Pergamon

Computers Math. Applic. Vol. 30, No. 8, pp. 75–83, 1995

0898-1221/95 $9.50 + 0.00

0898-1221(95)00139-5

An Optimal Path Cover Algorithm for Cographs*

R. LIN
Department of Computer Science
SUNY at Geneseo, Geneseo, NY 14454, U.S.A.

S. OLARIU
Department of Computer Science
Old Dominion University, Norfolk, VA 23529-0162, U.S.A.

G. PRUESSE
Department of Computer Science and Electrical Engineering
University of Vermont, Burlington, VT 05405-0156, U.S.A.

(Received November 1993; accepted March 1995)

Abstract—The class of cographs, or complement-reducible graphs, arises naturally in many different areas of applied mathematics and computer science. In this paper, we present an optimal algorithm for determining a minimum path cover for a cograph G. In case G has a Hamiltonian path (cycle) our algorithm exhibits the path (cycle) as well.

Keywords—Cographs, Scheduling, Path cover, Hamiltonicity, VLSI, Greedy algorithms, Optimal algorithms.

1. INTRODUCTION

The graphs are among the few fundamental objects that arise naturally in many algorithms in computer science and engineering. A computational problem with a large spectrum of practical applications is the minimum path cover, which involves finding a minimum number of vertex-disjoint paths which together cover the vertices of a graph. The path cover problem finds application to database design, networking, VLSI design, ring protocols, code optimization, and mapping parallel programs to parallel architectures, among many others. A graph that admits a path cover of size one is referred to as Hamiltonian. If the unique path that covers all the vertices can be extended to a cycle, the graph is said to possess a Hamiltonian cycle. It is, therefore, clear that the minimum path cover problem is at least as hard as the problem of deciding whether a graph has a Hamiltonian path (resp. cycle). It is well known that, as many other interesting problems in graph theory, the minimum path cover problem and many of its variants are NP-complete [1].

It is common knowledge that in spite of the fact that many interesting problems are NP-complete on general graphs, in practical applications one rarely has to contend with general graphs. Typically, a careful analysis of the problem at hand reveals sufficient structure to limit the graphs under investigation to a restricted class. The purpose of this paper is to exhibit a

*Work supported by NSF Grants CCR-9407180, MIP-09307664, and OSR-9350450.
The algorithm presented in this paper was discovered independently by the authors. We would like to thank L. Stewart and D. Corneil for making our cooperation possible. In addition, the third author wishes to thank D. Corneil and J. Edmonds for helpful discussions.

simple and elegant algorithm to return a minimum path cover in a class of graphs that we are about to define.

The *cographs*, or complement-reducible graphs, arise so naturally in many different area of applied mathematics and computer science that their independent discovery by various researchers comes as no surprise. In the literature, the cographs are also known as P_4-restricted graphs [2], D^*-graphs [3], HD-graphs [4], and CU-graphs [5]. This class of graphs has been studied extensively from both the theoretical and algorithmic points of view [2–8]. An early characterization [8] asserts that cographs are precisely the graphs which contain no induced subgraph isomorphic to the chordless path with three edges.

The class of cographs is defined recursively as follows:

- a single-vertex graph is a cograph;
- if G is a cograph, then its complement $\overline{G}$ is also a cograph;
- if G and H are cographs, then their union is also a cograph.

As it turns out [8], the cographs admit a unique tree representation up to isomorphism. Specifically, we can associate with every cograph G a unique rooted tree $T(G)$ called the *cotree* of G, featuring the following.

PROPERTY 1. Every internal node, except possibly for the root, has at least two children; furthermore, the root has only one child if, and only if, the underlying graph G is disconnected.

PROPERTY 2. The internal nodes are labeled by either 0 (0-nodes) or 1 (1-nodes) in such a way that the root is always a 1-node, and such that 1-nodes and 0-nodes alternate along every path in $T(G)$ starting at the root.

PROPERTY 3. The leaves of $T(G)$ are precisely the vertices of G, such that vertices x and y are adjacent in G if, and only if, the lowest common ancestor of x and y in $T(G)$ is a 1-node.

Figure 1 features a cograph along with its unique tree representation.

