Habib
Sami
Hadja

SIAM J. COMPUT.
Vol. 14, No. 4, November 1985

A LINEAR RECOGNITION ALGORITHM FOR COGRAPHS*

D. G. CORNEIL†, Y. PERL‡ AND L. K. STEWART†

Abstract. Cographs are the graphs formed from a single vertex under the closure of the operations of union and complement. Another characterization of cographs is that they are the undirected graphs with no induced paths on four vertices. Cographs arise naturally in such application areas as examination scheduling and automatic clustering of index terms. Furthermore, it is known that cographs have a unique tree representation called a cotree. Using the cotree it is possible to design very fast polynomial time algorithms for problems which are intractable for graphs in general. Such problems include chromatic number, clique determination, clustering, minimum weight domination, isomorphism, minimum fill-in and Hamiltonicity. In this paper we present a linear time algorithm for recognizing cographs and constructing their cotree representation.

Key words. cographs, permutation graphs, perfect graphs, linear algorithms, examination scheduling

1. Introduction. Following Cook's pioneering work in establishing the NP-completeness of the satisfiability problem [2], thousands of other problems were also shown to be NP-complete or NP-hard [6]. Since many practical problems are "intractable" in this sense, hopes for producing algorithms which would find optimal solutions in a reasonable amount of time had to be abandoned and instead attention was directed to the development and analysis of heuristic algorithms. It was soon realized that the very pessimistic worst-case analysis included in the NP-complete results did not accurately reflect the success which heuristic algorithms were achieving on real-world combinatorial optimization problems. As Karp noted "one of the great mysteries in the field of combinatorial algorithms is the baffling success of many heuristic algorithms" [9].

The search for a theoretical answer to this question leads to three main approaches. The first allows approximations to be made to the optimal solution. In a very few cases a constant bound on the ratio of the approximation to the optimal may be proven however in most cases such a bound is not known. Furthermore even if such a bound is known, in practice the observed behaviour of the algorithm is often much better than the bound. The second approach substitutes either expected case or average case analysis for worst-case analysis. Typically this type of research involves the probabilistic analysis of a particular heuristic algorithm. Often this is accomplished by analyzing the algorithm's average behaviour on some form of random input, for example random graphs. One shortcoming of this approach is that usually the graphs encountered in practice have some structure which violates the assumption that all edges have the same independent probability of being present. For this reason the algorithm's performance in a real-world environment often is not as good as predicted by the optimistic expected case or average case analysis.

The third approach, and the one we take in this paper, maintains the worst-case analysis and restricts the class of input to be considered. The hope here of course is that the intractable problem will be solvable in polynomial time on this restricted class of input and furthermore that membership in this restricted class can be recognized quickly. Examples of families of graphs which have received this type of study include

* Received by the editors August 16, 1983 and in final revised form July 27, 1984.

† Department of Computer Science, University of Toronto, Toronto, Ontario, M5S 1A1, Canada.
‡ Department of Computer Science, Rutgers University, New Brunswick, New Jersey 08903, and Bar Ilan University, Ramat-Gan, Israel.

comparability graphs [11], permutation graphs [11], interval graphs [1], chordal graphs [10] and planar graphs [8]. The difficulty with this approach is that one can usually provide only an intuitive justification that the restricted class is a good model for the input expected in practice. Furthermore, even if the restricted class does meet this criterion, it may happen that a particular input is very close to being a member of the restricted class but in fact is not. The algorithms developed for such a class must be sufficiently robust to allow adaptation to these "near misses".

In this paper, we are concerned with combinatorial optimization problems where the input graphs are expected to obey a certain "local density" property. In particular the application suggests that the graphs are unlikely to have more than a very few induced paths on four vertices. Thus if any four vertices a, b, c, d form a path then it is expected that at least one of (a, c), (b, d) and (a, d) is also in the edge set. Applications where this structure is expected to occur include examination scheduling and automatic clustering of index terms. In examination scheduling, the vertices represent courses and the edge (i, j) signifies the existence of a student taking both courses i and j. The weight of the edge indicates the number of such students. In any colouring of the graph the colour classes represent courses whose examinations may be held concurrently. It is anticipated that such conflict graphs are very unlikely to have long induced paths. In the second application we want to generate clusters (subsets of vertices with a high density of edges) on a graph where edges represent the proximity or self-referencing of index terms. Again it is expected that if four index terms form a path, then at least one of the other edges will also be present (see [7]). As we shall see, the P_4 restricted graphs are precisely cographs.

