A Time-Optimal Solution for the Path Cover Problem on Cographs *

K. Nakano† S. Olariu‡ A. Y. Zomaya§

Abstract

We show that the notoriously difficult problem of finding and reporting the smallest number of vertex-disjoint paths that cover the vertices of a graph can be solved time- and work-optimally for cographs. Our algorithm solves this problem in $O(\log n)$ time using $\frac{n}{\log n}$ processors on the EREW-PRAM for an n-vertex cograph G represented by its cotree.

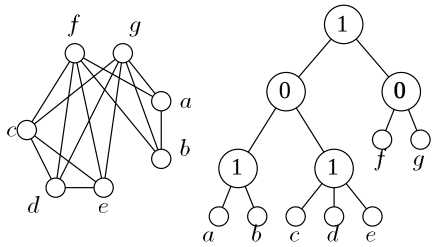

Figure 1. A cograph and the corresponding cotree.

1 Introduction

A graph-theoretic problem with a large number of practical applications is the *path cover problem*, which involves finding a minimum number of vertex-disjoint paths that together cover the vertices of a graph. A graph G that admits a path cover of size one is referred to as *Hamiltonian*. It is, therefore, clear that the path cover problem is at least as hard as the problem of deciding whether a graph G has a Hamiltonian path.

The class of *cographs*, or complement-reducible graphs, is defined recursively as follows: (1) A single-vertex graph is a cograph; (2) If $G = (V, E)$ is a cograph, then its complement $\overline{G} = (V, V \times V - E)$ is also a cograph; (3) If both $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ satisfying $V_1 \cap V_2 = \phi$ are cograph, then their union $G = (V_1 \cup V_2, E_1 \cup V_2)$ is also a cograph.

The cographs admit a tree representation unique up to isomorphism. Specifically, one can associate with every cograph $G = (V, E)$ a unique rooted tree $T(G)$ called the *cotree* of G featuring the following properties: (4) Every internal node of $T(G)$ has at least two children; (5) The internal nodes of $T(G)$ are labeled by either 0 (0-node) and 1 (1-node) in such a way that labels alternate along every path in $T(G)$ starting at the root; (6) Each leaf of $T(G)$ corresponds to a vertex in V, such that, $(x, y) \in E$ if and only if the lowest common ancestor of the leaves corresponding to x and y is a 1-node. We refer the reader to Figure 1 illustrating a cograph and its cotree. He [5] showed that the cotree of a cograph with n vertices and m edges can be built in $O((\log n)^2)$ time using $O(n + m)$ CRCW processors.

Lin *et al.* [7] showed that an instance of size n of the path cover problem for a cograph can be solved in $O(n)$ sequential time. Quite a while back Adhar and Peng [1] presented a parallel algorithm to find a minimum path cover, a Hamiltonian path, and a Hamiltonian cycle in n-vertex cographs. Their algorithm runs in $O(\log^2 n)$ time and using $O(n^2)$ processors on the CRCW. Surprisingly, the algorithm in [1] takes $O(\log^2 n)$ time and $O(n^2)$ processors on the CRCW even to determine whether a cograph contains a Hamiltonian path or cycle.

As a first step towards solving this open problem, Lin *et al.* [8] showed that one can determine the number of paths in a minimum path cover for cographs in $O(\log n)$ time and $O(n)$ work on the EREW. At the same time, Lin *et al.* [8] proposed an algorithm to report all the paths in a minimum path cover running in O($\log^2 n$) time, using $\frac{n}{\log n}$ processors on the EREW. Since, as shown in [7], a minimum path cover can be returned in $O(n)$ sequential time, the algorithm in [8] is suboptimal.

The main contribution of this work is to offer a time-

*This work was supported in part by NSF grant CCR-9522093, by ONR grant N00014-97-1-0526, and and by the Australian Research Council

†Department of Electrical and Computer Engineering, Nagoya Institute of Technology, Showa-ku, Nagoya 466-8555, JAPAN

‡Department of Computer Science, Old Dominion University, Norfolk, VA 23529, U.S.A.

§Department of Electrical and Electronic Engineering, The University of Western Australia, Perth, WA 6970, AUSTRALIA

and work-optimal solution to the path cover problem for cographs. Our algorithm runs in $O(\log n)$ time using $\frac{n}{\log n}$ processors on the EREW for an n-vertex cograph G represented by its cotree. Due to the page limitation, we omit the proof of the time- and work-optimality.