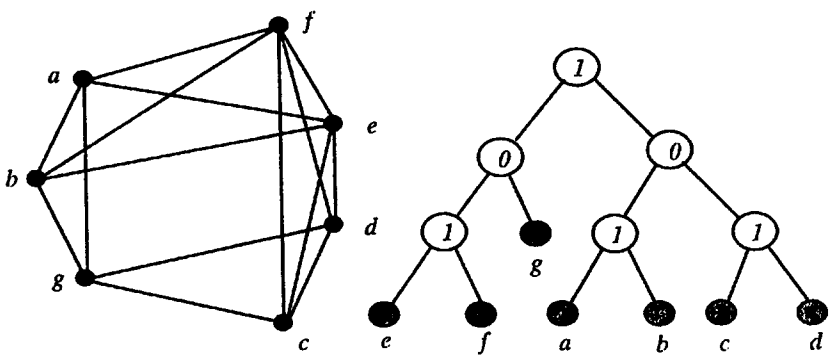
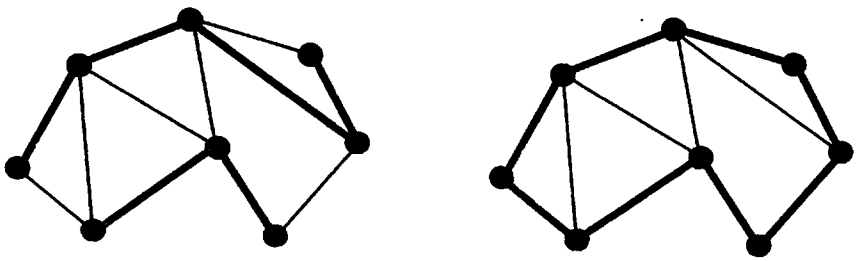


Figure 1. Illustrating a cograph and its cotree.

A *path cover* in a graph G is a set P of paths of G that contains all the vertices in G. A path cover is termed *minimum* if it uses the smallest possible number of paths. For an illustration, refer to Figure 2: Figure 2(a) shows a possible path cover; Figure 2(b) features a minimum path cover.

The *minimum path cover problem* is to find a path cover of the smallest cardinality. This problem finds important applications to scheduling, VLSI, operating systems, among many others. A graph G that admits a path cover of size one is referred to as *Hamiltonian*. If the unique path that covers G can be extended to a cycle, G is said to possess a Hamiltonian cycle. It is well



(a) (b)

Figure 2. Various path covers of a graph.

known that the problem of determining whether a graph G has a Hamiltonian path or cycle is one of the most difficult problems in computational graph theory.

An algorithm to determine the hamiltonicity of cographs was given in [6]. Unlike that algorithm, the one presented here is constructive in nature. In fact, we propose an algorithm that returns a minimum path cover for a cograph G. In case G has a path cover of size one, our algorithm exhibits a Hamiltonian path in G. In case G has a Hamiltonian cycle, the unique path returned by our algorithm as a path cover for G can be augmented trivially to yield a Hamiltonian cycle.

The remainder of this paper is organized as follows. Section 2 presents the idea of our approach in terms slightly more general than the cographs; Section 3 proposes the minimum path cover algorithm for cographs along with the proof of correctness and a timing analysis; Section 4 concludes with a number of open problems.

2. OUR BASIC IDEA

All graphs in this paper are finite with no loops or multiple edges. We use standard graph theoretical terminology compatible with [9]. Let G be an *arbitrary* graph whose vertex-set partitions into nonempty, disjoint sets A and B with $r = |A| \leq |B| = t$, and such that every vertex of A is adjacent to all the vertices in B. Let $P_B = \{p_1, p_2, \ldots, p_s\}$, $s \geq 1$, be a minimum path cover for B. For convenience, we enumerate the vertices of A arbitrarily as

$$v_1, v_2, \ldots, v_r; \tag{1}$$

similarly, enumerate the vertices of B as

$$w_1, w_2, \ldots, w_t \tag{2}$$

by first writing down the vertices of p_1 (in the same order as they appear in p_1), followed by the vertices in p_2, and so on.

The next result establishes a property of the minimum path cover of G which will be instrumental in our path cover algorithm for cographs.

THEOREM 1. *G has a minimum path cover of size* $\max\{1, s - r\}$.

PROOF. We first argue that G cannot have a path cover of size k with $k < \max\{1, s-r\}$. Assume that such a path cover exists. Consider removing from this path cover all the vertices in A. What results is a set of paths which is clearly a path cover for B.

Since the removal of a vertex in A will increase the number of paths by at most one, we obtain a path cover for B of size at most $k + r$. Now the assumption that $k < \max\{1, s - r\}$ guarantees that $k + r < s$, contradicting the minimality of P_B.

