Parallel Algorithm for Cograph Recognition with Applications

Xin He [1]
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260

Abstract

We present a parallel algorithm for recognizing cographs and constructing their cotrees. The algorithm takes $O(\log^2 n)$ time with $O(n+m)$ processors on a CRCW PRAM, where n and m are the number of vertices and edges of the graph. Using cotree representation, we obtain parallel algorithms for solving the maximum matching and the permutation representation problems for cographs using $O(\log n)$ time with $O(n)$ processors. We also obtain a parallel algorithm for the depth-first spanning tree problem for permutation graphs (a class properly contains cographs) which takes $O(\log^2 n)$ time with $O(n)$ processors.

1. Introduction

The *complement reducible graphs* (also called *cographs*) are defined as follows: (1) A graph consisting of a single vertex is a cograph; (2) If $G_1, G_2, \ldots, G_k$ are cographs, so is their union $G_1 \cup G_2 \cup \ldots \cup G_k$; (3) If G is a cograph, so is its complement.

Cographs have arisen in disparate areas of Mathematics and Computer Science and have been independently studied by various researchers [1, 2, 3, 4, 5, 9, 10, 11, 12, 14, 15, 20, 21]. Names synonymous with cographs include: D^*-graphs [14], P_4 restricted graphs [9, 11], Hereditary Dacey Graphs [20] and so on. Several characterizations of cographs are known. For examples, it is shown in [9] that G is a cograph iff G contains no P_4 (a path consisting of 4 vertices) as an induced subgraph. In [21], it was shown that G is a cograph iff G has an orientation G' such that G' is a transitive series parallel graph. The most important property of cographs for algorithm design is that G is a cograph iff G has a *cotree* representation (to be defined in section 2) [9]. Most algorithms for solving various cograph problems require the cotree as input. So the following cograph recognition problem is central to cograph algorithms: Determine if a given graph G is a cograph or not, and if so construct its cotree.

A linear time sequential algorithm for recognizing cographs was given in [11]. A parallel algorithm for solving this problem was given in [15] which takes $O(\log^2 n)$ time with $O(n^3/\log^2 n)$ processors

[1] Research supported by NSF grant CCR-9011214.

on a concurrent read exclusive write PRAM. Another parallel cograph recognition algorithm was developed in [4] which requires $O(\log^2 n)$ time with $O(nm)$ processors on a concurrent read concurrent write PRAM. In [18], a parallel algorithm for the modular decomposition problem was developed using $O(\log n)$ time with $O(n^3)$ processors on a concurrent read concurrent write PRAM. This implies a parallel cotree construction algorithm with the same time and processor bounds. A cotree construction algorithm was developed in [16] which uses $O(\log n)$ time with $O(\frac{n^2+mn}{\log n})$ processors on an exclusive read exclusive write PRAM. Using cotree representation, NC algorithms for many cograph problems were presented in [1, 2, 3, 4, 5, 6, 15], including: transitive orientation, maximum weighted clique, minimum coloring, minimum weighted domination, minimum fill-in and isomorphism testing for cographs. If the cotree is given, these algorithms take $O(\log n)$ time using a linear number of processors on a concurrent read exclusive write PRAM. Parallel algorithms for solving the Hamiltonian path, Hamiltonian cycle, minimum path covering, and maximum matching problems on cographs were developed in [5, 6]. If the cotree is given, these algorithms take $O(\log^2 n)$ time with $O(n^2)$ processors on a concurrent read concurrent write PRAM.

In this paper, we present several efficient parallel algorithms for cographs. The parallel computation model we use is CRCW PRAM. The model consists of a number of processors and a common memory. In each time unit, a processor can read a memory cell, perform an arithmetic or logic operation, and write into a memory cell. Both concurrent read and concurrent write are allowed. If a write conflict occurs, the processor with the lowest processor number succeeds.

We present an $O(\log^2 n)$ time $O(n+m)$ processors algorithm for recognizing cographs and constructing cotrees. Our algorithm is a refinement of the algorithms in [4, 15]. Since cotree construction dominates the complexity of the algorithms in [1, 2, 3, 4, 5, 6, 15] for solving various cograph problems, the efficiency of these algorithms is uniformly improved. We also present efficient parallel algorithms for solving three other problems. The first one is the maximum matching problem. Our algorithm takes $O(\log n)$ time with $O(n)$ processors which improves the $O(\log^2 n)$ time $O(n^2)$ processors algorithm in [6].

The second problem is the permutation representation of cographs. Cographs are a proper subclass of *permutation graphs* [12]. The permutation representation of permutation graphs can be found in $O(\log^2 n)$ time with $O(n^4)$ processors [13]. We show that if the cotree is given, the permutation representation of cographs can be constructed in $O(\log n)$ time with $O(n)$ processors.

The third one is the depth-first spanning tree (DFST) problem. This is a well-known hard problem for parallel algorithm design. No NC algorithm is known for solving this problem on general

graphs. We show that if the permutation representation is given, the DFST problem for permutation graphs (hence cographs) can be solved in $O(\log^2 n)$ time with $O(n)$ processors.

The present paper is organized as follows. Section 2 introduces the definitions and basic properties of cographs and cotrees. In sections 3 and 4, several lemmas needed by our algorithm are proved. The recognition algorithm is presented in section 5. The algorithms for solving various problems on cographs and permutation graphs are presented in sections 6, 7 and 8.

2. Properties of Cographs and Cotrees

Let $G = (V, E)$ be a graph with no self-loops and multiple edges. $|G|$ denotes the number of vertices of G. For $V_1 \subset V$, $G - V_1$ denotes the subgraph obtained from G by deleting the vertices in V_1. For each $v \in V$, define $N(v) = \{u \in V | (u, v) \in E\}$ and $\Gamma(v) = N(v) \cup \{v\}$. If $N(u) = N(v)$, u and v are called a *false twin* and denoted by $u \sim_f v$. If $\Gamma(u) = \Gamma(v)$, u and v are called a *true twin* and denoted by $u \sim_t v$. Note that $u \sim_f v$ implies $(u, v) \notin E$ and $u \sim_t v$ implies $(u, v) \in E$. Clearly both $\sim_f$ and $\sim_t$ are equivalence relations. The equivalence classes of V under the relation $\sim_t$ ($\sim_f$, respectively) will be called $\sim_t$ ($\sim_f$, respectively) classes.

Let T be a rooted tree. The nodes of T with no children are *leaf nodes*. Other nodes are *internal nodes*. An internal node is a *tip node* iff all of its children are leaf nodes. $L(T)$ denotes the set of leaf nodes of T. For each node x of T, $p_T(x)$ denotes the parent of x in T. $T(x)$ denotes the subtree of T rooted at x. $Leaf(x)$ denotes the set of the leaf **children** of x. If a node y is on the path from x to the root r of T, then y is an *ancestor* of x and x is a *descendent* of y and we say x and y are *related*. The first common node of the paths from x and y to r is called the *lowest common ancestor* of x and y and denoted by $lca_T(x, y)$. If y is an ancestor of x, then $lca_T(x, y) = y$.

Definition 1: A *cotree* is a rooted tree T with the following properties: (a) The internal nodes are labeled 0 or 1 such that the root of T is a 1-node; the children of a 1-node are 0-nodes; the children of a 0-node are 1-nodes. (b) Each internal node has at least two children, except that the root r might have only one child.

A cograph G can be naturally represented by a rooted tree T as follows. The leaf nodes of T represent the vertices of G. Each internal node of T represents either a "union" or a "complement" operator. A "union" node has at least two children. Each of them is either a "complement" node or a leaf node. A "complement" node has exactly one child which is a "union" node. We can modify T to be a cotree as follows. If the root r of T is a "union" node, add two new "complement" nodes r_1 and r_2 into T, make r a child of r_1 and r_1 a child of r_2 so that r_2 is the new root. (If r is a "complement" node, skip this step). Then merge each "union" node and its parent (which is a "complement" node)

into a single node. Finally, label the nodes of T so that Definition 1 (a) holds. The most important property of this representation is as follows [9]:

Property of Cotree: Let $G = (V, E)$ be a cograph and T be the cotree constructed as above. For any $x, y \in V$, $(x, y) \in E$ iff $lca_T(x, y)$ is a 1-node.

Fig. 1

If the root r of T has at least two children then G is connected. If r has only one 0-node child, then G is disconnected. If r has only one leaf child x, then G consists of a single vertex x. Fig 1 shows a cograph G and its cotree T. The following theorem is well-known.

Theorem 1 [9, 11]: G is a cograph iff G has a cotree representation T. Moreover, the cotree representation is unique up to the permutation of the children of the internal nodes of T. □

From the property of cotree, it's not hard to prove the following:

Lemma 2 [15]: Let G be a cograph with cotree T.

(1) $u \sim_t v$ ($u \sim_f v$, respectively) iff $p_T(u) = p_T(v)$ is a 1-node (0-node, respectively).

(2) For each 1-node α, $Leaf(\alpha)$ is a $\sim_t$ class. For each 0-node β and each $x \in Leaf(\beta)$, the singleton set $\{x\}$ is a $\sim_t$ class.

(3) For each 0-node β, $Leaf(\beta)$ is a $\sim_f$ class. For each 1-node α and each $x \in Leaf(\alpha)$, the singleton set $\{x\}$ is a $\sim_f$ class. □

For example, the $\sim_f$ classes of the graph G shown in Fig 1 are: $\{d, e, f\}$, $\{c\}$, $\{a\}$ and $\{b\}$. The $\sim_t$ classes of G are: $\{a, b\}$, $\{c\}$, $\{d\}$, $\{e\}$ and $\{f\}$.

3. Contracting Cotree

The basic idea of our cograph recognition algorithm is as follows. Given a graph G, the algorithm deletes some vertices from G to form a subgraph G' with the following properties: (a) $|G'| \leq c|G|$ for some constant $c < 1$. (b) G is a cograph iff G' is a cograph. (c) If G' is a cograph with cotree T', the cotree T of G can be constructed from T' easily.

The algorithm is recursively applied to G'. If G' is not a cograph, G is rejected. If G' is a cograph, the recursive call returns the cotree T' of G'. Then the cotree T of G is constructed from T'.

The problem is how to construct G'. Our method is developed along the following line: Assume G is a cograph with cotree T. We first define two operations *Trim-tree* and *Cut-tree*. These operations can be applied to T to produce a cotree T' such that $|T'| \leq c|T|$. We then define two operations *Trim-graph* and *Cut-graph* that, when applied to G, produce a new cograph G' corresponding to T'. Then we show Trim-graph and Cut-graph operations can be applied to *any* graph G to produce a

new graph G' satisfying the three requirements listed above. In this section we discuss the Trim-tree and Cut-tree operations.

Definition 2: Let T be a cotree. Let α be an internal node and $Q = Leaf(\alpha)$ with $|Q| > 1$. Let $q \in Q$. $Trim(T, q, Q)$ is the cotree obtained from T as follows: If α is a tip node then delete all nodes in Q and relabel α by q. If α is a non-tip node then delete all nodes in $Q - \{q\}$.

Definition 3: Let T be a cotree. *Trim-tree(T)* is the cotree obtained from T as follows. For each internal node α with $|Leaf(\alpha)| > 1$, select $q \in Leaf(\alpha)$ and perform $Trim(T, q, Leaf(\alpha))$.

An internal node α of T is *special* if α has exactly one leaf child and exactly one non-leaf child. A *trail* of T is a sequence $\{\alpha_1, \alpha_2, \ldots, \alpha_l\}$ of special nodes such that $\alpha_i = p_T(\alpha_{i-1})$ for $1 < i \leq l$. A *chain* of T is a **maximal** trail $C = \{\alpha_1, \alpha_2, \ldots, \alpha_l\}$ such that α_1 is a 0-node and α_l is a 1-node. (Note that l must be even.) The *length* of C is l. For each i, let u_i be the leaf child of α_i. We say u_i *belongs to* C. The sequence $L = \{u_1, \ldots, u_l\}$ is called a *line* of T.