Cographs (or *complement reducible graphs*) are defined as the class of graphs formed from a single vertex under the closure of the operations of union and complement. Cographs were independently discovered under various names and were shown by Lerchs (see [3]) to have the following two remarkable properties. First they are exactly the P_4 restricted graphs described above and secondly a cograph has a unique tree representation. This tree, called a *cotree*, forms the basis for fast polynomial time algorithms for problems such as isomorphism, colouring, clique detection, clusters, minimum weight dominating sets, minimum fill-in and Hamiltonicity [3], [4], [5].

Naturally before such algorithms can be employed it is necessary to have a fast cograph recognition algorithm which also produces the cotree on recognition of a cograph. In § 3 the outline of such a linear time algorithm is presented. This algorithm follows from a new theorem on the structure of cographs that forms the basis of a heuristic algorithm to deal with graphs which are almost cographs. Before discussing this material we present examples and properties of cographs.

2. Cographs. Cographs are perfect and in fact form a proper subset of the permutation graphs. Furthermore, cographs are precisely the underlying undirected graphs of transitive series-parallel digraphs [12]. Although there is a linear time algorithm for recognizing if a digraph is transitive series-parallel [13], this algorithm does not seem to be amenable to the cograph case. In fact, as noted by Spinrad "recognition of transitive series-parallel graphs seems easier than recognition of cographs" [11]. We will show that this is not the case.

An example of a cograph and its cotree is presented in Fig. 1. In the cotree representation leaves of the cotree represent vertices of the graph. Internal nodes of the cotree are labelled 0 or 1 in such a way that (0) nodes and (1) nodes alternate along every path starting from the root which is always a (1) node. The root will have only one (0) node child if and only if the represented cograph is disconnected. Two

vertices x and y of the graph are adjacent if and only if the unique path from x to the root of the tree meets the unique path from y to the root at a (1) node.

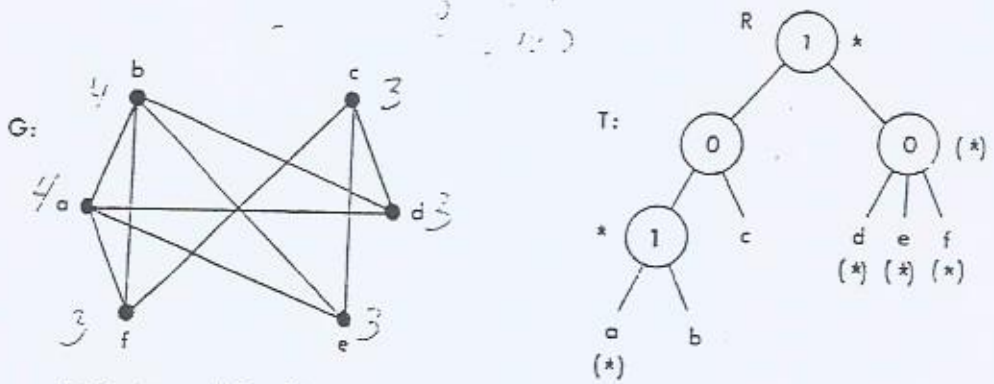

FIG. 1. *A cograph G and its cotree T. The * and (*) symbols refer to procedure* MARK *in* § 3.1.

Cograph algorithms typically associate operations with the (1) nodes and the (0) nodes of the cotree and assign weights or numbers to the leaves. The algorithm then uses the operations on the interior nodes to calculate particular values associated with the subgraph rooted at the interior node. The value assigned to the root is the value assigned to the cograph. For example, if the leaves are assigned the value $a=1$ and the (1) nodes have $a=\prod_{i=1}^{k} a_i$ and the (0) nodes have $a=\sum_{i=1}^{k} a_i$ where $a_1, \cdots, a_k$ are the values associated with the children, then the a value of the root is the number of maximal complete subgraphs in the given cograph. For the example in Fig. 1, the number of such maximal complete subgraphs is 6 corresponding to the subgraphs (abd), (abe), (abf), (cd), (ce), (cf).

