pi(2), \ldots, \pi(i-1)\}$, $i = 2, 3, \ldots, n$, using $O(i)$ processors, and then perform a binary search on the sorted list for finding $z_{i,1}, z_{i,2}, \ldots, z_{i,n}$. The sorting can be completed in $O(\log n)$ time, if cost-optimal sorting algorithms, e.g., Cole's sorting algorithm [5], are used. Similarly, Step 2.3 can be executed in $O(\log n)$ time using $O(n^2)$ processors.

If the given permutation graph is a cograph, we can construct its cotree from π by the following algorithm.

ALGORITHM 3.

Step 1. Construct (π) from π.

1.1. For each pair $(\pi(i), \pi(j))$, where $i < j$, find the maximal integer α and minimal integer β in $\{\pi(i), \pi(i+1), \ldots, \pi(j)\}$. If $\alpha - \beta = i - j$, then add a pair of parentheses, denoted by $p_{i,j}$, to enclose the integers $\pi(i), \pi(i+1), \ldots, \pi(j)$. We let $p_{i,j}$ be of type 0 if $\pi^{-1}(\alpha) > \pi^{-1}(\beta)$, and of type 1 otherwise.

1.2. Delete those $p_{i,j}$s that intersect with each other.

1.3. Set i as the weight of $\pi(i)$, $i=1,\ldots,n$, and set $i-(j-i)/n$, $i+(j-i)/n$ as the weights of the left parenthesis and the right parenthesis of $p_{i,j}$, respectively.

1.4. Sort $\pi(i)$, $i=1,\ldots,n$, and the left and right parentheses of remaining $p_{i,j}$s increasingly according to their weights. Suppose $(\pi)=c_1c_2\cdots c_k$ is the sorted list.

Step 2. Transform (π) into a set of plane points by the following rules. For each $\pi(i)=c_j$, $1\leqslant i\leqslant n$, $1\leqslant j\leqslant k$, generate a plane point $(-j,j)$. For each $p_{i,j}$ whose left parenthesis and right parenthesis are c_r and c_s, respectively, generate a plane point $(-r,s)$.

Step 3. Construct the cotree as follows. A pair of parentheses is the parent of another pair of parentheses (or an integer) if and only if the corresponding point of the former directly dominates the corresponding point of the latter. Also, a pair of parentheses is assigned as a 1-node of it is of type 1 and a 0-node if it is of type 0.

The correctness of Algorithm 3 is discussed as follows. In Step 1, because π is parenthesizable, we have $\{\pi(i),\pi(i+1),\ldots,\pi(j)\}$ parenthesizable if $\alpha-\beta=i-j$. Because the set of pairs of parentheses in (π) is exactly the set of $p_{i,j}$s that do not intersect with each other, all $p_{i,j}$s that intersect with each other are redundant and need to be deleted. It is clear that after setting weights as stated in Step 1.3, (π) will be obtained after sorting. So, Step 1 constructs (π) from π.

On the other hand, with transforming (π) into a set of plane points in Step 2, we have the fact that $p_{i,j}$ encloses $p_{r,s}$ and is of the smallest size (that is, $p_{i,j}$ is the parent of $p_{r,s}$ in the cotree) if and only if the corresponding point of $p_{i,j}$ directly dominates the corresponding point of $p_{r,s}$. Moreover, $p_{i,j}$ being of type 0 (type 1) means $\pi(i)\pi(i+1)\cdots\pi(j)=V_1\cdot V_2\cdot\cdots\cdot V_k$, where $V_1<V_2<\cdots<V_k$ $(V_1>V_2>\cdots>V_k)$, according to rule 2 of Definition 1. So, $p_{i,j}$ represents a 1-node if it is of type 1 and a 0-node if it is of type 0.

Algorithm 3 can be implemented in $O(\log n)$ time using $O(n^2)$ processors on CREW PRAM. Using the doubling technique, Step 1.1 can be implemented, similar to Step 1.2 of Algorithm 2, in $O(\log n)$ time using $O(n^2)$ processors.

Because $p_{i,j}$ intersects with $p_{r,s}$ if and only if $i<r<j<s$ or $r<i<s<j$, we can complete Step 1.2 in $O(\log n)$ time using $O(n^2)$ processors as follows. Let us define $w_i=\max\{j|$ for all $p_{i,j}$s created in Step 1.1$\}$, $i=1,2,\ldots,n$, and $m_j=\max\{i|$ for all $p_{i,j}$s created in Step 1.1$\}$, $j=1,2,\ldots,n$. Using the doubling technique, all w_is and m_is can be computed in $O(\log n)$ time using $O(n^2)$ processors. Then, $p_{i,j}$ intersects with some

other pairs of parentheses if and only if $\max\{w_{i+1}, w_{i+2}, \ldots, w_{j-1}\} > j$ or $\min\{m_{i+1}, m_{i+2}, \ldots, m_{j-1}\} < i$. Similar to Step 1.2 of Algorithm 2, $\max\{w_{i+1}, w_{i+2}, \ldots, w_{j-1}\}$ and $\min\{m_{i+1}, m_{i+2}, \ldots, m_{j-1}\}$ can be determined for all $p_{i,j}$s in $O(\log n)$ time using $O(n^2)$ processors.