Definition 4: Let T be a cotree and $L = \{u_1, \ldots, u_l\}$ be a line of T with $l \geq 4$. $Cut(T, L)$ is the cotree obtained from T as follows: Delete α_k, u_k for $2 \leq k \leq l-1$ and make α_1 a child of α_l.

Definition 5: Let T be a cotree. *Cut-tree(T)* is the cotree obtained from T as follows. Let $\{L_1, \ldots, L_r\}$ be all lines of T of length ≥ 4. Perform $Cut(T, L_i)$ for $1 \leq i \leq r$.

Fig 2a shows a cotree T. Fig 2b shows the cotree T' obtained by performing the Trim-tree and Cut-tree operations on T. In this figure, $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ is a chain.

Fig 2

Lemma 3: Let T' be the cotree obtained by performing the Trim-tree and Cut-tree operations on a cotree T. Then $|L(T')| \leq \frac{17}{18}|L(T)|$.

Proof: We assume the root r of T has at least two children. If not, delete r and apply the following arguments to the resulting tree. Partition $L(T)$ into two subsets: X consists of the leaf nodes that do not belong to any trail. Y consists of the leaf nodes that belong to a trail. Each of X and Y is further partitioned into two subsets as follows:

$L_1 = \{u \in X \mid p_T(u)$ has at least two leaf children$\}$.

$L_2 = \{u \in X \mid p_T(u)$ has exactly one leaf child$\}$.

$L_3 = \{u \in Y \mid u$ belongs to a maximal trail of length $\leq 5\}$.

$L_4 = \{u \in Y \mid u$ belongs to a maximal trail of length $\geq 6\}$.

For $i = 1, 2, 3, 4$, let $L'_i = L(T') \cap L_i$.

Let T_1 be the tree obtained from T by replacing each maximal trail C and the leaf nodes belonging to C by a single edge. Let T_2 be the tree obtained from T_1 as follows: For each non-tip internal

node x of T_1 with at least two leaf children, insert a new node x' between x and its leaf children. Note that $L(T_1) = L(T_2) = L_1 \cup L_2$. Let A be the set of tip nodes of T_2 and B be the set of non-tip internal nodes of T_2. (Fig 2c shows the tree T_2 obtained from the cotree T shown in Fig 2a. The maximal trial consisting of the nodes $\{\lambda, \alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ and the children of these nodes are deleted from T and replaced by a single edge (β, γ). A new node γ' is inserted between the node γ and its children. In T_2, $A = \{\delta, \beta, \gamma'\}$ and $B = \{r, \gamma\}$.) Each node in B has at most one leaf child and at least two internal node children. This implies $|B| \leq |A|$.

Since each node in A has at least two leaf children, and every leaf node in L_1 is a child of a node in A, we have $2|A| \leq |L_1|$.

Since each leaf node in L_2 is a child of a node in B and each node in B has at most one leaf child, we have $|L_2| \leq |B| \leq |A| \leq |L_1|/2$.

We next bound $|L_3|$. Each maximal trail of T corresponds to an edge between two internal nodes of T_1. So the number of maximal trails of T is bounded by the number of internal nodes of T_1. Since each node in L_3 belongs to a maximal trail of length at most 5, $|L_3|$ is at most 5 times the number of internal nodes of T_1. Therefore $|L_3|$ is at most 5 times the number of leaf nodes of T_1 (since each internal node of T_1 has at least two children). Thus we have $|L_3| \leq 5(|L_1| + |L_2|)$.

After performing the Trim-tree and Cut-tree operations on T, some nodes in L_1 are deleted by the Trim-tree operation and the number of remaining nodes is equal to the number of tip nodes of T_2. Namely $|L_1'| = |A|$. Some nodes in L_4 are deleted by the Cut-tree operation. It is easy to see at least one third of the nodes in L_4 are deleted by the Cut-tree operation. (The worst case is a line of length 6 and two nodes are deleted). Thus $|L_4'| \leq 2|L_4|/3$.

From the discussion above, we have:

$$|L(T)| = |L_1| + |L_2| + |L_3| + |L_4| \tag{1}$$

$$|L(T')| = |L_1'| + |L_2'| + |L_3'| + |L_4'| \leq |A| + |L_2| + |L_3| + 2|L_4|/3 \tag{2}$$

Consider the quotient $\frac{|A|+|L_2|+|L_3|}{|L_1|+|L_2|+|L_3|} \leq \frac{|L_1|/2+|L_2|+|L_3|}{|L_1|+|L_2|+|L_3|}$. Let $f(t) = \frac{|L_1|/2+t}{|L_1|+t}$ where $t = |L_2| + |L_3|$. It's easy to show $f(t)$ is an increasing function. Since $|L_2|+|L_3| \leq |L_2|+5(|L_1|+|L_2|) = 5|L_1|+6|L_2| \leq 8|L_1|$, $f(t) \leq f(8|L_1|) = \frac{17}{18}$. Combined with (1) and (2), we get $|L(T')| \leq \frac{17}{18}|L(T)|$. □

We next define two operations *UnTrim* and *UnCut* which are the "inverses" of the operations $Trim$ and Cut, respectively.

Definition 6: Let T be a cotree, q be a leaf node and $\alpha = p_T(q)$. Let Q be a set such that $Q \cap L(T) = \{q\}$. $UnTrim_t(T, q, Q)$ ($UnTrim_f(T, q, Q)$, respectively) is the cotree obtained from T

as follows: If α is a 1-node (0-node, respectively) then for each $x \in Q - \{q\}$ add a new leaf child x to α. If α is a 0-node (1-node, respectively) then delete q; add a new 1-node (0-node, respectively) β, make β a child of α and for each $x \in Q$ add a new leaf child x to β.

Definition 7: Let T be a cotree and u_1, u_l be two leaf nodes of T such that $\alpha_1 = p_T(u_1)$ is a 0-node, $\alpha_l = p_T(u_l)$ is a 1-node and $\alpha_l = p_T(\alpha_1)$. Let $L = \{u_1, u_2, \ldots, u_l\}$ be a set such that l is even and $L \cap L(T) = \{u_1, u_l\}$. $UnCut(T, L)$ is the cotree obtained from T as follows. Add a sequence of new nodes $\alpha_2, \ldots, \alpha_{l-1}$ between α_1 and α_l. Label α_k's ($2 \leq k \leq l-1$) so that the 0-nodes and 1-nodes alternate on the path $\{\alpha_1, \alpha_2, \ldots, \alpha_l\}$. For each $2 \leq k \leq l-1$, add a leaf child u_i to α_i.

4. Contracting Graphs

In this section we define the operations on a graph G that correspond to the effects of the Trim-tree and Cut-tree operations on T. The operation corresponds to Trim-tree is easy. By the property of cotree, it's straightforward to prove the following:

Lemma 4 [15]: Let $G = (V, E)$ be a graph. Let Q be a $\sim_t$ ($\sim_f$, respectively) class of V with $|Q| > 1$. Let $q \in Q$ and $G' = G - (Q - \{q\})$. Then G is a cograph with cotree T iff G' is a cograph with cotree T', where T and T' are related by $T' = Trim(T, q, Q)$ and $T = UnTrim_t(T', q, Q)$ ($T = UnTrim_f(T', q, Q)$, respectively). □

Let $G = (V, E)$ be a graph. Consider the equivalence classes of V under the relations $\sim_t$ and $\sim_f$. Let $Q_1, \ldots, Q_s$ be the $\sim_t$ classes containing more than one vertex. Let $R_1, \ldots, R_t$ be the $\sim_f$ classes containing more than one vertex. The collection $\Psi(G) = \{Q_1, \ldots, Q_s, R_1, \ldots, R_t\}$ is called the *class partition* of G. If $Q_i \cap R_j = \emptyset$ for $1 \leq i \leq s$ and $1 \leq j \leq t$, we say $\Psi(G)$ is *valid*.

Definition 8: Let $\Psi(G) = \{Q_1, \ldots, Q_s, R_1, \ldots, R_t\}$ be the class partition of a graph G. If $\Psi(G)$ is valid, then *Trim-graph(G)* is the graph obtained from G as follows: For each $Q_i \in \Psi(G)$, select $q_i \in Q_i$ and delete the vertices in $Q_i - \{q_i\}$. For each $R_j \in \Psi(G)$, select $r_j \in R_j$ and delete the vertices in $R_j - \{r_j\}$.

Lemma 5: (1) If G is a cograph then the class partition $\Psi(G)$ is valid.

(2) Suppose the class partition $\Psi(G)$ of G is valid. Then G is a cograph iff $G' = Trim\text{-}graph(G)$ is a cograph.

Proof: (1) Let T be the cotree of G. By Lemma 2, the collection $\{Leaf(\alpha) \mid \alpha$ is an internal node of T and $|Leaf(\alpha)| > 1\}$ is the class partition $\Psi(G)$ which is clearly valid.

(2) Suppose G is a cograph with cotree T. The elements of $\Psi(G)$ are disjoint $\sim_t$ or $\sim_f$ classes. By repeated applications of Lemma 4, it can be shown $T' = Trim\text{-}tree(T)$ is a cotree of G'. Hence G' is a cograph. Conversely, suppose G' is a cograph with cotree T'. Let T be the cotree obtained

from T' by performing $UnTrim_t(T', q_i, Q_i)$ for each $\sim_t$ class $Q_i \in \Psi(G)$ and $UnTrim_f(T', r_j, R_j)$ for each $\sim_f$ class $R_j \in \Psi(G)$. By repeated applications of Lemma 4, it can be shown T is a cotree of G. Hence G is a cograph. □

We next investigate the operation on a graph that corresponds to the Cut operation. Let G be a cograph with cotree T. Let $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ be a chain of T and u_i be the leaf child of α_i $(1 \leq i \leq 4)$. The following definition is motivated by the relationships of the neighborhoods of u_i's in G.

Definition 9: Let $G = (V, E)$ be a graph. Let u_1, u_2, u_3, u_4 be four vertices of G.

(1) $\{u_1, u_2, u_3\}$ is a *false trio* iff: (1a) $N(u_3) \subset N(u_1)$ and $N(u_1) - N(u_3) = \{u_2\}$; and (1b) both $\{u_1\}$ and $\{u_3\}$ are $\sim_t$ class and $\sim_f$ class.

(2) $\{u_2, u_3, u_4\}$ is a *true trio* iff: (2a) $\Gamma(u_2) \subset \Gamma(u_4)$ and $\Gamma(u_4) - \Gamma(u_2) = \{u_3\}$; and (2b) both $\{u_2\}$ and $\{u_4\}$ are $\sim_t$ class and $\sim_f$ class.

(3) $\{u_1, u_2, u_3, u_4\}$ is a *4-twin* of G iff $\{u_1, u_2, u_3\}$ is a false trio and $\{u_2, u_3, u_4\}$ is a true trio.

Fig 3

Lemma 6: Let G be a cograph with cotree T. Let $u_1, u_2, u_3, u_4 \in V$ and $\alpha_i = p_T(u_i)$ $(1 \leq i \leq 4)$.

(1) $\{u_1, u_2, u_3\}$ is a false trio iff they are positioned in T as shown in Fig 3a. $\{u_2, u_3, u_4\}$ is a true trio iff they are positioned in T as shown in Fig 3b.

(2) $\{u_1, u_2, u_3, u_4\}$ is a 4-twin iff: (a) α_1 and α_3 are 0-nodes, α_2 and α_4 are 1-nodes; (b) $\alpha_{i+1} = p_T(\alpha_i)$ for $1 \leq i < 4$; (c) For $1 \leq i \leq 4$, u_i is the only leaf child of α_i (Fig 3c).