3. Cograph recognition.

3.1. Theoretical foundations of the algorithm. In [12] an $O(n^2)$ algorithm is presented for the recognition of cographs and the construction of the corresponding cotrees. Both that algorithm and our algorithm are incremental in the sense that the vertices are processed one at a time until the entire graph has been handled. Since cographs are *hereditary* (i.e., every induced subgraph is a cograph) we assume we have a cograph G with cotree T and present an algorithm which determines if $G+x$ is also a cograph and, if so, modifies T to represent $G+x$. Given the list of the vertices of G adjacent to x we percolate this adjacency information up the tree from the leaves to the root using a simple marking scheme. Before presenting that algorithm we introduce the following notation. For a node w of T (rooted at R), $d(w)$ denotes the number of children of w in T and $md(w)$ is the current number of children of w which have been both "marked" and "unmarked". For all nodes w, the value of $md(w)$ is initially 0 and is reset to 0 when w is "unmarked". The following procedure uses the adjacency information of the new vertex x to "mark" and then "unmark", where appropriate, the nodes in T.

```
MARK(x)
  Mark all leaves of T which are adjacent to x
  For each marked node u of T with d(u) = md(u)
    do unmark (u);
       md(u) ← 0;
```

if $u \neq R$ *then mark* (w) where w is the parent of u;
$md(w) \leftarrow md(w)+1$;
insert u at the head of a linked list of marked
and unmarked children of w

end
If any vertex remains marked and $d(R)=1$ *then mark* R
end MARK

Let M denote the set of internal nodes of T which remain marked after the procedure MARK has been performed, and let α be a node in M with lowest level in the tree and let β be a node in $M\backslash\{\alpha\}$ with lowest level. We say a marked (1) node γ is *properly marked* if and only if $md(\gamma)=d(\gamma)-1$. A *legitimate alternating path* in a marked cotree is a path of adjacent alternating properly marked (1) nodes and unmarked (0) nodes, the extreme points of which are (1) nodes.

The procedure MARK is illustrated for adding a vertex x adjacent to the vertices a, d, e and f in the graph of Fig. 1. We associate * with marked nodes and (*) with nodes which are subsequently unmarked. See Fig. 1. In this example $M=\{R, \text{parent}(a)\}$, $\alpha=\text{parent}(a)$ and $\beta=R$.

We now show that the marked set M determines whether or not $G+x$ is a cograph. Later we use this theorem as the basis of our linear time cograph recognition algorithm.

THEOREM 1. *If G is a cograph with cotree T then $G+x$ is a cograph if and only if*

1. *M is empty or*

2. (i) *$M\backslash\{\alpha\}$ consists of exactly the* (1) *nodes of a (possibly empty) legitimate alternating path which ends at R and*

(ii) *α is either a* (0) *node whose parent is β or α is a* (1) *node whose grandparent, if it exists, is β.*

Proof. Only if. If the conditions of the theorem do not hold then we have at least one of the following:

(i) $M\backslash\{\alpha\}$ contains a (0) node.
(ii) $\exists$ a (1) node in $M\backslash\{\alpha\}$ which is not properly marked.
(iii) $\exists \gamma \neq R$ in $M\backslash\{\alpha\}$ s.t. the grandparent of γ is not in $M\backslash\{\alpha\}$.
(iv) The vertices of $M\backslash\{\alpha\}$ do not lie on one path to R.
(v) α is a (0) node whose parent is not β.
(vi) α is a (1) node which has a grandparent which is not β.

By definition, any vertex in M has been marked but not unmarked, and this implies there is at least one descendant leaf adjacent to x and at least one not adjacent to x. Using this fact, it is fairly straightforward to show that any of the above six conditions implies the existence of an induced P_4 in $G+x$. As an example, we demonstrate an induced P_4 in the graph $G+x$ if condition (i) is found to be true. The following notation is used: for any internal node θ of T, des (θ) denotes the set of descendant leaves of θ, that is, the leaves of the subtree of T rooted at θ. Let γ be a (0) node in $M\backslash\{\alpha\}$ and let δ be the lowest common ancestor of α and γ in T. Note the possibility that $\delta=\gamma$. There are four cases to be considered, depending on the labels (0) or (1) of α and δ.