2 Finding a minimum path cover: a first look

We begin by reviewing the sequential algorithm of [7] for finding a minimum path cover of n-vertex cograph in $O(n)$ time, as well as the parallel algorithm of [8] for computing the number of paths in a minimum path cover in $O(\log n)$ time using $\frac{n}{\log n}$ processors on the EREW.

For convenience and ease of presentation we now show how to binarize the cotree $T(G)$ corresponding to a cograph G, in such a way that each of its internal nodes has exactly two children [8]. Let u be an internal node with children $v_1, v_2, \ldots, v_k$, ($k \geq 3$). We replace node u by $k-1$ nodes $u_1, u_2, \ldots u_{k-1}$ such that u_1 has children v_1 and v_2, and each u_i, ($2 \leq i \leq k$), has children u_{i-1} and v_i. We shall refer to the binarized version of $T(G)$ as $T_b(G)$ and note that $T_b(G)$ satisfies properties (4) and (6) above.

For an internal node u of $T_b(G)$ its *left* and *right* children will be denoted by v and w, respectively. Let $G(u)$ denote the subgraph of G induced by the leaf descendants of u in $T_b(G)$. Further, let $L(u)$ denote the *number* of leaf descendants of u in $T_b(G)$, that is, the number of vertices of $G(u)$. Let $p(u)$ denote the number of paths in a minimum path cover of $G(u)$. We say that $T_b(G)$ is *leftist*, if for every internal node u, the condition $L(v) \geq L(w)$ is satisfied. Let $T_{bl}(G)$ denote the leftist binarized cotree of G.

We now review the ideas in [7] for finding a minimum path cover of a cograph G, given its leftist binarized cotree $T_{bl}(G)$. Suppose that the minimum path covers of $G(v)$ and $G(w)$ have already been obtained. If u is 0-node, then no edge in $G(u)$ connects vertices from $G(v)$ and $G(w)$. Thus, a minimum path cover for G is just the union of minimum path covers for $G(v)$ and $G(w)$.

If u is 1-node, recall that every vertex in $G(v)$ is adjacent to all the vertices in $G(w)$. Referring to Figure 2, we distinguish the following two cases.

Case 1, $p(v) > L(w)$: We use the $L(w)$ vertices in $G(w)$ to bridge $L(w)+1$ of the paths in a minimum path cover of $G(v)$ into one path and the resulting minimum path cover has $p(v) - L(w)$ paths. In Figure 2, $L(w) = 2$ vertices bridge $p(v) = 4$ paths into $p(v) - L(w) = 2$ paths.

Case 2, $p(v) \leq L(w)$: In this case, $p(v)-1$ vertices in $G(w)$ are used to bridge the $p(v)$ paths in a minimum path cover of $G(v)$ into one path. These vertices are said to be *bridge* vertices. The remaining $L(w) - p(v) + 1$ vertices, called *insert* vertices, will be inserted into the path thus obtained. The resulting minimum path cover is a Hamiltonian path. In Figure 2, $p(v) - 1 = 3$ vertices are used to bridge $p(v) = 4$ paths into one path and $L(w) - p(v) + 1 = 4$ vertices are inserted into the path.

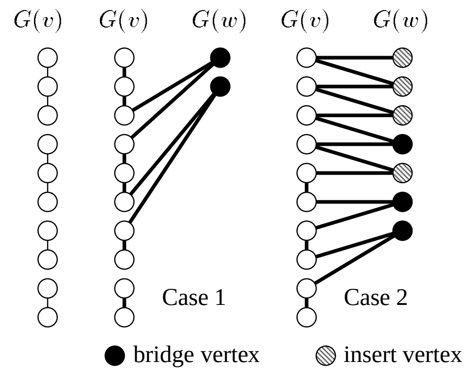

Figure 2. Illustrating Case 1 and Case 2.

We refer the reader to [7] for a detailed proof of the correctness of this approach. As it turns out, all the paths in a minimum path cover of G can be obtained by traversing $T_{bl}(G)$ in a bottom-up fashion from the leaves to the root. A careful implementation guarantees that the corresponding algorithm runs in time linear in the size of $T_{bl}(G)$. Thus, we have the following result.