To complete the proof of Theorem 1, we shall present an algorithm that actually returns a path cover of G of size $\max\{1, s - r\}$. In outline, our algorithm proceeds in the following two stages. (Refer to Figures 3–5 for an illustration.)

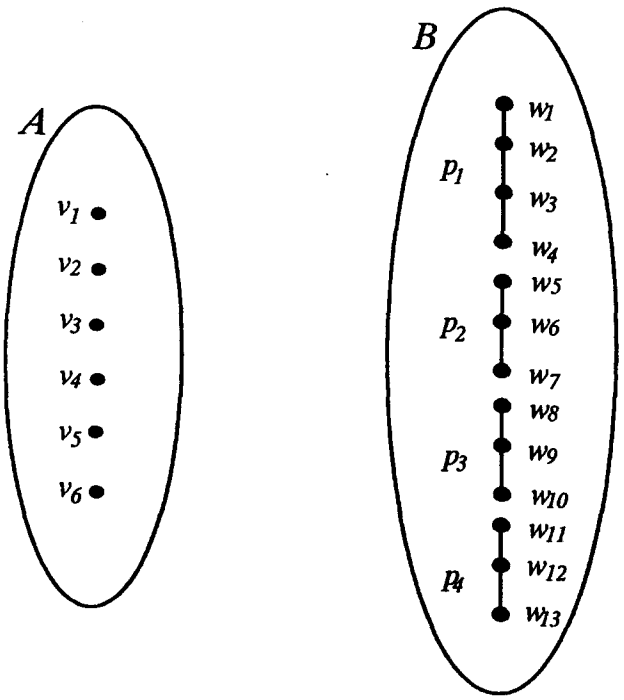


Figure 3. Illustrating Mend_and_Merge: the initial setup.

THE MENDING STAGE. The idea of this stage is to use vertices in A to "stitch" together disjoint paths in P_B. Specifically, we begin by initializing

$$i \leftarrow 1, \quad A' \leftarrow A, \quad p \leftarrow p_i, \quad \text{and } B' \leftarrow B \setminus \{p_i\}.$$

Then, repeatedly, we attempt to extend p, in the natural way, by using the vertex v_i to join p and $p_i + 1$; afterwards, we remove v_i from A', the path p_{i+1} from B', and set $i \leftarrow i + 1$.

The mending stage ends when precisely one of the following conditions is satisfied:

$$A' = \emptyset; \tag{3}$$

the number of vertices in B' becomes less than the number of vertices in A'. (4)

In case the mending stage ends with $A' = \emptyset$, the algorithm returns the set of paths $\{p, p_{i+1}, \ldots, p_s\}$ which, by our construction, is a path cover of G of size $\max\{1, s - r\}$.

If at the end of the mending stage A' is not empty, we proceed to the next stage of our algorithm. For further reference we note that at this point A' contains $r-i+1$ vertices, namely $v_i, v_{i+1}, \ldots, v_r$. Furthermore, B' is the set of vertices contained in the paths $p_{i+1}, p_{i+2}, \ldots, p_s$.

THE MERGING STAGE. The idea of this stage is to incorporate the vertices in A' into the set of paths $\{p, p_{i+1}, \ldots, p_s\}$ to create a unique path that covers all the vertices of G. For this purpose, consider the last $r - i + 1$ vertices

$$w_{t-r+i}, w_{t-r+i+1}, \ldots, w_t$$

in the enumeration of the vertices of B specified by (2).

The correctness of this stage relies on the following intermediate result.

LEMMA 2. *The vertex w_{t-r+i} belongs to the path p. Furthermore, no vertex w_j with $j \geq t-r+i$ belongs to A.*

PROOF OF LEMMA 2. To begin, note that since A' is not empty, it must be the case that (4) holds true. Consequently, B' contains strictly fewer than $r - i + 1$ vertices, implying that w_{t-r+i} belongs to the path p. We distinguish between the following two cases.