(3) For any vertex $u_2, u_3 \in V$, u_2 and u_3 belongs to at most one true trio $\{u_2, u_3, u_4\}$.

(4) If $\{u_1, u_2, u_3, u_4\}$ is a 4-twin, then $N(u_1) \subset N(u_2)$ and $N(u_3) \subset N(u_2)$.

Proof: (1) was proved in [15]. (2) and (3) immediately follow from (1). (4) immediately follows from (2) and the property of cotree. □

Let $G = (V, E)$ be a graph. A set $L = \{u_1, \ldots, u_l\} \subset V$, where l is even and ≥ 4, is called a *contractable sequence* iff $\{u_i, u_{i+1}, u_{i+2}, u_{i+3}\}$ is a 4-twin for each odd $i \leq l - 3$.

Lemma 7: Let $G = (V, E)$ be a graph and $L = \{u_1, \ldots, u_l\} \subset V$. Let $G_1 = G - (L - \{u_1, u_l\})$. Suppose L is a contractable sequence of G. Then G is a cograph with cotree T iff G_1 is a cograph with cotree T_1 such that $\alpha_1 = p_{T_1}(u_1)$ is a 0-node, $\alpha_l = p_{T_1}(u_l)$ is a 1-node, and $\alpha_l = p_{T_1}(\alpha_1)$, where T and T_1 are related by $T_1 = Cut(T, L)$ and $T = UnCut(T_1, L)$.

Proof: Suppose $l = 2k$. The proof is by induction on k. First consider the base case $k = 2$.

Suppose G is a cograph with cotree T. Since $L = \{u_1, u_2, u_3, u_4\}$ is a 4-twin, $T_1 = Cut(T, L)$ can be constructed by Lemma 6 (2). For any $x, y \in V - \{u_2, u_3\}$, $lca_T(x, y) = lca_{T_1}(x, y)$. Thus (x, y) is an edge in G_1 iff (x, y) is an edge in G iff $lca_T(x, y)$ is a 1-node iff $lca_{T_1}(x, y)$ is a 1-node. So T_1

is a cotree of G_1 and G_1 is a cograph. By the definition of the Cut operation, $p_{T_1}(u_1) = p_T(u_1)$ is a 0-node, $p_{T_1}(u_4) = p_T(u_4)$ is a 1-node, and $p_{T_1}(u_4)$ is the parent of $p_{T_1}(u_1)$ in T_1.

Suppose G_1 is a cograph with cotree T_1 such that $p_{T_1}(u_1)$ is a 0-node, $p_{T_1}(u_4)$ is a 1-node, and $p_{T_1}(u_4)$ is the parent of $p_{T_1}(u_1)$ in T_1. Then $T = UnCut(T_1, L)$ can be constructed. For any $x, y \in V - \{u_2, u_3\}$, $lca_T(x, y) = lca_{T_1}(x, y)$. Thus (x, y) is an edge in G iff (x, y) is an edge in G_1 iff $lca_{T_1}(x, y)$ is a 1-node iff $lca_T(x, y)$ is a 1-node. In order to show T is a cotree of G, we need to show for $\{x, y\} \cap \{u_2, u_3\} \neq \emptyset$, (x, y) is an edge of G iff $lca_T(x, y)$ is a 1-node.

Case 1: $x = u_2$ and $y = u_3$. In this case, $lca_T(u_2, u_3) = p_T(u_3)$ is a 0-node. On the other hand, $u_2 \in N(u_1) - N(u_3)$ implies $(u_2, u_3) \notin E$.

Case 2: $x \notin \{u_2, u_3\}$ and $y = u_3$.

Case 2a: $x \in L(T(\alpha_1))$ (Fig 4a). Then $lca_T(x, u_3) = p_T(u_3) = \alpha_3$ is a 0-node. If $x = u_1$ then $u_1 \notin N(u_1)$ and $N(u_3) \subset N(u_1)$ implies $(u_1, u_3) \notin E$. Suppose $x \neq u_1$. Since $lca_{T_1}(x, u_1) = \alpha_1$ is a 0-node, (x, u_1) is not an edge in G_1. So (x, u_1) is not an edge in G and $x \notin N(u_1)$. Since $N(u_3) \subset N(u_1)$, we have $x \notin N(u_3)$. Hence $(x, u_3) \notin E$.

Fig 4

Case 2b: $x \in L(T(\alpha_4)) - L(T(\alpha_3))$ (Fig 4b). Suppose $lca_T(x, u_3)$ is a 1-node. Since $lca_{T_1}(x, u_1) = lca_T(x, u_3)$ is a 1-node, (x, u_1) is an edge in G_1. So (x, u_1) is also an edge in G and $x \in N(u_1)$. Since $N(u_1) - N(u_3) = \{u_2\}$ and $x \neq u_2$, we have $x \in N(u_3)$ and $(x, u_3) \in E$. Suppose $lca_T(x, u_3)$ is a 0-node. Since $lca_{T_1}(x, u_1) = lca_T(x, u_3)$ is a 0-node, (x, u_1) is not an edge in G_1. Hence (x, u_1) is not an edge in G and $x \notin N(u_1)$. Since $N(u_3) \subset N(u_1)$, we have $x \notin N(u_3)$ and $(x, u_3) \notin E$.

Case 3: $x = u_2$ and $y \notin \{u_2, u_3\}$. Similar to case 2, we can show $(u_2, y) \in E$ iff $lca_T(u_2, y)$ is a 1-node.

Thus T is a cotree of G and G is a cograph. This completes the proof for the base case.

The induction step follows from the following easily proven claims: (a) If L is a contractable sequence of G, then $L' = \{u_1, u_2, u_3, u_4\}$ is a contractable sequence of G and $L'' = L - \{u_2, u_3\}$ is a contractable sequence of $G' = G - \{u_2, u_3\}$. (b) $Cut(T, L) = Cut((Cut(T, L'), L'')$ and $UnCut(T_1, L) = UnCut((UnCut(T_1, L''), L')$. □

Let F be a directed forest (the edges are directed toward the roots of the subtrees). A vertex v of F is a *branch vertex* if v has at least two incoming arcs. A *walk* of F is a maximal directed path P in F such that: (1) the starting vertex of P is either a branch vertex or a vertex with indegree 0; and (2) the end vertex of P is either a branch vertex or a vertex with outdegree 0. The *walk decomposition* of F, denoted by $WD(F)$, is the set of all walks of F.

Let $G = (V, E)$ be a graph. In order to define the Cut-graph operation on G, we first construct a directed graph $H(G)$ as follows. The vertex set of $H(G)$ is V. For each 4-twin $\{u_1, u_2, u_3, u_4\}$ of G, there are three arcs $u_1 \rightarrow u_2$, $u_2 \rightarrow u_3$, $u_3 \rightarrow u_4$ in $H(G)$. (The multiple arcs of $H(G)$ are deleted.) If $H(G)$ is a forest and each walk in $WD(H(G))$ corresponds to a contractable sequence of G, we say $H(G)$ is a *consistent decomposition* of G.

Definition 10: Suppose $H(G)$ is a consistent decomposition of G. Let $\{L_1, \ldots, L_r\}$ be the set of contractable sequences obtained from $WD(H(G))$. For each L_i, suppose $L_i = \{u_1^i, u_2^i, \ldots, u_{l_i}^i\}$ for some l_i. *Cut-graph(G)* is the graph obtained from G as follows: For each $L_i \in WD(H(G))$, delete the vertices u_k^i $(2 \leq k \leq l_i - 1)$ from G.

Note that the Trim-graph operation deletes the vertices whose $\sim_t$ or $\sim_f$ class contains more than one vertex. On the other hand, the Cut-graph operation deletes the vertices whose $\sim_t$ and $\sim_f$ class contains exactly one vertex.

Lemma 8: (1) If G is a cograph then $H(G)$ is a consistent decomposition of G.

(2) Suppose $H(G)$ is a consistent decomposition of G. Let $\{L_1, \ldots, L_r\}$ be the set of the contractable sequences obtained from $WD(H(G))$. Then G is a cograph iff $G' = Cut\text{-}graph(G)$ is a cograph whose cotree T' satisfies the following conditions: For each $L_i = \{u_1^i, \ldots, u_{l_i}^i\}$, $p_{T'}(u_1^i)$ is a 0-node, $p_{T'}(u_{l_i}^i)$ is a 1-node, and $p_{T'}(u_{l_i}^i)$ is the parent of $p_{T'}(u_1^i)$ in T'.

Proof: (1) Let T be the cotree of G. Consider the maximal trails $C_1, \ldots, C_r$ of T. Suppose $C_i = \{\alpha_1^i, \ldots, \alpha_{l_i}^i\}$. Let α_0^i be the non-leaf child of α_1^i and $\alpha_{l_i+1}^i = p_T(\alpha_{l_i}^i)$. If α_0^i has exactly one leaf child, include α_0^i into C_i. If $\alpha_{l_i+1}^i$ has exactly one leaf child, include $\alpha_{l_i+1}^i$ into C_i. Let C_i' be the maximal sub-sequence of C_i such that the first node of C_i' is a 0-node and the last node of C_i' is a 1-node. To simplify the notation, the nodes of C_i' are still indexed by $1, \ldots, l_i$. Let u_k^i be the leaf child of α_k^i $(1 \leq k \leq l_i)$. By Lemma 6 (2), $\{u_j^i, u_{j+1}^i, u_{j+2}^i, u_{j+3}^i\}$ $(1 \leq j \leq l_i - 3$ and j is odd) is a 4-twin and it introduces three arcs $u_j^i \rightarrow u_{j+1}^i \rightarrow u_{j+2}^i \rightarrow u_{j+3}^i$ into $H(G)$. By Lemma 6 (2), these are the only arcs in $H(G)$. Clearly $H(G)$ is a forest and each walk in $WD(H(G))$ corresponds to a contractable sequence of G. Hence $H(G)$ is a consistent decomposition of G.

(2) Since G' is obtained from G by deleting vertices in the disjoint contractable sequences $L_i \in WD(H(G))$, this can be shown by repeated applications of Lemma 7. □

5. Recognition Algorithm

We are now ready to present our cograph recognition algorithm **Test**. The algorithm takes two parameters: The first one is the input graph G. The second one is an integer *depth* used to control the recursion depth. On a graph G of n vertices, the algorithm is invoked by Test$(G, \lceil \log_{18/17} n \rceil)$.

Algorithm 1: Test$(G, depth)$

(0) If $depth = 0$ and $|G| > 1$ then G is not a cograph. Return "no".

(1) If G has only one vertex, then G is a cograph. Return its trivial cotree.

(2) Find the class partition $\Psi(G) = \{Q_1, \ldots, Q_s, R_1, \ldots, R_t\}$. If $\Psi(G)$ is not valid, return "no".

(3) Construct the directed graph $H(G)$. If $H(G)$ is not a consistent decomposition of G, return "no". Otherwise let $WD(H(G)) = \{L_i\}$ be the set of contractable sequences obtained from $H(G)$.

(4) Using $\Psi(G)$ and $WD(H(G))$ found in Steps (2) and (3), perform the Trim-graph and Cut-graph operations on G. Let G' be the resulting graph.

(5) Call Test$(G', depth - 1)$.

(6) If G' is not a cograph, then G is not a cograph. Return "no".

(7) Suppose G' is a cograph with cotree T'. For each $L_i \in WD(H(G))$, check if the conditions in Lemma 8 (2) hold. If not, G is not a cograph, return "no".

(8) Construct a cotree T as follows. For each $Q_i, R_j \in \Psi(G)$, perform $UnTrim_t(T', q_i, Q_i)$ and $UnTrim_f(T', r_j, R_j)$. For each $L_i \in WD(H(G))$, perform $UnCut(T', L_i)$. Return the resulting tree T as the cotree of G.

End.

Theorem 9: The algorithm Test correctly recognizes cographs and constructs their cotrees.