Case 1. *α and δ are both* (0) *nodes.*

Proof. There is an induced P_4 on vertices (b, c, x, d) if x and c are adjacent in $G+x$, or on (b, c, a, x) if x and c are not adjacent, where: $a \in \text{des}(\alpha)$ and is adjacent to x; $b \in \text{des}(\alpha)$ and is not adjacent to x; $c \in \text{des}(\text{parent}(\alpha))\backslash\text{des}(\alpha)$; $d \in \text{des}(\gamma)$ and is adjacent to x. If $\delta=\gamma$, we require that $d \in \text{des}(\gamma)\backslash\text{des}(\theta)$, where θ is the child of γ on the $\alpha-\gamma$ path.

Cases 2, 3 and 4 follow similarly.

If. We complete this part of the proof by constructing T', the cotree representing $G+x$

1. If M is empty, then x can be added as a child of the root if $G+x$ is connected, or as a child of the only child of the root in the case where $G+x$ and G are both disconnected. If $G+x$ is disconnected but G is connected the root of T and x both become children of the only child of a new root.

2. There is a lowest marked node $\alpha \in M$. Let A be the children of α which were marked and subsequently unmarked by procedure MARK. Similarly, let B be the children of α which were not marked by MARK. The fact that $\alpha \in M$ implies that $|A| \geq 1$ and $|B| \geq 1$. To construct T' there are two cases to consider.

Case (i). *α is a* (0) *node.*

In this case, the elements of A and B are either leaves or (1) nodes.

If $|A|=1$ and $a \in A$ is a leaf then we add a new (1) node in place of a and make a and x children of this node. If $|A|=1$ and $a \in A$ is a (1) node then we simply add x as a new child of a.

If $|A|>1$ then we remove all elements of A from α and add a new (1) node in their place. Children of this new node are x and a new (0) node with elements of A as children.

Case (ii). *α is a* (1) *node.*

The proof follows exactly as in case (i), except that B is examined instead of A, and the roles of (0) nodes and (1) nodes are reversed.

To see that T' is an accurate representation of $G+x$, we observe that the alterations to T correctly reflect adjacencies of x with vertices in the subtree rooted at α, and the fact that we have a legitimate alternating path from α to R guarantees that all other adjacencies of x are correctly represented. Adjacencies among vertices of G remain unchanged as required. □

3.2. The cograph recognition algorithm. As stated previously, the algorithm for recognizing cographs and constructing their cotrees is an incremental one. We begin with the cotree for two vertices and incorporate the remaining vertices into the tree one by one. Each iteration essentially consists of an efficient implementation of the preceding theorem. The complete algorithm follows.

ALGORITHM COGRAPH-RECOGNITION. Given a graph $G=(V, E)$ with vertices arbitrarily indexed $v_1, \cdots, v_m$ this algorithm determines whether or not G is a cograph and constructs G's cotree T if G is a cograph. We assume that *md* is set to zero for all new nodes including leaves as they are added to T.

1. (Initialize.)
   Create a new (1) node R
   If $(v_1, v_2) \in E$
   then add v_1, v_2 as children of R
   else create a new (0) node N;
   add N as a child of R; add v_1 and v_2 as children of N
2. (Iteratively incorporate $v_3, \cdots, v_n$ into T.)
   For $x \leftarrow v_3, \cdots, v_n$ *do*:
   2.1. *Call* procedure MARK (x)
   2.2. *If* all nodes of T were marked and unmarked
   then add x as a child of R;
   goto endloop