Lemma 2.1 *[7] Given the cotree $T(G)$ of an n-vertex cograph G, a minimum path cover can be returned in $O(n)$ sequential time.*

Lin *et al.* [8] showed that the simpler problem of computing the *number* of paths in a minimum path cover can be computed in $O(\log n)$ time. The idea is as follows. From the construction of the minimum path cover, the number $p(u)$ of paths in the minimum path cover of $G(u)$ can be computed by the following formula:

$$\begin{aligned} p(u) &= p(v) + p(w) \quad \text{if } u \text{ is 0-node} \\ &= \max\{p(v) - L(w), 1\} \quad \text{if } u \text{ is 1-node} \end{aligned}$$

The well-known tree contraction technique [6] enables us to evaluate this formula for each internal node u. Using this idea, the following result was proved in [8].

Lemma 2.2 *[8] For every internal node u in $T_{bl}(G)$, the number $p(u)$ of paths in a minimum path cover of $G(u)$ can be computed in $O(\log n)$ time using $\frac{n}{\log n}$ EREW processors.*

We now further modify $T_{bl}(G)$. The vertices of the cograph G (i.e. leaves of $T_{bl}(G)$) will be partitioned into three categories as follows: **bridge vertex**: a vertex bridging paths at a 1-node; **insert vertex**: a vertex to be inserted in the path at a 1-node; **primary vertex**: a vertex neither bridging nor being inserted. Note that a primary vertex corresponds to a leaf of $T_{bl}(G)$ such that every internal node

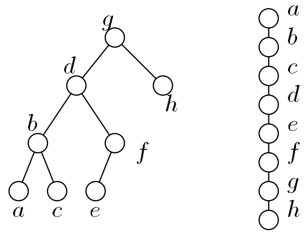

Figure 3. A path tree and the corresponding path.

along a path from the root to the leaf is not the right child of a 1-node. Conversely, a bridge or insert vertex belongs to a subtree rooted at an internal node that is the right child of a 1-node.

3 Finding a minimum path cover using path trees

The main goal of this section is to introduce *path trees* that will turn out to be key ingredients in our time- and work-optimal parallel algorithm for the path cover problem.

Let G be a cograph. A path tree is a rooted binary tree with each node of the path tree corresponding to a vertex of some path π in G. The path π is captured by the inorder traversal of the path tree. We refer the reader to Figure 3 illustrating a path tree and the corresponding path. Clearly, once a path tree is available it can be readily converted into the desired path by using the Euler tour technique. Multiple vertex-disjoint paths will be captured by disjoint collections of path trees.

Let u be an internal node of $T_{bl}(G)$ with left and right children v and w, respectively. We are interested in computing a path tree of $G(u)$ using those for $G(v)$ and $G(w)$. First, suppose that u is a 0-node and the path trees of $G(v)$ and $G(w)$ are already available. Since no edge in the graph connects edges from $G(v)$ and $G(w)$, the union of the path trees for $G(v)$ and $G(w)$ yields the path trees for $G(u)$.

Next, suppose that u is a 1-node and the path trees of $G(v)$ have already been computed. We consider the following two cases:

Case 1, $p(v) > L(w)$: The $L(w)$ vertices in $G(w)$ bridge $L(w)+1$ paths and the resulting minimum path cover of $G(u)$ has $p(v) - L(w)$ paths. To perform the corresponding operation on the path trees, we construct a binary tree having the $L(w)$ vertices in $G(w)$ as internal nodes and the roots of $L(w)+1$ path trees in $G(v)$ as leaves. This process is illustrated in Figure 4, where we construct a binary tree with vertices a, b, c having the roots of the path trees B, C, D, E

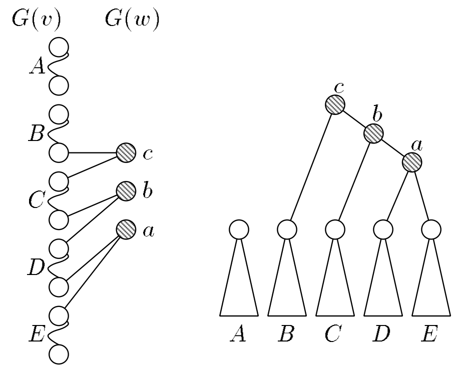

Figure 4. Construction for Case 1.