Proof: By induction on $|G|$. If $|G| = 1$, the algorithm is correct by Step 1. Suppose the algorithm is correct on all graphs with $< n$ vertices. Consider a graph G with n vertices.

Suppose G is a cograph. By Lemmas 5 and 8, the valid class partition $\Psi(G)$ and the consistent decomposition $H(G)$ of G can be successfully found in Steps 2 and 3. By Lemmas 5 and 8, the graph G' obtained in Step 4 is a cograph. By the induction hypothesis, the algorithm correctly recognizes G' as a cograph and constructs its cotree T'. By Lemma 8, the tests in Step 7 are satisfied and the tree T constructed in Step 8 is a cotree of G. Hence the algorithm recognizes G as a cograph and outputs its cotree T. By Lemma 3, the recursion depth is bounded by $\lceil \log_{18/17} n \rceil$. So the recursion depth control statement (0) has no effect in this case.

Suppose G is not a cograph. Five cases may occur: (a) $\Psi(G)$ is not valid. (b) $H(G)$ is not a consistent decomposition of G. (c) $\Psi(G)$ is valid and $H(G)$ is a consistent decomposition, but G' constructed in Step 4 is not a cograph. (d) G' is a cograph, but at least one test fails in Step 7. (e) After $\lceil \log_{18/17} n \rceil$ recursive calls, the graph still contains more than one vertex. In any case the algorithm correctly rejects G. □

We now turn to the implementation of the algorithm. By Lemma 3, the recursion depth of the

algorithm is bounded by $\lceil \log_{18/17} n \rceil$. We show each iteration of Test can be implemented in $O(\log n)$ time with $O(n+m)$ processors. Steps 0, 1, 5, and 6 are trivial. We discuss other steps one by one.

Let $\{1, \ldots, n\}$ denote the vertices of $G = (V, E)$. For each $i \in V$, $deg(i)$ denotes the degree of i in G. $N[i]$ is an array containing $deg(i)$ entries. Each entry corresponds to a vertex in $N(i)$. $\Gamma[i]$ is an array containing $deg(i)+1$ entries. Each entry corresponds to a vertex in $\Gamma(i)$. We assume the entries in the arrays $N[i]$'s and $\Gamma[i]$'s are sorted in the increasing order. This can be done in $O(\log n)$ time with $O(n+m)$ processors from the input [8].

Given two arrays $X[1..a]$ and $Y[1..b]$, define $diff_L(X,Y) = \min\{i \mid X[i] \neq Y[i]\}$. (If $a < b$ and $X[i] = Y[i]$ for all $1 \leq i \leq a$, define $diff_L(X,Y) = a+1$. If $a = b$ and $X[i] = Y[i]$ for all $1 \leq i \leq a$, $diff_L(X,Y)$ is not defined.) Define $X <_L Y$ iff $X[d] < Y[d]$ where $d = diff_L(X,Y)$. (If $a < b$ and $X[i] = Y[i]$ for all $1 \leq i \leq a$, define $X <_L Y$). Define $diff_R(X,Y) = \min\{i \mid X[a+1-i] \neq Y[b+1-i]\}$. The relation $X <_R Y$ is defined analogous to $<_L$.

In order to implement Step 2, we need to find the $\sim_f$ classes of V. We sort the set of the arrays $\{N[i] \mid 1 \leq i \leq n\}$ by $<_L$. Since $i \sim_f j$ iff $N[i] = N[j]$, each $\sim_f$ class corresponds to a contiguous block in the sorted list. We use the $O(\log n)$ time sorting algorithm in [8]. For an array $N[i]$ of size $k = deg(i)$, we assign k processors. The comparison between two arrays $N[i]$ and $N[j]$ can be done in $O(1)$ time using the processors assigned to them. The total number of processors needed is $\sum_{i=1}^{n} deg(i) = 2m$. So this step takes $O(\log n)$ time with $O(m)$ processors. The $\sim_t$ classes can be found by using the same method on the set of the arrays $\{\Gamma[i] \mid 1 \leq i \leq n\}$. Let $Q_1, \ldots, Q_s$ be the $\sim_t$ classes and $R_1, \ldots, R_t$ be the $\sim_f$ classes containing more than one vertex. We need to check $Q_c \cap R_d = \emptyset$ for $1 \leq c \leq s$ and $1 \leq d \leq t$. This can be done as follows. For each vertex i of G, we keep a record $A(i)$ that indicates the $\sim_t$ class Q_c containing i and the number of elements in Q_c. Similarly, we keep a record $B(i)$ that indicates the $\sim_f$ class R_d containing i and the number of elements in R_d. The records $A(i)$ and $B(i)$ (for all vertices i) can be computed as a by-product of the sorting. We assign one processor to each vertex i. By using the information stored in $A(i)$ and $B(i)$, we can easily determine if there is any vertex i such that $i \in Q_c \cap R_d$ and both Q_c and R_d contains more than one vertex. Thus this step totally takes $O(1)$ time with $O(n)$ processors.

In order to implement Step 3, we need to find all 4-twins of G. This is the most difficult part of the implementation. We break it into several easier problems. Three vertices $\{u, v, w\}$ are called a *triple* of G if $\Gamma(u) \subset \Gamma(w)$ and $\Gamma(w) - \Gamma(u) = \{v\}$. We first discuss how to find the triples of G.

For $u, w \in V$ define: $l(u,w) = diff_L(\Gamma[u], \Gamma[w])$ and $r(u,w) = diff_R(\Gamma[u], \Gamma[w])$. Clearly $\{u, v, w\}$ is a triple iff: (1) $deg(u) = deg(w) - 1$; (2) $l(u,w) + r(u,w) = deg(w) + 2$; and (3)

$\Gamma[w][l(u,w)] = v$. So the problem is reduced to computing $l(u,w)$'s and $r(u,w)$'s. We only discuss the computation of $l(u,w)$'s. The computation of $r(u,w)$'s is similar.

Lemma 10: (1) If $\{u,v,w\}$ is a triple, then $\Gamma[w] <_L \Gamma[u]$.

(2) Let $w, u_1, u_2 \in V$ with $deg(u_1) = deg(u_2) = deg(w) - 1$. If $\Gamma[w] <_L \Gamma[u_1] <_L \Gamma[u_2]$, then $l(u_1,w) \geq l(u_2,w)$.

Proof: (1) Suppose $\{u,v,w\}$ is a triple. Then $\Gamma[u][j] = \Gamma[w][j]$ for all $j < l(u,w)$, $\Gamma[w][l(u,w)] = v$ and $\Gamma[u][l(u,w)] = \Gamma[w][l(u,w)+1]$. Since the entries in the array $\Gamma[w]$ are sorted in the increasing order, we have $\Gamma[w][l(u,w)] < \Gamma[w][l(u,w)+1]$. Hence $\Gamma[w] <_L \Gamma[u]$.

(2) Let $t = l(u_1,u_2)$ and $s = l(u_1,w)$.

Case 1: $s < t$. Since $\Gamma[w][s] \neq \Gamma[u_1][s]$ and $\Gamma[u_1][s] = \Gamma[u_2][s]$, we have $\Gamma[w][s] \neq \Gamma[u_1][s]$. Hence $l(u_2,w) = diff_L(\Gamma[u_2],\Gamma[w]) \leq s = l(u_1,w)$.

Case 2: $s > t$. Since $\Gamma[w][t] = \Gamma[u_1][t]$ and $\Gamma[u_1][t] \neq \Gamma[u_2][t]$, we have $\Gamma[w][t] \neq \Gamma[u_2][t]$. Hence $l(u_2,w) = diff_L(\Gamma[u_2],\Gamma[w]) \leq t < s = l(u_1,w)$.

Case 3: $s = t$. Since $\Gamma[w] <_L \Gamma[u_1] <_L \Gamma[u_2]$, we have $\Gamma[w][t] < \Gamma[u_1][t] < \Gamma[u_2][t]$. Hence $l(u_2,w) = t = s = l(u_1,w)$. □

We partition the set of the arrays $\{\Gamma[i] \mid i \in V\}$ into blocks $B_1, \ldots, B_\Delta$ (Δ is the maximum degree of G), where $B_k = \{\Gamma[i] \mid deg(i) = k\}$. The arrays within each block B_k are sorted in non-increasing order under $<_L$. This can be done in $O(\log n)$ time with $O(n+m)$ processors [8].

For each $w \in V$, we want to find u (if it exists) such that $\{u,v,w\}$ is a triple for some v. If $\{u,v,w\}$ is a triple, then $\Gamma(u) \subset \Gamma(w)$. This implies $u \in N(w)$. So we only need to search the neighbors of w for u. Lemma 10 suggests a binary-search-like algorithm as follows.

Algorithm 2:

(1) Let $U = \{u_1, \ldots, u_q\}$ be the set of the neighbors of w with degree $deg(w) - 1$. (U is sorted in non-increasing order of $\Gamma(u_i)$ under $<_L$.)

(2) Do a binary search in the set $\{\Gamma[u_i] \mid u_i \in U\}$ to find the largest index l such that $\Gamma(w) <_L \Gamma(u_l)$. For each j with $l < j \leq q$, let $l(u,w_j) = -1$. (By Lemma 10 (1), u_j (for $j > l$) and w cannot form a triple with any v. So $l(u_j,w)$ is assigned a dummy value -1.)

(3) Call $Find(1,l,1,deg(w)+1)$.

End.

The steps (1) and (2) clearly take $O(\log n)$ time with $k = deg(w)$ processors. The recursive procedure $Find(a,b,c,d)$ finds $l(u_i,w)$ for each $u_i \in U$ with $a \leq i \leq b$. The procedure call is concerned only with the array entries whose indices are between c and d. This procedure uses a

subroutine $Comp(\Gamma[u], \Gamma[w], c, d)$ which, given two arrays $\Gamma[u]$, $\Gamma[w]$ and two integers $c \leq d$, computes $\min\{j \mid c \leq j \leq d, \Gamma[u][j] \neq \Gamma[w][j]\}$. This takes $O(1)$ time with $d - c$ processors.

Procedure Find(a,b,c,d):

If $c = d$ then for all i with $a \leq i \leq b$, let $l(u_i, w) = c$.

Else do:

$e = \lceil (a + b)/2 \rceil$;

$f = Comp(\Gamma[u_e], \Gamma[w], c, d)$;

let $l(u_e, w) = f$;

In parallel, call $Find(a, e - 1, c, f)$ and $Find(e + 1, b, f, d)$;

end do;

End.

From Lemma 10 (2), it is easy to show Algorithm 2 correctly computes $l(u_i, w)$ for all u_i with $i \leq l$. Since $|U| \leq k = deg(w)$ and each array $\Gamma[u_i]$ ($u_i \in U$) has size $\leq k$, this computation takes $O(\log n)$ time with k processors. Thus $l(u, w)$'s for all u, w can be computed in $O(\log n)$ time with $O(n+m)$ processors. Similarly $r(u, w)$'s for all u, w can be computed in $O(\log n)$ time with $O(n+m)$ processors. From $l(u, w)$ and $r(u, w)$, all triples of G can be identified in $O(1)$ time with $O(n + m)$ processors.

A triple $\{u, v, w\}$ is a true trio iff both $\{u\}$ and $\{w\}$ are $\sim_f$ and $\sim_t$ classes. Since the $\sim_f$ and $\sim_t$ classes have been computed in Step 2, this can be checked in $O(1)$ time with $O(n + m)$ processors.