Steps 1.3 and 1.4 take $O(1)$ and $O(\log n)$ time, respectively, if $O(n)$ processors are used. Step 2 takes $O(1)$ time using $O(n)$ processors. Step 3 can be completed in $O(\log n)$ time using $O(n^2)$ processors, if for each point, we first find all points that dominate it, and then determine their minimal set (in our case, the minimal set contains only one point) [2].

5. CONCLUDING REMARKS

It is known that the class of cographs is a subclass of permutation graphs, which is again a subclass of circle graphs [13]. In [19], Read, Rotem, and Urrutia found that a circle graph is a permutation graph if it satisfies a predefined condition. In this paper, we have shown in Theorem 1 that a permutation graph is a cograph if it satisfies either of two predefined conditions. Then, by the aid of Theorem 1, we designed sequential and parallel algorithms for recognizing a cograph. Also, the cotree is constructed if the input graph is recognized as a cograph. When the input graph is dense [i.e., $m = O(n \log n)$ or $O(n^2)$], our algorithms require less time and/or fewer processors than all previous ones. The improvement made by our algorithm has a sacrifice that the input graph has to be a permutation graph.

REFERENCES

1. G. S. Adhar and S. Peng, Parallel algorithms for cographs and parity graphs with applications, *J. Algor.* 11:252–284 (1990).
2. M. Atallah, R. Cole, and M. Goodrich, Cascading divide-and-conquer: a technique for designing parallel algorithms, in *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, 1987, pp. 151–120.
3. M. J. Atallah, G. K. Manacher, and J. Urrutia, Finding a minimum independent dominating set in a permutation graph, *Discrete Appl. Math.* 21:177–183 (1988).
4. A. Brandstadt and D. Kratsch, On domination problems for permutation and other graphs, *Theor. Comput. Sci.* 54:181–198 (1987).
5. R. Cole, Parallel merge sort, *SIAM J. Comput.* 17:770–785 (1988).
6. D. G. Corneil, H. Lerchs, and L. S. Burlingham, Complement reducible graphs, *Discrete Appl. Math.* 3:163–174 (1981).
7. D. G. Corneil, Y. Perl, and L. K. Stewart, A linear recognition algorithm for cographs, *SIAM J. Comput.* 14:926–934 (1985).
8. M. Farber and J. M. Keil, Domination in permutation graphs, *J. Algor.* 6:309–321 (1985).

9. A. Gibbons and W. Rytter, *Efficient Parallel Algorithms*, Cambridge University Press, Cambridge, 1988.
10. M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic, New York, 1980.
11. X. He, Parallel algorithm for cograph recognition with applications, Report TR 91-16, Computer Science State Dept., State University of New York at Buffalo, 1991.
12. D. Helmbold and E. Mayr, Perfect graphs and parallel algorithms, in *Proceedings of the International Conference on Parallel Processing*, 1986, pp. 853–860.
13. D. S. Johnson, Ed., The NP-completeness column: an ongoing guide, *J. Algor.* 6:434–451 (1985).
14. D. G. Kirkpatrick and T. Przytycka, Parallel recognition of complement reducible graphs and cotree construction, *Discrete Appl. Math.* 29:79–96 (1990).
15. D. Kozen, U. V. Vazirani, and V. V. Vazirani, NC algorithms for comparability graphs, interval graphs, and testing for unique perfect matching, in *Proceedings of the Fifth Conference on Foundation of Software Technology and Theoretical Computer Science*, New Dehli, 1985, pp. 498–503.
16. R. Lin and S. Olariu, An NC recognition algorithm for cographs, *J. Parallel Distributed Comput.* 13:76–90 (1991).
17. M. Novick, Fast parallel algorithms for modular decomposition, Report TR 89-1016, Computer Science Dept., Cornell University, 1989.
18. A. Pnueli, A. Lempel, and S. Even, Transitive orientation of graphs and identification of permutation graphs, *Can. J. Math.* 23:160–175 (1971).
19. R. C. Read, D. Rotem, and J. Urrutia, Orientations of circle graphs, *J. Graph Theory* 6:325–341 (1982).
20. C. H. Shyu, A fast algorithm for cographs, *French Israeli Conference on Combinatorics and Algorithms*, Nov. 1988.
21. J. Spinrad, On comparability and permutation graphs, *SIAM J. Comput.* 14:658–670 (1985).
22. J. Spinrad, A. Brandstadt, and L. Stewart, Bipartite permutation graphs, *Discrete Appl. Math.* 18:279–292 (1987).
23. K. J. Supowit, Decomposing a set of points into chains, with applications to permutation and circle graphs, *Inform. Process. Lett.* 21:249–252 (1985).
24. K. H. Tsai and W. L. Hsu, Fast algorithms for the dominating set problem on permutation graphs, *Lecture Notes in Computer Science: Algorithms*, vol. 450, Springer, Berlin, 1990, pp. 109–117.
25. C. W. Yu and G. H. Chen, Parallel algorithms for permutation graphs, Technical Report 91-11, Dept. of Computer Science & Information Engineering, National Taiwan Univ., July 1991.
26. C. W. Yu and G. H. Chen, The weighted maximum independent set problem in permutation graphs, Technical Report 91-12, Dept. of Computer Science & Information Engineering, National Taiwan Univ., July 1991.

Received 1 March 1992; revised 29 January 1993