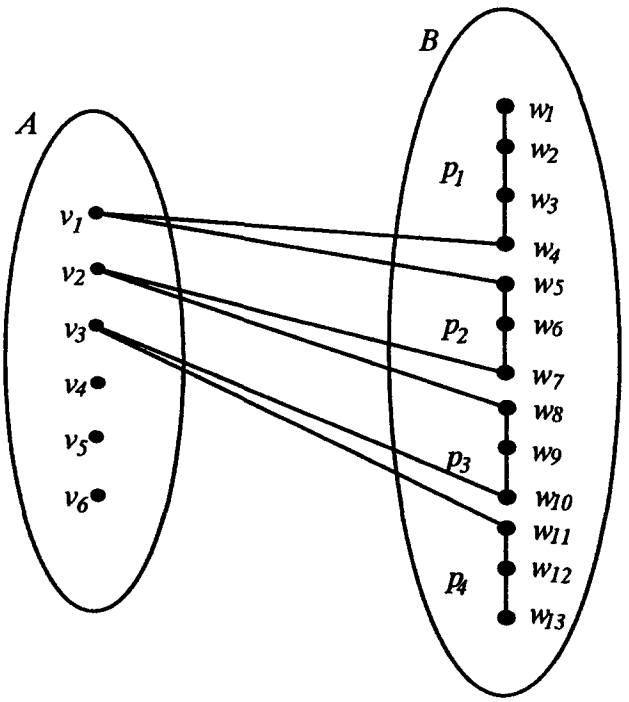


Figure 4. Illustrating Mend_and_Merge: after the mending stage.

CASE 1. $i = 1$.
If this is the case, then no vertex in A is used to stitch together vertices in B, and so $A' = A$ and $B' = B \setminus \{p_1\}$. Now the assumption that $r = |A| \leq |B| = t$ guarantees that $t - r + i \geq 1$ and so all vertices w_j with $j \geq t - r + i$ belong to B, as claimed.

CASE 2. $i \geq 2$.
By our construction, vertices $v_1, v_2, \ldots v_{i-1}$ of A were used in the mending stage to join paths in B. In particular, when v_{i-1} was so used, neither of conditions (3) and (4) was satisfied, implying that the set of vertices contained in the paths $p_i, p_{i+1}, \ldots, p_s$ contained at least $r - i + 2$ elements. On the other hand, at the end of the mending stage, the set B' contained fewer than $r - i + 1$ elements. Since v_{i-1} was used to join p and p_i, it follows that all vertices w_j with $j \geq t - r + i$ belong to $p_i, p_{i+1}, \ldots, p_s$, and Case 2 is settled.

With this, the proof of Lemma 2 is complete. ∎

Formally, the merging stage begins by removing from the path p all vertices w_j with $j \geq t - r + i + 1$. Next, assign every vertex

- v_j with $1 \leq j \leq r$ the label $2(j - i + 1)$;
- w_j with $t - r + i \leq j \leq t$ the label $2(j - t + r - i) + 1$.

Trivially, after this assignment, the vertices in A' receive labels $2, 4, 6, \ldots, 2(r - i + 1)$, while the vertices in B' receive labels $1, 3, \ldots, 2(r - i) + 1$. Now consider the sequence

$$p' : w_{t-r+i}, v_i, w_{t-r+i+1}, \ldots, v_{r-1}, w_t, v_r$$

obtained by merging the vertices in A' and B' according to their labels. By Lemma 2, together with the assumption that every vertex in A is adjacent to all the vertices in B, it follows that p' is, in fact, a path in G. Note, further, that the paths p and p' share exactly one vertex, namely $w_t - r + i$.

Therefore, the path obtained by concatenating p and p' contains all the vertices in G. The details are spelled out by the following procedure. (Here, the procedure append used in lines 6 and 16 works as follows: in line 6, v_i is joined to the last vertex in p and to the first vertex

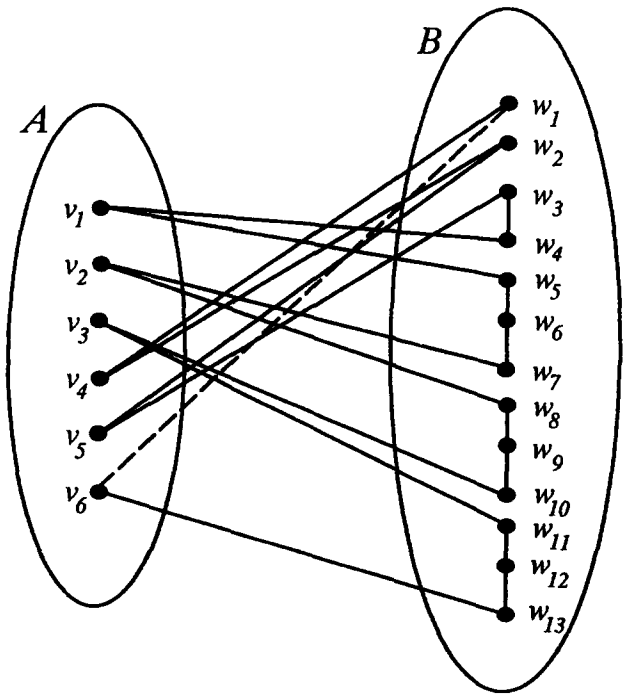


Figure 5. Illustrating Mend_and_Merge: after the merging stage.