For each true trio $\{u, v, w\}$, we need to determine if there exists a vertex z such that $\{z, u, v, w\}$ is a 4-twin. If $\{z, u, v, w\}$ is a 4-twin, then we have $u \in N(z) - N(v)$ which implies $z \in N(u)$. Moreover, by Lemma 6 (4), $N(z) \subset N(u)$ and $N(v) \subset N(u)$. Let $p = deg(u)$ and $W = \{j \in N(u) \mid deg(j) \leq p\}$. In order to find z, we only need to search the vertices in W. By Lemma 6 (3), if G is a cograph then u, v belongs to at most one true trio $\{u, v, w\}$. Thus our task is to search the set of the arrays $\{N[j] \mid j \in W\}$ against the array $N[v]$ to find z such that $N(v) \subset N(z)$ and $N(z) - N(v) = \{u\}$. Since $|W| \leq deg(u) = p$ and the size of the arrays $N(v)$ and $N[j]$ (for $j \in W$) is $\leq p$, this task can be carried out by using the similar method for finding the triples. It takes $O(\log n)$ time with p processors. Hence the total number of processors needed for finding all 4-twins is $O(n + m)$.

If $\{u, v, w, z\}$ is a 4-twin of G, three arcs $u \rightarrow v \rightarrow w \rightarrow z$ are added in to $H(G)$. Since $H(G)$ is acyclic, in order to determine if $H(G)$ is a forest, we only need to check no vertex in $H(G)$ has outdegree greater than 1. This can be done in $O(1)$ time with $O(n)$ processors. Then the walk decomposition $WD(H(S))$ can be found in $O(\log n)$ time with $O(n)$ processors by using the pointer

jump method.

Step 4: The Trim-graph and Cut-graph operations simply delete some vertices from G and can be done in $O(\log n)$ time with $O(n+m)$ processors.

Steps 7 and 8: The testing and the construction of T from T' can be done in $O(1)$ time with $O(n)$ processors. This completes the discussion of the implementation. In summary we have:

Theorem 11: The cograph recognition and cotree construction problem can be solved in $O(\log^2 n)$ time with $O(n+m)$ processors on a CRCW PRAM.

6. Maximum Matching

A *matching* of a graph $G=(V,E)$ is a subset $M \subseteq E$ such that no two edges in M share a common end vertex. A *maximum matching* (MM) is a matching with maximum cardinality. If a vertex $x \in V$ is incident to an edge in M, x is *M-saturated*. Otherwise x is *M-unsaturated*. M is a *perfect matching* if all vertices of G are M-saturated. M is a *nearly perfect matching* of G if either $|G|$ is even and M is a perfect matching; or $|G|$ is odd and there is only one M-unsaturated vertex in G. In this section, we present a parallel algorithm for the MM problem for cographs. First we prove a technical lemma.

A graph $G=(V,E)$ is called a *complete k-partite graph* if V can be partitioned into k disjoint subsets $V_1,\ldots,V_k$ such that no two vertices in the same V_i are adjacent and for any $u \in V_i$ and $v \in V_j$ with $i \neq j$, $(u,v) \in E$. We will always assume $|V_1| \geq |V_2| \geq, \ldots, \geq |V_k|$.

Lemma 12: Let $G=(V,E)$ be a complete k-partite graph with partition $V_1,\ldots,V_k$. Let $n_i = |V_i|$ $(1 \leq i \leq k)$ and $n = \sum_{i=1}^{k} n_i$. Suppose $n_i \geq n_{i+1}$ for $1 \leq i < k$. If $n_1 \leq n - n_1$, then G has a nearly perfect matching M (if n is odd, the M-unsaturated vertex can be taken from V_1). Moreover, M can be constructed in $O(\log n)$ time with $O(n)$ processors.

Proof: If n is odd, we delete a vertex in V_1 from G. It's easy to show the conditions of the lemma still hold. So we assume n is even and construct a perfect matching M of G in three stages.

Stage 1: Define $s_j = \sum_{i=j}^{k} n_i$ (for $1 \leq j \leq k$) and $s_{k+1} = 0$. Define $t_j = \sum_{i=1}^{j-1}(n_i - n_j)$ (for $1 \leq j \leq k$) and $t_{k+1} = n$. In Fig 5a, the ith vertical line represents the set of the vertices in V_i. t_j is the number of vertices in the region T_j and s_j is the number of vertices in the region S_j.

Figure 5

The values of t_j's increase from $t_1 = 0$ to $t_{k+1} = n$. The values of s_j's decrease from $s_1 = n$ to $s_{k+1} = 0$. Let j be the largest index such that $t_j < s_j$. (Hence $t_{j+1} \geq s_{j+1}$). Note that $n_1 \leq n - n_1 = n_2 + \ldots + n_k$ implies $n_1 - n_2 < n_2 + \ldots + n_k$. Thus $t_2 < s_2$ and $j \geq 2$.

Case 1a: $t_j \leq s_{j+1}$. Consider Fig 5b. t_j is the number of vertices in the region T_j. s_{j+1} is the number of vertices in the region S_{j+1}. Let d be the number of vertices in the region D. Since $t_j \leq s_{j+1} \leq t_{j+1} = t_j + d$, we can match all vertices in T_j with some vertices in S_{j+1}. The remaining vertices in S_{j+1} can be matched to some vertices in D in such a way that the numbers of unmatched vertices in V_i's $(1 \leq i \leq j)$ differ by at most 1.

Case 1b: $t_j > s_{j+1}$. Consider Fig 5c. t_j is the number of vertices in T_j. s_{j+1} is the number of vertices in S_{j+1}. Let b be the number of vertices in the region B. Since $n_j \leq b$ and $s_{j+1} < t_j$, we have $s_j = n_j + s_{j+1} < t_j + b$. Thus all vertices in $V_j, \ldots, V_k$ can be matched with some vertices in $V_1, \ldots, V_{j-1}$ in such a way that the numbers of unmatched vertices in $V_1, \ldots, V_{j-1}$ differ by at most 1.

Stage 2: We proceed to match the remaining unmatched vertices. We now have sets $V_1, \ldots, V_j$ and some integers l and h such that $|V_i| = h$ for $1 \leq i \leq l$ and $|V_i| = h - 1$ for $l < i \leq j$.

Case 2a: l is even. In this case, take one vertex from each of $V_1, \ldots, V_l$. These vertices can be matched in pairs.

Case 2b: l is odd and j is even. In this case, $|\cup_{i=1}^{j} V_i| = (h-1) \times j + l$ is an odd number. This is impossible.

Case 2c: l is odd and j is odd. Take one vertex from each of V_i for $2 \leq i \leq l$. These vertices can be matched in pairs. Take one vertex from V_1 and match it with a vertex in V_j. Finally, take one vertex from each of V_i $(1 \leq i \leq j-1)$. These vertices can be matched in pairs.

Stage 3: Now, we have j sets $V_1, \ldots, V_j$ of equal size.

Case 3a: $j = 2p$ is even, we can match the vertices in V_{2i-1} with the vertices in V_{2i} for $1 \leq i \leq p$.

Case 3b: j is odd. If $j > 3$, match the vertices in $V_4, \ldots, V_j$ as in case 3a. Now we have three sets V_1, V_2, V_3 each having h vertices. h must be even. We can easily match all vertices in V_1, V_2, V_3.

In order to implement this algorithm, note that the numbers s_i's and t_i's can be computed in $O(\log n)$ time with $O(n)$ processors by using the post- and pre-sum algorithms. After s_i's and t_i's are known, the stages 1, 2, 3 can be implemented in $O(1)$ time with $O(n)$ processors. □

We now turn to the maximum matching algorithm for cographs. Let $G = (V, E)$ be a cograph with cotree T. For each node x of T, let $G(x)$ be the subgraph of G induced by the vertex set $L(T(x))$. (Recall that $L(T(x))$ denotes the set of leaf nodes in the subtree $T(x)$ of T rooted at x.) Let $n(x) = |L(T(x))|$ be the number of vertices in $G(x)$. Define $\bar{n}(x) = n(p_T(x)) - n(x)$ to be the number of vertices in $G(p_T(x))$ but not in $G(x)$. Let $M(x)$ denote a maximum matching of $G(x)$ and $D(x)$ denote the set of the vertices in $G(x)$ unmatched by $M(x)$. Define $\rho(x) = |D(x)|$. Given

T, $n(x)$'s for all nodes x of T can be computed in $O(\log n)$ time with $O(n)$ processors be using tree contraction method. Then $\bar{n}(x)$ can be computed in $O(1)$ time with $O(n)$ processors. For each internal node α with children $\beta_1, \ldots, \beta_k$, rename the indices so that $n(\beta_1) \geq n(\beta_2) \geq, \ldots, \geq n(\beta_k)$.

We will compute $\rho(x)$, $D(x)$ and $M(x)$ for each node x of T. The algorithm is a bottom-up computation on T. For each leaf node x of T, since $G(x)$ consists of only one vertex we have $\rho(x) = 1$, $M(x) = \emptyset$ and $D(x) = \{x\}$. Consider a node α of T with children $\beta_1, \ldots, \beta_k$. Let V_i be the vertex set of $G(\beta_i)$. Suppose $\rho(\beta_i)$, $D(\beta_i)$ and $M(\beta_i)$ have been computed for each $1 \leq i \leq k$. We describe how to compute $\rho(\alpha)$, $D(\alpha)$ and $M(\alpha)$.

Case 1: α is a 0-node. For any $1 \leq i, j \leq k$, $G(\beta_i)$ and $G(\beta_j)$ are disconnected. So $M(\alpha) = \bigcup_{i=1}^{k} M(\beta_i)$ is an MM of $G(\alpha)$. Thus $\rho(\alpha) = \sum_{i=1}^{k} \rho(\beta_i)$ and $D(\alpha) = \cup_{i=1}^{k} D(\beta_i)$.

Case 2: α is a 1-node. For any $1 \leq i, j \leq k$, $G(\beta_i)$ and $G(\beta_j)$ are completely connected. Namely for any vertex u in $G(\beta_i)$ and v in $G(\beta_j)$, $(u, v) \in E$.

Case 2a: $n(\beta_1) \leq \bar{n}(\beta_1)$. $G(\alpha)$ contains a complete k-partite subgraph on $V_1, \ldots, V_k$ which satisfies the conditions of Lemma 12. So if $n(\alpha)$ is even, then $G(\alpha)$ has a perfect matching and we set $\rho(\alpha) = 0$ and $D(\alpha) = \emptyset$. If $n(\alpha)$ is odd, we set $\rho(\alpha) = 1$ and $D(\alpha) = \{v\}$ where v is any vertex in $G(\beta_1)$. The MM $M(\alpha)$ of $G(\alpha)$ is computed via Lemma 12. It is completely determined from the integers $n(\beta_1), \ldots, n(\beta_k)$ and is independent from $M(\beta_i)$'s.

Case 2b: $n(\beta_1) > \bar{n}(\beta_1)$ and $\rho(\beta_1) \leq \bar{n}(\beta_1)$. If $\rho(\beta_1) \geq n(\beta_2)$, let $t = 0$. Otherwise let $t = \lceil (n(\beta_2) - \rho(\beta_1))/2 \rceil$. Let U' be a set of $2t$ dummy vertices. Consider the complete k-partite graph G' on the sets $V_1' = D(\beta_1) \cup U'$, $V_2, \ldots, V_k$. G' satisfies the conditions of Lemma 12. Thus G' has a nearly perfect matching M'. Consider an MM $M(\beta_1)$ of $G(\beta_1)$. ($M(\beta_1)$ is a perfect matching of the subgraph induced by the vertex set $V_1 - D(\beta_1)$.) Let M_1 be a subset of $M(\beta_1)$ containing t edges. Let U be the set of the $2t$ vertices in V_1 incident to the edges in M_1. For each $v \in V_i$ $(2 \leq i)$ that was matched by M' to a dummy vertex $u' \in U'$, rematch v to a vertex $u \in U$. Let M'' be the matching obtained from M' in this way. Then $M(\alpha) = M(\beta_1) - M_1 \cup M''$ is a nearly perfect matching of $G(\alpha)$. So if $n(\alpha)$ is even, we set $\rho(\alpha) = 0$ and $D(\alpha) = \emptyset$. If $n(\alpha)$ is odd, we set $\rho(\alpha) = 1$ and $D(\alpha) = \{v\}$ where v is any vertex in $D(\beta_1)$. In this case, $M(\alpha)$ only depends on $M(\beta_1)$ and is independent from $M(\beta_i)$ for $i \geq 2$.