2.3. *If* no nodes of T were marked
then if $d(R)=1$
then add x as a child of the only child of R
else create a new (1) node R with one child (a new (0) node)
and two grandchildren: x and the old root;
goto endloop
2.4. $u \leftarrow$ FIND-LOWEST
2.5. Let $A(B)$ denote the set of children of u which were (were not) marked
if label$(u)=0(=1)$
then if $|A|=1$ $(|B|=1)$
then if $w \in A(\in B)$ is a leaf
then add a new (1) node ((0) node) in place of w
and make w and x children of this node
else add x as a new child of w
else remove all elements of A from u and add them as children
of a new node y with $label(y) = label(u)$
if u is a (0) node
then add a new (1) node as a child of u; children of
this new (1) node are x and y
else remove u from its parent and add y in its place;
add a new (0) node as a child of y; children of
this new (0) node are x and u

endloop
end COGRAPH-RECOGNITION

Function FIND-LOWEST. This function checks whether $G+x$ is a cograph and, if so, returns u, the lowest marked vertex of T. To form the cotree for $G+x$, x must then be added to the subtree of T rooted at u. The following notation is used: u is the lowest marked vertex so far examined; w denotes the lowest marked (1) node examined before u; y is a marked (1) node which is not properly marked or a marked (0) node, if either exists in T. Whenever the procedure finds that $G+x$ is not a cograph, an accompanying comment indicates which of the conditions (i)–(vi) from the proof of the theorem holds. When this occurs, we assume the entire algorithm is terminated.

1. (Initialize and check root.)
$y \leftarrow \Lambda$
If R is not marked
then $G+x$ is not a cograph /* condition (iii)
else do if $md(R) \neq d(R)-1$
then $y \leftarrow R$
unmark R; $md(R) \leftarrow 0$;
$u \leftarrow w \leftarrow R$
end
2. (Choose an arbitrary marked vertex u and follow the path from u to w, checking for a legitimate alternating path and unmarking vertices along the path.)
while there are marked vertices remaining in T
do choose an arbitrary marked vertex u
2.1. *if* $y \neq \Lambda$
then $G+x$ is not a cograph /* condition (i) or (ii)
if label $(u)=1$

```
        then do if md(u) ≠ d(u) − 1
                then y ← u
                if parent (u) is marked
                then G + x is not a cograph /* conditions (i) and (vi)
                else t ← parent (parent (u))
             end
        else do y ← u;
                t ← parent (u)
             end
        unmark u; md(u) ← 0
```

2.2. (Now check if the $u-w$ path is part of the legitimate alternating path $u-R$.)

```
        while t ≠ w
        do
          if t = R
          then G + x is not a cograph /* condition (iv)
          if t is not marked
          then G + x is not a cograph /* condition (iii) or (v) or (vi)
          if md(t) ≠ d(t) − 1
          then G + x is not a cograph /* condition (ii)
          if parent (t) is marked
          then G + x is not a cograph /* condition (i)
          unmark t; md(t) ← 0;
          t ← parent (parent (t))
        end
```

2.3. (Reset w for next choice of marked vertex.)

```
        w ← u
      end (step 2)
    end FIND-LOWEST
```

3.3. Example of the algorithm. To illustrate the algorithm, assume we have the cograph G and cotree T of Fig. 1 and consider the graph $G+x$ where x is adjacent to a, d, e and f. When procedure MARK terminates, the following vertices have been marked: a, d, e, f, parent (a), parent (d), R; and the following have been unmarked: a, d, e, f and parent (d); leaving parent (a) and R marked. See Fig. 1. Function FIND-LOWEST *unmarks* R and sets $u \leftarrow w \leftarrow R$ in step 1. Thus in step 2 there is only one remaining marked vertex to examine: $u \leftarrow$ parent (a). Since u is a (1) node with $md(u) = d(u) - 1$ and parent (u) is not marked, the only actions taken in step 2.1 are $t \leftarrow R$ and *unmark* (parent (a)). The loop of step 2.2 is never executed because $t = w = R$. In step 2.3 we set $w \leftarrow$ parent (a) in preparation for the next step 2 loop, but this loop is not repeated in this case since no marked vertices remain. The FIND-LOWEST function terminates with $u =$ parent (a), indicating that $G+x$ is a cograph. Back in step 2.5 of the main procedure, we identify $A = \{a\}$ and $B = \{b\}$. Since u is a (1) node, $|B| = 1$ and $b \in B$ is a leaf, we need only add a new (0) node in place of b and make b and x children of this node. The resulting cotree T' representing cograph $G+x$ is shown in Fig. 2.