in p_{i+1}, thus extending the path p; in line 16, the two paths p and p' are joined along their common vertex.)

Procedure Mend_and_Merge(A, P_B)
0. **begin**
1. $A' \leftarrow A$;
2. $i \leftarrow 1$;
3. $p \leftarrow p_i$;
4. $B' \leftarrow B \setminus \{p_i\}$;
5. **while** $A' \neq \emptyset$ and $|B'| \geq |A'|$ **do**
6. $p \leftarrow$ append(p, v_i, p_{i+1});
7. $A' \leftarrow A' \setminus \{v_i\}$;
8. $B' \leftarrow B' \setminus \{p_{i+1}\}$;
9. $i \leftarrow i + 1$
10. **endwhile**;
11. **if** $A' = \emptyset$ **then**
12. return$(\{p, p_{i+1}, \ldots, p_s\})$
13. **else** {start merging stage}
14. remove vertices w_j with $j \geq t - r + i + 1$ from p;
15. $p' \leftarrow w_{t-r+i}, v_i, w_{t-r+i+1}, \ldots, v_{r-1}, w_t, v_r$;
16. return(append(p, p'))
17. **endif**
18. **end**; {Mend_and_Merge}

This completes the proof of Theorem 1. ∎

COROLLARY 1.1. *If $r = s - 1$ then G admits a Hamiltonian path. If $r > s-$ then G admits a Hamiltonian cycle.*

PROOF. Trivially, in case $r = s - 1$, at the end of the mending stage, A' and B' are both empty confirming that G has a Hamiltonian path. In case $r > s - 1$, then A' is nonempty when the mending stage ends; consequently, G has a path cover of size one returned in line 16 of procedure Mend_and_Merge. Note that, by our construction, the first vertex of path p is in B, while the last vertex of the path p' is in A. Now the fact that every vertex in A is adjacent to all the vertices in B guarantees that the edge $v_r\ w_1$ can be added to the concatenation of p and p' to obtain a Hamiltonian cycle of G. ∎

Next, we show that if a suitable data structure is used to maintain the collection P_B of paths, then the procedure Mend_and_Merge can be implemented efficiently. More precisely, we maintain the vertices of B in a *unique* doubly linked list corresponding to the enumeration in (4). The set A is maintained as a doubly-linked list, as well. Individual paths in P_B are delimited using a second doubly-linked list: the j^{th} entry $(1 \leq j \leq s)$ in this second list contains the following information:

- a pointer to the first element of path p_j;
- a pointer to the last element of p_j;
- the number of elements in p_j.

It is easy to confirm that each iteration of the mending stage takes O(1) time if a counter for the elements in B' is also maintained. Consequently, the mending stage takes at most $O(|A|)$ time altogether.

The merging stage starts off by removing from p all the vertices w_j with $j \geq t - r + i + 1$. This takes $O(r - i) \subseteq O(|A|)$ time. The subsequent merging itself takes $2|A| - 1$ operations, implying that the running time of the procedure Mend_and_Merge is bounded by $O(|A|)$. To summarize our findings, we state the following result.

THEOREM 3. *The procedure Mend_and_Merge can be implemented in such a way that its running time is bounded by O(—A—).* ∎

3. A MINIMUM PATH COVER ALGORITHM FOR COGRAPHS

Let G be an n-vertex cograph represented by its cotree $T(G)$. Imagine that the cotree $T(G)$ is a binary tree, and let x be an arbitrary node of this cotree. We plan to compute, recursively, a minimum path cover of the subgraph of G induced by the leaves of the subtree of $T(G)$ rooted at x. For this purpose, we note that in case x is a leaf we return, simply, $\{x\}$. By Property 2 and Property 3 in the definition of the cotree, should x be a 0-node we only need return the union of the paths covers corresponding to its left and right subtrees, respectively.