Case 2c: $\rho(\beta_1) > \bar{n}(\beta_1)$. Construct a matching $M(\alpha)$ of $G(\alpha)$ as follows. $M(\alpha)$ contains $M(\beta_1)$. For each $x \in V_j$ $(2 \leq j \leq k)$, x is matched with a vertex $y \in D(\beta_1)$ and the edge (x, y) is included in $M(\alpha)$. Since $|D(\beta_1)| = \rho(\beta_1) > \bar{n}(\beta_1) = \sum_{i=2}^{k} |V_i|$, all vertices in V_j $(2 \leq j \leq k)$ can be matched this way. We show $M(\alpha)$ is an MM of $G(\alpha)$. Let M' is be a maximum matching of

$G(\alpha)$. Let $M_1' = \{(u,v) \in M' \mid u \in V_1, v \in V_1\}$ and $M_2' = M' - M_1'$. Since $M(\beta_1)$ is an MM of $G(\beta_1)$ and M_1' is a matching of $G(\beta_1)$, we have $|M_1'| \leq |M(\beta_1)|$. Clearly $|M_2'| \leq \sum_{i=2}^{k} |V_i|$. Thus $|M'| = |M_1'| + |M_2'| \leq |M(\beta_1)| + \sum_{i=2}^{k} |V_i| = |M(\alpha)|$. Therefore $M(\alpha)$ is an MM of G. So $\rho(\alpha) = \rho(\beta_1) - \bar{n}(\beta_1)$ and $D(\alpha)$ is a subset of $D(\beta_1)$ containing arbitrary $\rho(\alpha)$ vertices. Note that $M(\alpha)$ only depends on $M(\beta_1)$ and is independent from $M(\beta_i)$ for $i \geq 2$.

From these discussions, the value of $\rho(\alpha)$ for all nodes α of T can be computed in $O(\log n)$ time with $O(n)$ processors by using the tree contraction algorithm. After $\rho(\alpha)$'s have been computed, the sets $D(\alpha)$'s for all nodes α of T can be computed in $O(\log n)$ time with $O(n)$ processors by using the tree contraction algorithm again [1].

Next we describe how to construct an MM M of G. Let α be a 1-node of T. If the case 2a (2b or 2c, respectively) applies to α, then α is called a type 2a (2b or 2c, respectively) node. Let $P(\alpha)$ be the path in T from α to the root r of T. Suppose $P(\alpha) = \{\alpha, \gamma_q, \delta_q, \gamma_{q-1}, \delta_{q-1}, \ldots, \gamma_1, \delta_1\}$, where γ_i's are 0-nodes and δ_i's are 1-nodes and $\delta_1 = r$. α is called an *active* node iff: (1) γ_i is the first child of δ_i for $1 \leq i \leq q$; and (2) no δ_i in $P(\alpha)$ is a type 2a node. Fig 6 shows a cotree T and the active nodes of T. The type of each 1-node is indicated near the node. The heavy cycles indicate the active nodes. Note the nodes labeled by α and β. Since they are type 2a nodes, no other nodes in the subtrees $T(\alpha)$ and $T(\beta)$ are active. All active nodes of T can be identified in $O(\log n)$ time with $O(n)$ processors by using tree contraction algorithm [1].

Figure 6

Each active node α is charged to construct a subset of M. Let α be an active node and let $\beta_1, \ldots, \beta_k$ be its children. Let V_i be the vertex set of $G(\beta_i)$.

If α is a type 2a node, α constructs a nearly perfect matching in the complete k-partite graph induced by $V_1, \ldots, V_k$.

If α is a type 2b node, α constructs a nearly perfect matching in the complete k-partite graph induced by the sets $V_1' = D(\beta_1) \cup U', V_2, \ldots, V_k$, where $D(\beta_1)$ has been computed and U' is a set of $2t$ dummy vertices as discussed in the case 2b.

If α is a type 2c node, α constructs a matching that matches each vertex in $V_2, \ldots, V_k$ to a vertex in $D(\beta_1) - D(\alpha)$.

Since there are $O(n)$ processors, we can assign each active node α enough processors so that its task can be carried out in $O(\log n)$ time via Lemma 12. After this is done, some edges in the matchings constructed by the type 2b active nodes have to be adjusted in order to eliminate the

dummy vertices. This can be done in $O(\log n)$ time with $O(n)$ processors by using the tree contraction algorithm again. In summary, we have:

Theorem 13: If the cotree is given, a maximum matching of a cograph can be constructed in $O(\log n)$ time with $O(n)$ processors.

7. Permutation Representation

A graph $G = (V, E)$ is a *permutation graph* if V can be labeled by $\{1, \ldots, n\}$ and there exists a permutation $\pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ such that $(i, j) \in E$ iff $(i - j)(\pi(i) - \pi(j)) < 0$. π is called a *permutation representation* (PR) of G. Cographs are a proper subset of permutation graphs [12]. A parallel algorithm for recognizing permutation graphs and constructing their PR was developed in [13]. It takes $O(\log^2 n)$ time with $O(n^4)$ processors. In this section, we show a PR of a cograph can be found in $O(\log n)$ time with $O(n)$ processors if the cotree is given. Similar method was used in [21] for finding the two dimension total order representation of transitive series parallel graphs.

We first describe an interpretation of the PR. Consider an $n \times n$ grid. Let $\pi = \{\pi(1), \ldots, \pi(n)\}$ be a PR of a permutation graph $G = (V, E)$. For each i $(1 \leq i \leq n)$, plot the vertex i at the grid point $P(i) = (i, \pi(i))$. We will say i is the x-coordinate and $\pi(i)$ is the y-coordinate of the vertex i. Then $(i, j) \in E$ iff either $P(i)$ is located to the north-west of $P(j)$ or $P(j)$ is located to the north-west of $P(i)$. If we can plot the vertices of G on the grid with this property, the PR of G can be easily obtained.

Let $G = (V, E)$ be a cograph with cotree T. We plot the vertices of G on an $n \times n$ grid. For each node α of T, the vertices of $G(\alpha)$ are plotted on an $n(\alpha) \times n(\alpha)$ square region $R(\alpha)$ on the grid, where $n(\alpha)$ is the number of vertices in $G(\alpha)$. We represent $R(\alpha)$ by $[x_1(\alpha), y_1(\alpha), x_2(\alpha), y_2(\alpha)]$ where $(x_1(\alpha), y_1(\alpha))$ is the lower-left corner and $(x_2(\alpha), y_2(\alpha))$ is the upper-right corner of $R(\alpha)$.

The algorithm is a top-down computation on T. For the root r of T, since we will plot the graph $G = G(r)$ on the entire grid, let $R(r) = [1, 1, n, n]$. Consider an internal node α with children $\beta_1, \ldots, \beta_k$. Suppose $R(\alpha) = [x_1(\alpha), y_1(\alpha), x_2(\alpha), y_2(\alpha)]$ has been computed, we describe how to compute $R(\beta_i) = [x_1(\beta_i), y_1(\beta_i), x_2(\beta_i), y_2(\beta_i)]$ for each $1 \leq i \leq k$.

Case 1: α is a 0-node. Since the subgraphs $G(\beta_i)$'s are pairwise disconnected, the squares $R(\beta_i)$'s are placed on the grid running from the south-west to the north-east (Fig 7a). Therefore $R(\beta_i)$ can be computed by using the following formulas:

$x_1(\beta_i) = x_1(\alpha) + \sum_{j=1}^{i-1} n(\beta_j)$; $\quad y_1(\beta_i) = y_1(\alpha) + \sum_{j=1}^{i-1} n(\beta_j)$;

$x_2(\beta_i) = x_1(\alpha) + \sum_{j=1}^{i} n(\beta_j) - 1$; $\quad y_2(\beta_i) = y_1(\alpha) + \sum_{j=1}^{i} n(\beta_j) - 1$.

Fig 7

Case 2: α is a 1-node. Since the subgraphs $G(\beta_i)$'s are completely connected to each other, the squares $R(\beta_i)$'s are placed on the grid running from the north-west to the south-east (Fig 7b). Therefore $R(\beta_i)$ can be computed by using the following formulas:

$x_1(\beta_i) = x_1(\alpha) + \sum_{j=1}^{i-1} n(\beta_j)$; $\quad y_1(\beta_i) = y_1(\alpha) + \sum_{j=i+1}^{k} n(\beta_j)$;

$x_2(\beta_i) = x_1(\alpha) + \sum_{j=1}^{i} n(\beta_j) - 1$; $\quad y_2(\beta_i) = y_1(\alpha) + \sum_{j=i}^{k} n(\beta_j) - 1$.

This gives the following algorithm for constructing a PR for cographs.

Algorithm 3: Permutation representation

(1) Compute $n(\alpha)$ for each node α of T.

(2) For each internal node α with children $\beta_1, \ldots, \beta_k$, calculate the pre-sum and the post-sum of the sequence $\{n(\beta_1), \ldots, n(\beta_k)\}$.

(3) Assign the root r of T the value $R(r) = [1, 1, n, n]$. Assign the internal nodes of T the operators as defined above. Perform the top-down tree contraction algorithm [1] to T. Each leaf node u of T receives a value $R(u) = [x_1(u), y_1(u), x_2(u), y_2(u)]$. Note that $x_1(u) = x_2(u)$ and $y_1(u) = y_2(u)$. This means the vertex u is plotted at the grid point $(x_1(u), y_1(u))$.

(4) Define a permutation π such that $\pi(x_1(u)) = y_1(u)$ for each $u \in V$. π is a PR of G.

End.

Fig 8

Fig 8 shows a cograph G, its cotree T, the value $R(\alpha)$ for each node α of T, the grid drawing of G and the PR π. Since pre-sum, post-sum and tree contraction can be done in $O(\log n)$ time with $O(n)$ processors [1], we have:

Theorem 14: Given the cotree as input, the permutation representation of a cograph can be constructed in $O(\log n)$ time with $O(n)$ processors.

8. Depth First Spanning Tree

Let $G = (V, E)$ be a connected graph and H be a spanning tree of G rooted at $r \in V$. A non-tree edge $(u, v) \in E$ is called a *cross edge* if u and v are not related in H. H is a *depth first spanning tree* (DFST) of G if there exist no cross edges. The DFST problem is as follows: Given G and a vertex $r \in V$, construct a DFST of G rooted at r. We show DFST problem can be solved efficiently for permutation graphs, assuming the permutation representation is given.

Let G be a permutation graph. Consider the grid drawing R of G. Let $l = \lfloor n/2 \rfloor$. Draw a horizontal line with y-coordinate $l + 0.5$ and a vertical line with x-coordinate $l + 0.5$. These two lines divide R into four regions R_1, R_2, R_3, R_4 (Fig 9). Let n_i $(1 \leq i \leq 4)$ be the number of vertices in

R_i. The following simple observation is crucial to our algorithm: Because $n_1 + n_2 = n_1 + n_3 = l$, we must have $n_2 = n_3$. Since each vertex in R_2 is adjacent to every vertex in R_3, we can construct a cycle C consisting of all vertices in R_2 and R_3, where the vertices of C alternate between R_2 and R_3. Moreover if C is removed from G, the graph becomes disconnected and each connected component has at most $n/2$ vertices (since each of them consists of vertices either in R_1 or R_4).

Fig 9

The following algorithm solves the DFST problem for permutation graphs.

Algorithm 4: Depth First Spanning Tree

(1) Construct the cycle C consisting of the vertices in R_2 and R_3 as described above.