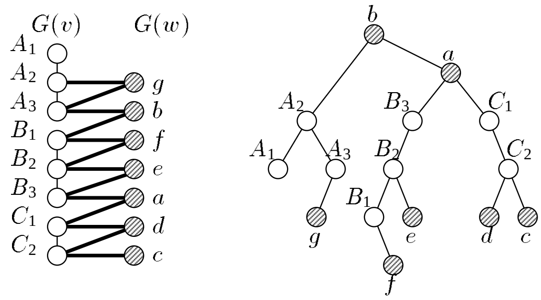

Figure 5. Construction for Case 2

as their children. The inorder traversal of the path tree thus obtained is

$$B \to c \to C \to b \to D \to a \to E,$$

which corresponds to the path we should obtain.

Case 2, $p(v) \leq L(w)$: We refer the reader to Figure 5 for an illustration of the construction of a path tree in Case 2. In this case, $p(v) - 1$ bridge vertices from $G(w)$ connect the roots of the path trees in a way similar to Case 1. In the figure, two vertices a and b connect three path trees. Each of the $L(w) - p(v) + 1$ insert vertices is connected to path trees as leaves. In the figure, five vertices $c, d, e, f,$ and g are connected to path trees as leaves.

Notice that in this process, a vertex of the original path trees with at most on child may end up with one (or two) insert vertices from $G(w)$ as leaves. However, not all such vertices can have a child. For example, C_1 cannot have an insert vertex as a left child; if vertex g were a left child of C_1, then a and g would be adjacent in the corresponding Hamiltonian path. However, it is not necessarily the case the a and g are adjacent in the underlying graph G. For the same reason, B_3 cannot have an insert vertex as a right child.

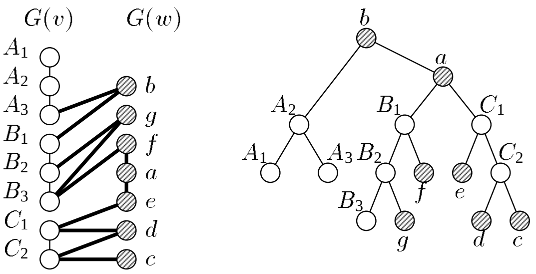

Figure 6. Illustrating a pseudo path tree

More generally, we may have *illegal* children as follows: Let T_1, T_2, ..., $T_{p(v)}$ denote the path trees placed left-to-right in this order such that their roots are connected by $p(v)-1$ bridge vertices.

1. The right child of the rightmost vertex (i.e. the vertex that appears last in the inorder traversal) of T_i, $(1 \le i \le p(v)-1)$. If an insert vertex is connected as the right child, then it must be adjacent to the bridge vertex that is the lowest common ancestor of T_i and T_{i+1}, a contradiction.

2. The left child of the leftmost vertex in T_i, $(2 \le i \le p(v))$, by a mirror argument.

For later reference, we introduce a *pseudo path tree*, which may have illegal insert vertices. Figure 6 illustrates a pseudo path tree: vertices e and f are illegal, and the corresponding path is *invalid* in the graph since the edges (f, a) and (a, e) are not present.

4 Constructing path trees using brackets

To begin, we demonstrate how pseudo path trees can be constructed efficiently. Once this is done, we can convert pseudo path trees to (correct) path trees. For constructing pseudo path trees efficiently, we will generate a sequence of brackets, each corresponding to a vertex in the pseudo path tree. We use two types of brackets: *square brackets* ("[" and "]") and *round brackets* ("(" and ")"). By finding matching pairs of square brackets and matching pairs of round brackets independently, we can construct pseudo path trees as follows.

Let u be a node of $T_{bl}(G)$ and let v and w denote, respectively, its left and right children if any. We associate with u a sequence $B(u)$ of brackets as follows: If u is a leaf corresponding to a primary vertex, then $B(u) = \overset{u^p}{[}\ \overset{u^l}{(}\ \overset{u^r}{(}$. If u is 0-node then $B(u) = B(v) \cdot B(w)$. If u is 1-node and $p(v) > L(w)$ (i.e. Case 1), then, $B(u) = B(v) \cdot \overset{s_1^r}{]}\ \overset{s_1^l}{]}\ \overset{s_1^p}{[}\ \overset{s_2^r}{]}\ \overset{s_2^l}{]}\ \overset{s_2^p}{[}\ \cdots\ \overset{s_{L(w)}^r}{]}\ \overset{s_{L(w)}^l}{]}\ \overset{s_{L(w)}^p}{[}$, where, s_i $(1 \le i \le L(w))$ denotes the bridge vertices of w. The matching pair $\overset{s_i^p}{[}$ and $\overset{s_{i+1}^r}{]}$ $(1 \le i \le L(w)-1)$ corresponds to an edge connecting the right child s_i and the parent s_{i+1}. Further, each of $\overset{s_1^l}{]}, \overset{s_2^l}{]}, \ldots, \overset{s_{L(w)}^l}{]}$ and $\overset{s_1^r}{]}$ matches a square bracket in $B(v)$, and corresponds to an edge connecting with a root of a path tree of $G(v)$.