In case x is a 1-node, Property 3 guarantees that every leaf in the left subtree of x is adjacent (as a vertex of G) to all the leaves in the right subtree of x. Now to use procedure Mend_and_Merge developed in the previous section, we only need ensure that the left subtree of x contains no more leaves than its right subtree.

The previous discussion motivates us to preprocess $T(G)$ as follows: first, we binarize $T(G)$ and then proceed to swap the subtrees of every 1-node such that no left subtree contains more leaves than the corresponding right subtree. Finally, we shrink the left subtree of every 1-node to its root, as we are about to explain.

First, to *binarize* $T(G)$ (refer to Figure 6 for illustration) we add to every node x of degree k by $k - 2$ identical copies of x, namely $x_1, x_2, \ldots, x_{k-2}$ in such a way that, with x_0 standing for x,

- the parent of x_i is x_{i-1} whenever $i \geq 1$;
- the left child of x_i is the $(i+1)^{\text{st}}$ child of x in $T(G)$;
- the right child of x_i is x_{i+1} in case $i \leq k - 3$, and the k^{th} child of x in $T(G)$ otherwise.

Let $BT(G)$ be the binarized version of $T(G)$. For each node x of $BT(G)$, we let $BT(x)$ stand for the subtree of $BT(G)$ rooted at x; $L(x)$ stands for the set of all the leaves in $BT(x)$; x_left

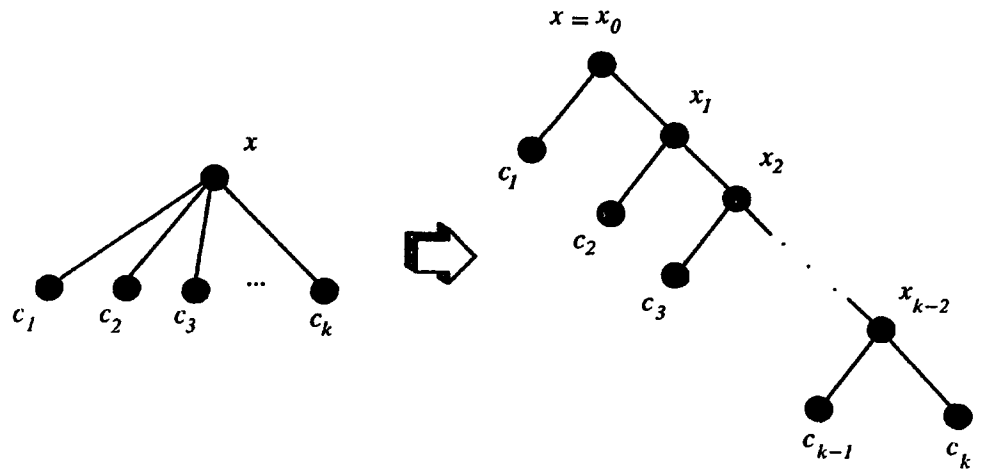


Figure 6. Illustrating the process of binarizing a tree.

and x_right will denote the left and right children of x, respectively. Next, by using standard traversing techniques, we determine $L(x)$ for every node x of $BT(G)$. In case x is a 1-node, we do the following (we continue to refer to the new version of the tree as $BT(G)$):

---

if $|L(x\_left)| > |L(x\_right)|$ **then**
  swap($T(x\_left), T(x\_right)$);
  shrink $BT(x\_left)$ to its root;

In other words, we ensure that for every 1-node, its left subtree contains at most as many leaves as the right subtree. Furthermore, the left subtree of every 1-node is shrunk to its root (we can think of this as replacing the root of this subtree by a *supernode* that contains, in some order, all the vertices in $L(x\_left)$. The details of our algorithm to compute a minimum path cover for cographs can be spelled out as follows.

Procedure Find_Min_Path_Cover(v);
0. **begin**
1. **if** v is a leaf **then** return(v);
2. **if** v is a 0-node **then**
3. return(Find_Min_Path_Cover(v_left) ∪ Find_Min_Path_Cover(v_right))
4. **else**
5. return(Mend_and_Merge($L(v\_left)$, Find_Min_Path_Cover(v_right)))
6. **end**; {Find_Min_Path_Cover}

THEOREM 4. *With an arbitrary n-vertex cograph G represented by its cotree as input, procedure Find_Min_Path_Cover returns a minimum path cover of G in $O(n)$ time.*