Let us also consider the graph G' in Fig. 2 and add vertex z where z is adjacent to a, c, d, e and f. Procedure MARK will *mark* the following: a, c, d, e, f, parent (a), parent (c), parent (d), R; and *unmark* a, c, d, e, f and parent (d); leaving parent (a), parent (c) and R marked. In FIND-LOWEST we *unmark* R and set $u \leftarrow w \leftarrow R$ in step

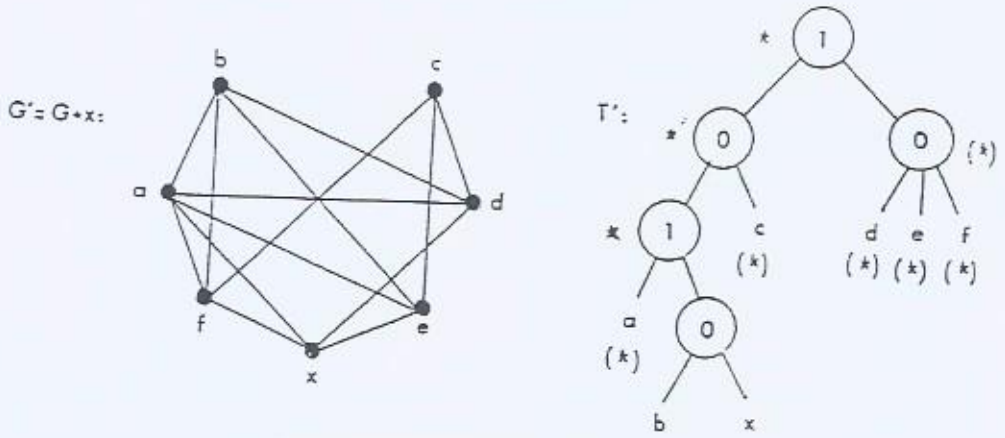

FIG. 2. *Cograph $G' = G + x$ and its cotree T'.*

1. Assume $u \leftarrow$ parent (c) is chosen in step 2. As label $(u) = 0$, we end step 2.1 with $y = u$, $t = R$ and parent (c) unmarked. Step 2.2 is not executed because $w = t = R$ and in 2.3 we set $w \leftarrow$ parent (c). Now $u \leftarrow$ parent (a) and we see in 2.1 that $G' + z$ is not a cograph because $y \neq \Lambda$, indicating in this case that $M \backslash \{\alpha\}$ contains a (0) node (condition (i) of the proof). If $u \leftarrow$ parent (a) is chosen first in step 2, we find that $G' + z$ is not a cograph in step 2.1, where we find that label $(u) = 1$ and parent (u) is marked. Fig. 3 shows the graph $G' + z$ with a $P_4(b, a, z, c)$ indicated.

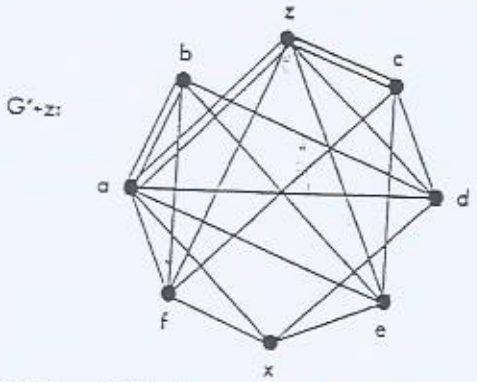

FIG. 3. *The graph $G' + z$ is not a cograph—vertices b, a, z, c form a P_4.*

3.4. Timing analysis. The timing analysis for the algorithm relies on a time bound of $O(\deg(x))$ for the iteration adding x to T, where $\deg(x)$ is the degree of x in $G + x$.

Since all internal nodes of T, except possibly the root, have at least-two children, we know that the MARK procedure will examine only $O(\deg(x))$ nodes. For each of these nodes, the processing is done in constant time, and thus the time bound for procedure MARK is $O(\deg(x))$. It is clear that $|M|$ is also bounded by $O(\deg(x))$, and since FIND-LOWEST examines each marked node once in constant time, the time for this function is $O(|M|) = O(\deg(x))$.