If u is 1-node and $p(v) \le L(w)$ then, $B(u) = B(v) \cdot\ \overset{s_1^r}{]}\ \overset{s_1^l}{]}\ \overset{s_1^p}{[}\ \overset{s_2^r}{]}\ \overset{s_2^l}{]}\ \overset{s_2^p}{[}\ \cdots\ \overset{s_{p(v)-1}^r}{]}\ \overset{s_{p(v)-1}^l}{]}\ \overset{s_{p(v)-1}^p}{[}\ \overset{t_{p(v)}^p}{)}\ \overset{t_{p(v)+1}^p}{)}\ \cdots\ \overset{t_{L(w)}^p}{)}\ \overset{t_{p(v)}^l}{(}\ \overset{t_{p(v)}^r}{(}\ \overset{t_{p(v)+1}^l}{(}\ \overset{t_{p(v)+1}^r}{(}\ \cdots\ \overset{t_{L(w)}^l}{(}\ \overset{t_{L(w)}^r}{(}$, where, s_i $(1 \le i \le p(v)-1)$ denotes the bridge vertices, and t_i $(p(v) \le i \le L(w))$ denotes the insert vertices. The square brackets for s_i work similarly to Case 1. Each of the round brackets $\overset{t_{p(w)}^p}{)}\ \overset{t_{p(w)+1}^p}{)}\ \cdots\ \overset{t_{L(w)}^p}{)}$ is used to find a parent of t_i in $G(v)$. Further, the round brackets $\overset{t_i^l}{(}\ \overset{t_i^r}{(}$ $(p(w) \le i \le L(w))$ are used to find the left and the right children, which will appear in the right of $B(u)$.

By finding matchings for square brackets and round brackets in the sequence $B(R)$ of brackets of root R of $T_{blr}(G)$, we can construct pseudo path trees. For the readers benefit, we now show an example of $B(R)$. The following sequence of brackets corresponds to the cotree illustrated in Figure 7.

$$\overset{a^p}{[}\ \overset{a^l}{(}\ \overset{a^r}{(}\ \overset{b^p}{)}\ \overset{b^l}{(}\ \overset{b^r}{(}\ \overset{c^p}{[}\ \overset{c^l}{(}\ \overset{c^r}{(}\ \overset{d^r}{]}\ \overset{d^l}{]}\ \overset{d^p}{[}\ \overset{e^p}{)}\ \overset{f^p}{)}\ \overset{e^l}{(}\ \overset{e^r}{(}\ \overset{f^l}{(}\ \overset{f^r}{(}\ .$$

Note that a and c are primary vertices, b, e, and f are insert vertices, and d is a bridge vertex. Assume that matching of square brackets and that of round brackets are computed independently. In the above sequence of brackets, we can find the following matching:

$$\overset{a^p}{[}\ \overset{d^l}{]},\ \overset{c^p}{[}\ \overset{d^r}{]},\ \overset{a^r}{(}\ \overset{b^p}{)},\ \overset{c^l}{(}\ \overset{f^p}{)}\ .\ \overset{c^r}{(}\ \overset{e^p}{)},$$

These matchings corresponds to an edge of a pseudo path tree. For example, $\overset{a^p}{[}\ \overset{d^l}{]}$ corresponds to an edge connecting the vertex a to the parent d as a left child.