PROOF. The correctness of the procedure follows instantly from Property 2, Property 3 together with Theorem 1. To argue for the complexity, we note that processing a 0-node takes O(1) time; by virtue of Theorem 3, processing a 1-node v takes $O(L(v\_left))$. Since $|L(v\_left)| \leq |L(v\_right)|$, it follows that the overall time needed to process 1-nodes is bounded by $2n$. The conclusion follows. ∎

4. CONCLUSIONS AND OPEN PROBLEMS

We have proposed an optimal algorithm to compute a minimum path cover for cographs. An interesting open question would be to see if a similar technique applies for the purpose

of determining a minimum path cover for other classes of graphs related to the cographs. Of particular interest are the classes of P_4-reducible graphs [10] defined as graphs for which every vertex can belong to at most one chordless path of length three, and the class of P_4-sparse graphs [11], defined as graphs for which no set of five vertices induces more than one chordless path of length three.

A second direction for further research is to obtain efficient parallel algorithms to compute a minimum path cover for cographs. The first known attempt at solving the problem in parallel goes back to [12]. Specifically, with an n-vertex cograph G represented by its cotree as input, the algorithm in [12] returns a minimum path cover in a cograph in $O(\log^2 n)$ time using $O(n^2)$ processors in the Concurrent Read Concurrent Write (CRCW) PRAM. It is interesting to note that the algorithm in [12] requires $O(\log^2 n)$ time and $O(n^2)$ CRCW processors even to detect whether the graph at hand contain a Hamiltonian cycle. More recently, Lin *et al.* [13] have obtained an algorithm that determines the number of paths in a minimum path cover in $O(\log\ n)$ time using $n/\log n$ processors in the Exclusive Read Exclusive Write (EREW) PRAM. Consequently, they can answer the question: "Does the graph have a Hamiltonian path (cycle)?" optimally in parallel. Unfortunately, the problem of exhibiting all the paths in a minimum path cover requires $O(\log^2 n)$ time and uses $n/\log n$ processors in the EREW-PRAM, being suboptimal. However, the result in [13] is a considerable improvement over the algorithm in [12] both in the model of computation and in the number of processors used. It would be interesting to see if a cost-optimal parallel algorithm for this problem can be devised. To the best of our knowledge, no such algorithm has been proposed in the literature.

REFERENCES

1. M.R. Garey and D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-completeness*, Freeman, San Francisco, CA, (1979).
2. D. Seinsche, On a property of the class of n-colorable graphs, *Journal of Combinatorial Theory (B)* **16**, 191–193 (1974).
3. H.A. Jung, On a class of posets and the corresponding comparability graphs, *Journal of Combinatorial Theory (B)* **24**, 125–133 (1978).
4. D.P. Sumner, Dacey graphs, *Journal of the Australian Math. Society* **18**, 492–502 (1974).
5. D.G. Corneil and D.G. Kirkpatrick, Families of recursively defined perfect graphs, *Congressus Numerantium* **39**, 237–246 (1983).
6. D.G. Corneil, H. Lerchs and L.S. Burlingham, Complement reducible graphs, *Discrete Applied Mathematics* **3**, 163–174 (1981).
7. D.G. Corneil, Y. Perl and L.K. Stewart, A linear recognition algorithm for cographs, *SIAM Journal on Computing* **14**, 926–934 (1985).
8. H. Lerchs, On the clique-kernel structure of graphs, a manuscript, Department of Computer Science, University of Toronto, (October 1972).
9. J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, North-Holland, Amsterdam, (1976).
10. B. Jamison and S. Olariu, P_4-reducible, a class of uniquely tree representable graphs, *Studies in Applied Mathematics* **81**, 79–87 (1989).
11. B. Jamison and S. Olariu, A tree representation for P_4-sparse graphs, *Discrete Applied Mathematics* **35**, 115–129 (1992).
12. G.S. Adhar and S. Peng, Parallel algorithm for path covering, Hamiltonian path, and Hamiltonian cycle in cographs, In *Proceedings of the International Conference on Parallel Processing*, Vol. III, St. Charles, IL, 364–365, (August 1990).
13. R. Lin, S. Olariu, J.L. Schwing and J. Zhang, An efficient EREW algorithm for minimum path cover and hamiltonicity on cographs, *Parallel Algorithms and Applications* **2**, 99–113 (1994).
14. L. Stewart, Cographs, a class of tree representable graphs, M.Sc. Thesis, Department of Computer Science, University of Toronto, (1978, TR 126/78).