All but one of the tree alterations can be done in constant time. The only cases which may require more than constant time are those where the lowest marked node is a (0) node ((1) node) which has two or more children which have been marked and unmarked (not been marked). In both cases, we are careful to move the children which

were both marked and unmarked, since the cardinality of this set is $O(\deg(x))$ whereas the cardinality of the set of children which were not marked is not similarly bounded. In procedure MARK we have maintained a linked list of the children which were marked and subsequently unmarked, and hence, they can be accessed in time bounded by $O(\deg(x))$. Therefore, all of the tree modifications can be done in $O(\deg(x))$ time.

Thus, we have the required bound for each iteration, implying an overall time bound of $O(m+n)$ for the entire algorithm.

4. Concluding remarks. As noted in § 1, in most typical applications, the graphs encountered may not be cographs but in fact will be very close to being a cograph. It may be necessary to add or delete a few edges in order to destroy all P_4s and thus to achieve a cograph. For this purpose one would want to find as large a subcograph as possible. Not surprisingly this problem is intractable, however using Theorem 1 it is possible to get a good heuristic algorithm. We first note that applying the recognition algorithm to the graph and rejecting any vertex whose addition does not yield a cograph does result in a subcograph. This procedure has two drawbacks. First, it greatly depends on the order in which the vertices are presented. One would expect that a non-increasing degree order would help in obtaining a large subcograph. Secondly, some particularly bad vertex may be included in the subcograph and thereafter cause many other vertices to be rejected. Obviously, a complete backtracking scheme is out of the question; however using Theorem 1 it is possible to develop a limited backtracking procedure. Under this scheme any time a vertex is rejected, a note is made of the existing vertices which cause the rejection. Once an existing vertex has accumulated enough such notes it is removed and other rejected vertices are tried again.

Acknowledgments. The authors wish to thank the Natural Sciences and Engineering Research Council of Canada for financial assistance.

REFERENCES

[1] K. S. BOOTH AND G. S. LUEKER, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 335–379.

[2] S. A. COOK, *The complexity of theorem-proving procedures*, Proc. 3rd Annual ACM Symposium on Theory of Computing, ACM, New York, 1971, pp. 151–158.

[3] D. G. CORNEIL, H. LERCHS AND L. STEWART BURLINGHAM, *Complement reducible graphs*, Disc. Appl. Math., 3 (1981), pp. 163–174.

[4] D. G. CORNEIL AND Y. PERL, *Clustering and domination in perfect graphs*, Disc. Appl. Math., 9 (1984), pp. 27–39.

[5] D. G. CORNEIL, Y. PERL AND L. K. STEWART, *Cographs: recognition, applications and algorithms*, Proc. 15th Southeastern Conference on Combinatorics, Graph Theory and Computing, LSU, 1984, to appear.

[6] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.

[7] C. C. GOTLIEB AND S. KUMAR, *Semantic clustering of index terms*, J. Assoc. Comput. Mach., 15 (1968), pp. 493–513.

[8] J. E. HOPCROFT AND R. E. TARJAN, *Efficient planarity testing*, J. Assoc. Comput. Mach., 21 (1974), pp. 549–568.

[9] R. M. KARP, *On the complexity of combinatorial problems*, Networks, 5 (1975), pp. 45–68.

[10] D. J. ROSE, R. E. TARJAN AND G. S. LUEKER, *Algorithmic aspects of vertex elimination graphs*, this Journal, 5 (1976), pp. 266–283.

[11] J. SPINRAD, *Transitive orientation in $O(n^2)$ time*, Proc. 15th Annual ACM Symposium on Theory of Computing, ACM, New York, 1983, pp. 457–466.

[12] L. STEWART, *Cographs, a class of tree representable graphs*, TR 126/78, Dept. Computer Science, Univ. Toronto, Toronto, Ontario, 1978.

[13] J. VALDES, R. E. TARJAN AND E. L. LAWLER, *The recognition of series parallel digraphs*, this Journal, 11, 2 (1982), pp. 298–313.