In order to convert a pseudo path tree to the correct one, we need to remove illegal insert vertices. For this purpose, we will use $2p(v)-2$ *dummy* vertices. Figure 8 illustrates $2p(v)-2 = 4$ dummy vertices d_1, d_2, d_3, and d_4 connected to a pseudo path tree. The dummy vertices can have one child. After constructing the pseudo path trees with dummy vertices, for each insert vertex we check whether it is illegal. Further, we also check whether each dummy vertex is illegal, that is, it is adjacent to a bridge vertex. If there are

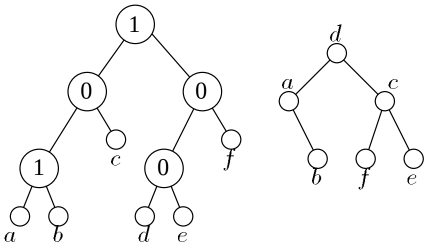

Figure 7. Construction of a pseudo path tree using brackets

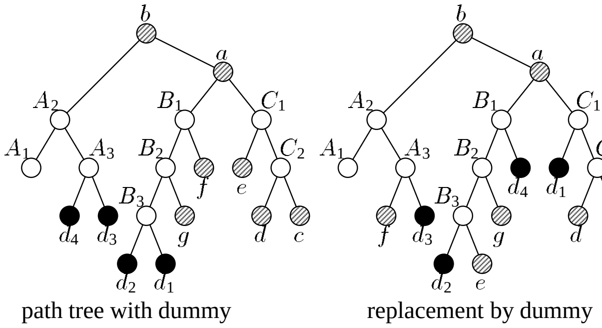

Figure 8. Construction of a path tree using dummy vertices

illegal insert vertices, they are exchanged with legal dummy vertices. In Figure 8, e and f are illegal insert vertices. Also, dummy nodes d_2 and d_3 are illegal. Thus, e and f are exchanged with d_1 and d_4, respectively. Note that this exchanging is not only for the vertices but also for the subtrees. That is, exchanging of e and d_1 means that the parent of d_1 becomes new parent of e and vice versa. After that, bypassing dummy vertices in the path tree, we can obtain a correct path tree.

5 Parallel algorithm for finding the minimum path cover of a cograph

We will show that, for given the cotree $T(G)$ of a cograph G, the minimum path cover can be exhibited efficiently. The algorithm is spelled out as follows:
Input: the adjacent list of the cotree $T(G)$ of a cograph G;
Output: the minimum path cover of G;
Step 1: Find the leftist binarized $T_{bl}(G)$;
Step 2: Generate a sequence of brackets $B(R)$ of the root R of $T_{bl}(G)$;
Step 3: Find the pseudo path tree by finding all matchings of $B(R)$;
Step 4: Convert the pseudo path tree into the (correct) path tree;
Step 5: Find the minimum path cover using the path trees.

The reader should have no difficulty to confirm that the algorithm above correctly exhibits the minimum path cover. Further, it is not so difficult to confirm that the above algorithm can be implemented to run in $O(\log n)$ time using $\frac{n}{\log n}$ processors on the EREW PRAM by the following basic algorithms: the prefix-sums, list ranking, bracket matching, preorder, postorder and inorder numberings [2, 3, 4, 6, 9].

Finally, we have

Theorem 5.1 *The task of exhibiting the minimum path cover of a cograph can be done efficiently, that is, in $O(\log n)$ time using $\frac{n}{\log n}$ processors on the EREW PRAM.*

References

[1] G. S. Adhar and S. Peng, Parallel Algorithm for Path Covering, Hamiltonian Path, and Hamiltonian Cycle in Cographs, *Proc. Internat. Conf. on Parallel Processing*, 1990, III, 364-365.

[2] R. J. Anderson, G. L. Miller, Deterministic parallel list ranking, *Algorithmica*, 6, (1991), 859-868.

[3] R. Cole and U. Vishkin, Approximate parallel scheduling. Part I: The basic technique with applications to optimal parallel list ranking in logarithmic time, *SIAM Journal on Computing*, 17, (1988) 128-142.

[4] A. Gibbons and W. Rytter, *Efficient Parallel Algorithms*, Cambridge University Press, 1989.

[5] X. He, Parallel algorithm for cograph recognition with applications, *J. Algorithms*, 15, 2, (1993), 284–313.

[6] J. JaJa, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.

[7] R. Lin, S. Olariu, G. Pruesse, An optimal path cover algorithm for cographs, *Computers and Mathematics with Applications*, 30, (1995), 75–83.

[8] R. Lin, S. Olariu, J. L. Schwing, and J. Zhang, A fast EREW algorithm for minimum path cover and hamiltonicity for cographs, *Parallel Algorithms and Applications*, 2, (1994), 99–113.

[9] R. E. Tarjan and U. Vishkin, An efficient parallel biconnectivity algorithm, *SIAM Journal on Computing*, 14 (1985) 862-874.