(2) Find a path P_1 from the specified root r to a vertex z on C. (If r is on C then $z = r$ and $P_1 = \emptyset$.) Let x be a vertex such that (x, z) is an edge of C. Let $P = P_1 \cup (C - \{(x, z)\})$. P is a path of G. Number the vertices of P as $\{p_1, p_2, \ldots, p_t\}$ starting at $r = p_1$.

(3) Let $G' = G - P$. Compute the connected components $G_1, \ldots, G_s$ of G'.

(4) For each G_i $(1 \leq i \leq s)$, let p_{k_i} be the vertex of P with the largest index such that p_{k_i} is adjacent to a vertex q_i in G_i.

(5) For each i $(1 \leq i \leq s)$, recurse on G_i to find a DFST T_i of G_i rooted at q_i.

(6) Return the tree $T = P \cup (\cup_{i=1}^{s} T_i) \cup (\cup_{i=1}^{s} \{(q_i, p_{k_i})\})$.

End.

Fig 10

Fig 10 illustrates the construction of the Algorithm 4. It is easy to verify that the tree T constructed in Step 6 has no cross edges and hence is a DFST of G. We next discuss the implementation of Algorithm 4. Let $\pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$ be a permutation representation of G.

Step 1: Clearly, $R_2 = \{i \mid 1 \leq i \leq l \text{ and } \pi(i) > l\}$ and $R_3 = \{i \mid l < i \leq n \text{ and } \pi(i) \leq l\}$, where $l = \lfloor n/2 \rfloor$. Both R_2 and R_3 can be identified from π in $O(1)$ time with $O(n)$ processors. Suppose $R_2 = \{u_1, \ldots, u_s\}$ where the x-coordinate of u_i's is increasing. Suppose $R_3 = \{v_1, \ldots, v_s\}$ where the x-coordinate of v_j's is increasing. $C = \{u_1, v_1, u_2, v_2, \ldots, u_s, v_s\}$ can be constructed in $O(1)$ time with $O(n)$ processors.

Step 2: We assume the root r is in the region R_1. Other cases are similar. By using the algorithm in [7], we can find a spanning tree T' of G rooted at r in $O(\log n)$ time with $O(n)$ processors. Compute the distance of each vertex x to r in T'. This takes $O(\log n)$ time with $O(n)$ processors. Let z be a vertex of C whose distance to r is minimum among all vertices of C. Suppose z is in R_2 (the

other case is similar). Let y be the parent of z in T'. Let P_1 be the path in T' from r to y. Clearly $(y,z) \in E$ implies $(y,u_1) \in E$. Thus the vertices of the path P can be numbered in the order $r = p_1, p_2, \ldots, p_l = y, u_1, v_1, u_2, v_2, \ldots, u_s, v_s$. This can be done in $O(\log n)$ time with $O(n)$ processors. For each $x \in P$, let $index(x)$ denote the index of x in this numbering.

Step 3: Delete the entries in π corresponding to the vertices of P. Renumber the remining entries in π so that it is a permutation $\pi' : \{1, \ldots, |G'|\} \rightarrow \{1, \ldots, |G'|\}$. π' is a permutation representation of G'. Using π', the connected components of G' and the permutation representation of each connected component can be found in $O(\log n)$ time with $O(n)$ processors [7].

Step 4: We need to find the vertex p_{k_i} and q_i for each connected component G_i of G'. For each vertex $z \in V - P$, define $buddy(z)$ to be the vertex in P that is adjacent to z and has maximum index. We describe the computation of $buddy(z)$ for $z \in R1$. The case $z \in R_4$ is similar.

Let $buddy_1(z)$ denote the neighbor of z that is in the path P_1 and has maximum index. Let $buddy_2(z)$ ($buddy_3(z)$, respectively) denote the neighbor of z that is in R_2 (R_3, respectively) and has maximum index. In order to find $buddy(z)$, we compute $buddy_i(z)$ for $i = 1, 2, 3$ and pick the one with the largest index.

The computation of $buddy_2(z)$ is easy: Let j be the largest integer such that the x-coordinate of $u_j \in R_2$ is less than the x-coordinate of z. Then $buddy_2(z) = u_j$. This can be done in $O(\log n)$ time with one processor by binary search.

The computation of $budda_3(z)$ is slightly more complex. Consider a vertex $v_i \in R_3$. If there is a vertex $v_{i'} \in R_3$ such that $i < i'$ and the y-coordinate of v_i is larger than the y-coordinate of $v_{i'}$, then v_i cannot be $buddy(z)$ for any $z \in R_1$. This is because $(z, v_i) \in E$ implies $(z, v_{i'}) \in E$ and $index(v_i) < index(v_{i'})$. Let $W = \{w_1, \ldots, w_t\}$ be the subsequence of $\{v_1, \ldots, v_s\}$ such that for each w_i, no vertex in R_3 is located to the south-east of w_i. In order to find $buddy_3(z)$, we only need to consider the vertices in W. W is a sequence of vertices running from the south-west to the north-east in R_3 (Figure 11a). By using the post-sum algorithm on the sequence of the y-coordinates of $\{v_1, \ldots, v_s\}$ with "min" operator, W can be computed in $O(\log n)$ time with $O(n)$ processors. After this pre-processing is done, let i be the largest integer such that the y-coordinate of w_i is less than the y-coordinate of z. Then $buddy_3(z) = w_i$. This can be done in $O(\log n)$ time with one processor by binary search.

Figure 11

Next we consider the computation of $buddy_1(z)$. The path P_1 is obtained from the spanning tree T' constructed by using the algorithm in [7]. This path have the following property [7]: Let $P_1' =$

$\{p_1, p_3, \ldots\}$ be the subsequence of P_1 containing the vertices with odd indices. Let $P_1'' = \{p_2, p_4, \ldots\}$ be the subsequence of P_1 containing the vertices with even indices. Then P_1' is a sequence of vertices running from the south-west to the north-east. P_1'' is also a sequence of vertices running from the south-west to the north-east. The relative position of the vertices in P_1 is shown in Fig 11b [7]. By using the method described in the last two paragraphs, we can find the vertex in P_1' that is adjacent to z and has the maximum index and the vertex in P_1'' that is adjacent to z and has largest index. Then $buddy_1(z)$ is the one with larger index.

Thus we can find $buddy(z)$'s for all z in $O(\log n)$ time with $O(n)$ processors. Then for each connected component G_i, the vertex q_i and p_{k_i} can be found in $O(\log n)$ time with $O(n)$ processors.

Step 5 is recursive call. Step 6 takes $O(1)$ time with $O(n)$ processors. This completes the discussion of the implementation. Since each iteration of the algorithm takes $O(\log n)$ time with $O(n)$ processors and the recursion depth is $O(\log n)$, we have:

Theorem 15: Let G be a permutation graph given by the permutation representation. The depth first spanning tree of G can be constructed in $O(\log^2 n)$ time with $O(n)$ processors.

Acknowledgements: In an earlier version of this paper, the cotree recognition algorithm takes $O(\log^2 n)$ time with $O(n^2)$ processors. The depth-first spanning tree algorithm for permutation graphs takes $O(\log^2 n)$ time with $O(n+m)$ processors. Mark Novick pointed out to the author that the first algorithm can be implemented using $O(n+m)$ processors and the second algorithm can be implemented using $O(n)$ processors without increasing the time bound. Mark Novick, S-T. Peng and a anonymous referee pointed out a technical error in the maximum matching algorithm. The author is grateful to them for their very helpful comments.

References

[1] K. Abrahamson, N. Dadoun, D. G. Kirkpatrick and T. Przytycka, A Simple Parallel tree Contraction Algorithm, J. Algorithms 10 (2), 1989, pp. 287-302.

[2] G. S. Adhar and Shietung Peng, NC Algorithms for Complement Reducible Graphs and Parity Graphs, in Proc. of International Conference on Computing and Information, 1989.

[3] G. S. Adhar and Shietung Peng, Parallel Algorithms for Complement Reducible Graphs and Parity Graphs with Applications, in Workshop on Algorithms and Data Structures, Lecture Notes in Theoretical Computer Science, Springer-Verlag, New York, 1989.

[4] G. S. Adhar and Shietung Peng, Parallel Algorithms for Cographs and Parity Graphs with Applications, J. Algorithms 11, 1990, pp. 252-284.

[5] G. S. Adhar and Shietung Peng, Parallel Algorithms for Finding Minimal Path Cover, Hamiltonian Path and Hamiltonian Cycle in Cographs, Technical Report, UMIACSi-TR-89-64, University of Maryland, 1989.

[6] G. S. Adhar and Shietung Peng, A Parallel Algorithm for Maximum Matching in Cographs, Congressus Numerantium 77, 1990, pp. 217-227.

[7] K. Arvind, V. Kamakoti and C. Pandu Rangan, Efficient Algorithms for Permutation Graphs, Indian Institute of Technology, manuscript. To appear.

[8] R. Cole, Parallel Merge Sort, Proc. 27th IEEE Symposium on Foundation of Computer Science, 1986, pp. 511-517.

[9] D. G. Corneil, H. Lerchs and L. Stewart Burlinham, Complement Reducible Graphs, Disc. Appl. Math. 3, 1981, pp. 163-174.

[10] D. G. Corneil, Y. Perl and L. K. Stewart Burlinham, Cographs: Recognition, applications and algorithms, in Proc. 15th Southeastern Conference on Combinatorics, Graph Theory and Computing, Congressus Numerantium, Vol 43, 1984, pp. 249-259.

[11] D. G. Corneil, Y. Perl and L. K. Stewart Burlinham, A Linear Recognition Algorithm for Cographs, SIAM J. on Computing 14 (4), 1985, pp. 926-934.

[12] M. C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.

[13] D. Helmbold and E. Mayr, Perfect Graphs and Parallel Algorithms, in Proc. 1986 Intl. Conf. on Parallel Processing, pp. 853-860.

[14] H. A. Jung, On a Class of Posets and the Corresponding Comparability Graphs, J. Combinatorial Theory (B) 24, 1978, pp. 125-133.

[15] D. G. Kirkpatrick and T. Przytycka, Parallel Recognition of Complement Reducible Graphs and Cotree Construction, Technical Report 88-1, Department of Computer Science, University of British Columbia, 1988. To appear.

[16] Lin and S. Olariu, An NC Algorithm for Cographs, Department of Computer Science, Old Dominion University, Tech. Rep. TR-89-32.

[17] G. Miller and J. Reif, Parallel Tree Contraction and its Applications, Proc. IEEE Symposium on Foundation of Computer Science, 1985, pp. 478-489.

[18] M. Novick, Fast Parallel Algorithms for the Modular Decomposition, TR 89-1016, Department of Computer Science, Cornell University, 1989.

[19] Y. Shiloach and U. Vishkin, An $O(\log n)$ Parallel Connectivity Algorithm, J. Algorithms 3, 1982, pp. 57-67.

[20] D. P. Sumner, Dacey Graphs, J. Australian Math. Soc. 18 (4), 1974, pp. 492-502.

[21] J. Valdes, R. E. Tarjan and E. L. Lawler, The Recognition of Series Parallel Digraphs, SIAM J. on Computing 11 (2), 1982, pp. 298-313.

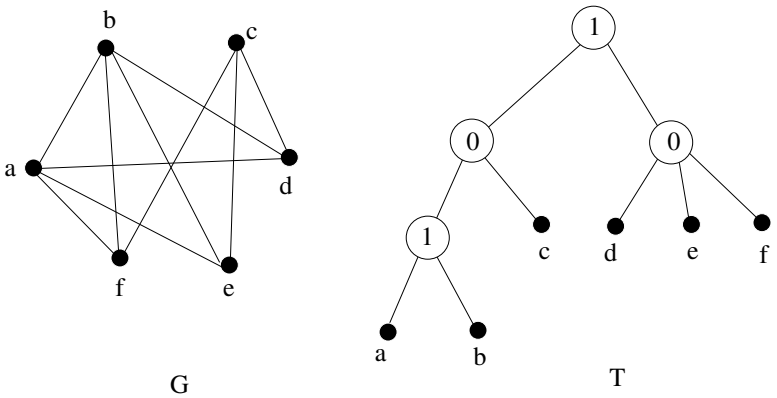


Figure 1. A cograph G and its cotree T

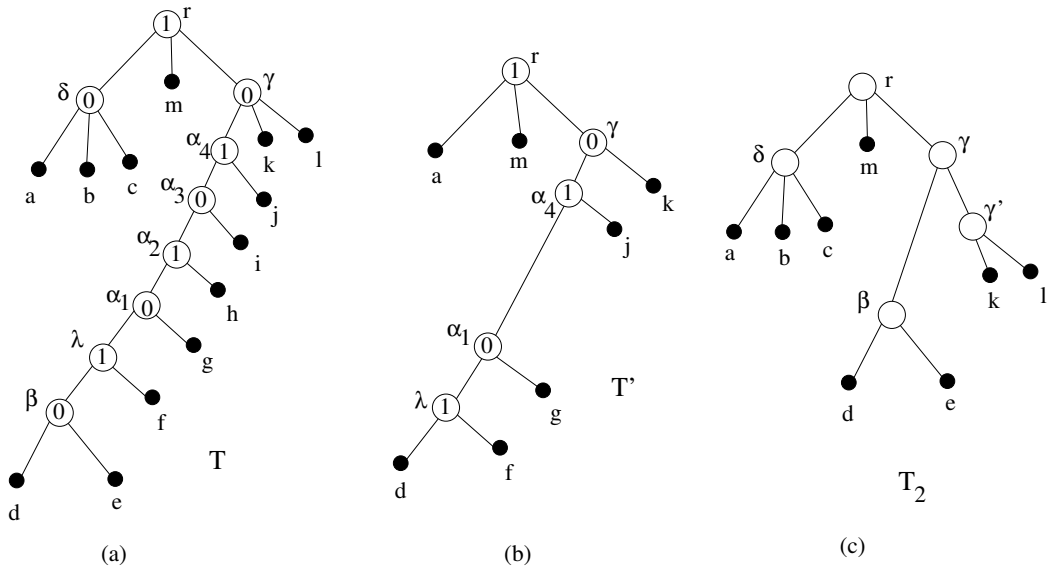


Figure 2. Trim-tree and Cut-tree operations

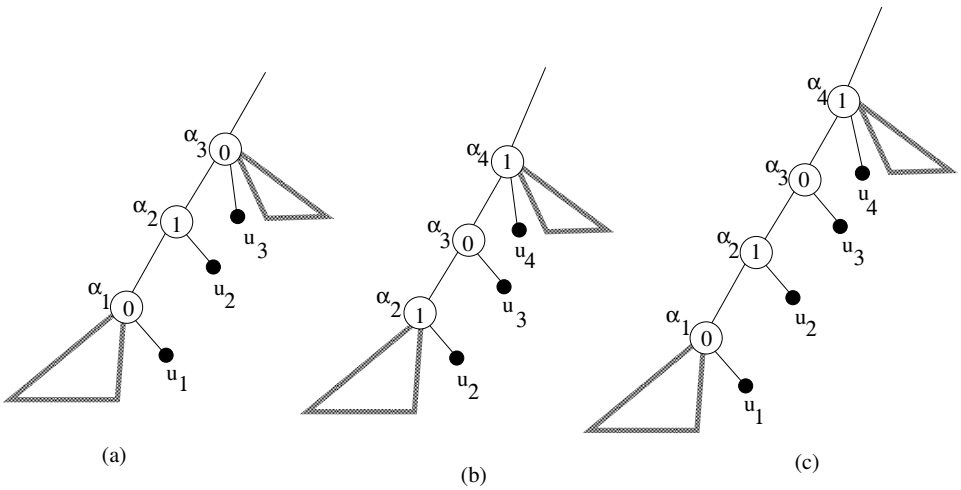


Figure 3. The section of a cotree corresponding to (a) false trio, (b) a true trio, and (c) 4-twin

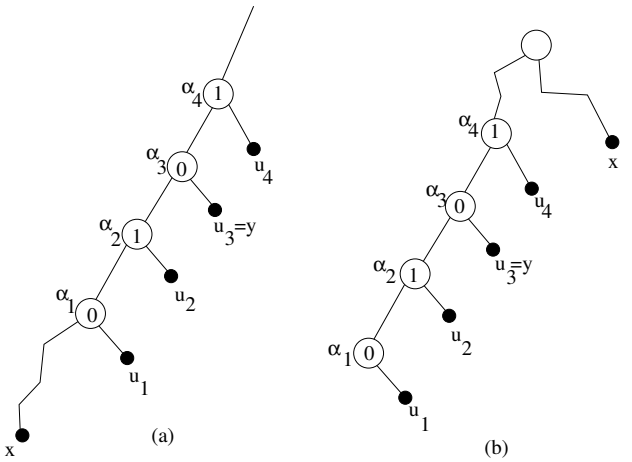


Figure 4. The proof of Lemma 7

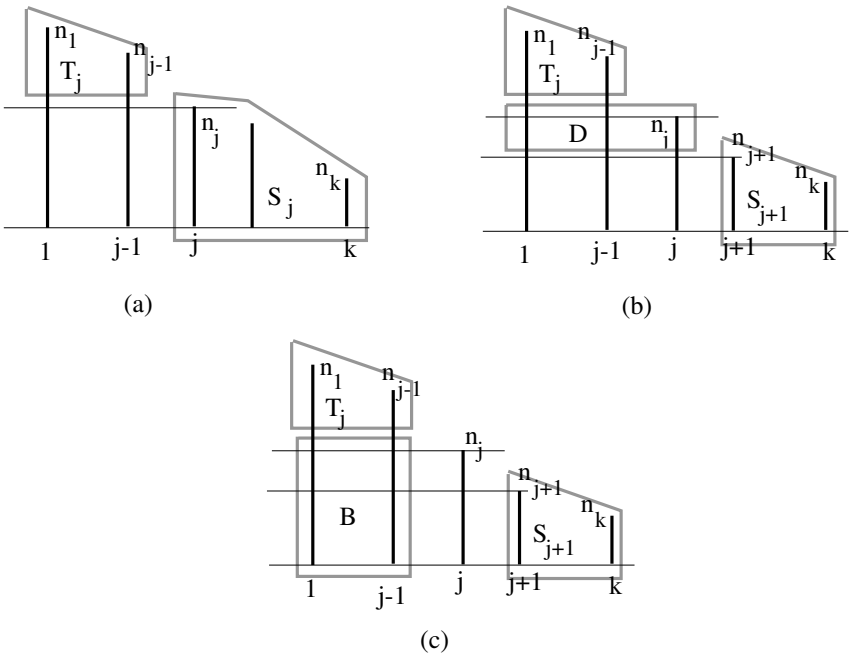


Figure 5. The proof of Lemma 12.

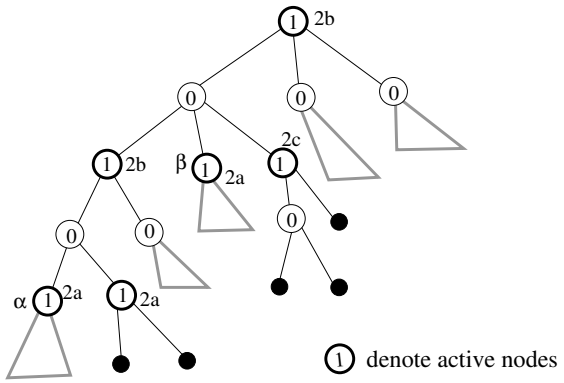


Figure 6. The active nodes in a cotree.

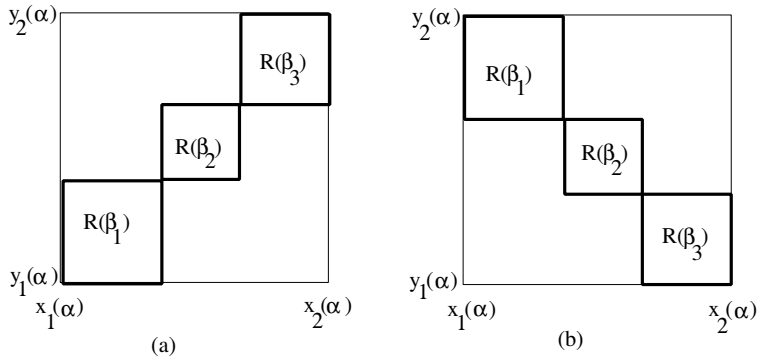


Figure 7. Grid Drawing of a graph

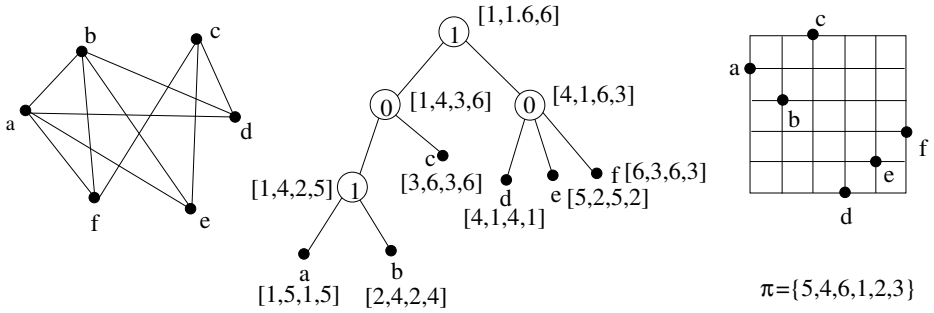


Figure 8. A cograph and its permutation representation.

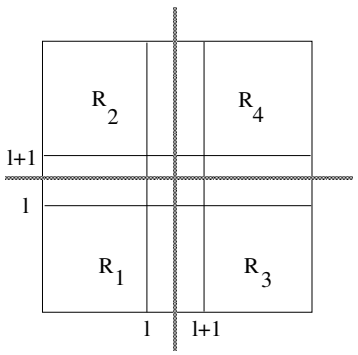


Figure 9. Divide a grid into four regions

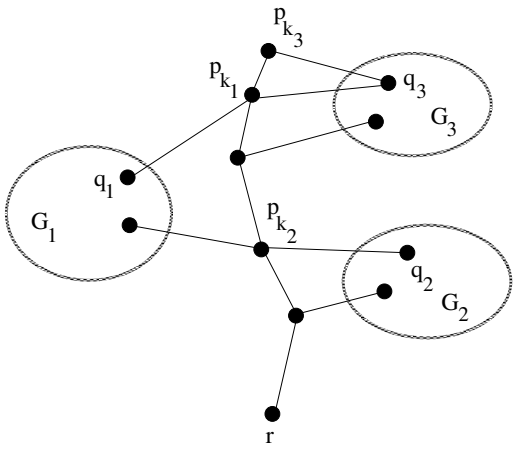
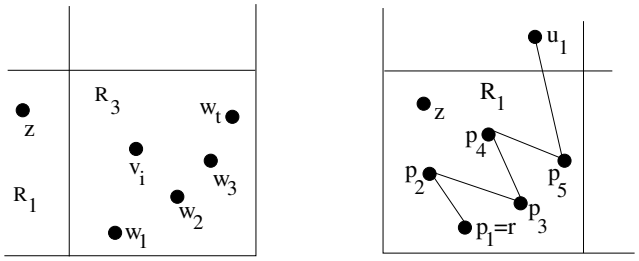


Figure 10. Constructing a depth first spanning tree



(a) (b)

Figure 11. Finding the vertex on P with the largest index.