SIAM J. DISCRETE MATH.
Vol. 22, No. 4, pp. 1277–1296

A SIMPLE LINEAR TIME LexBFS COGRAPH RECOGNITION ALGORITHM[*]

ANNA BRETSCHER[†], DEREK CORNEIL[†], MICHEL HABIB[‡], AND CHRISTOPHE PAUL[§]

Abstract. Recently lexicographic breadth first search (LexBFS) has been shown to be a very powerful tool for the development of linear time, easily implementable recognition algorithms for various families of graphs. In this paper, we add to this work by producing a simple two LexBFS sweep algorithm to recognize the family of cographs. This algorithm extends to other related graph families such as P_4-reducible, P_4-sparse, and distance hereditary. It is an open question whether our cograph recognition algorithm can be extended to a similarly easy algorithm for modular decomposition.

Key words. graph algorithms, cograph recognition, linear time algorithms, LexBFS, P_4-free graphs, modular decomposition

AMS subject classification. 05C85

DOI. 10.1137/060664690

1. Introduction.

We introduce a new[1] linear time algorithm to recognize *cographs*, namely, graphs with no induced path on four vertices (P_4). Although several optimal cograph recognition algorithms already exist, our algorithm is of interest for several reasons: it is conceptually and practically simpler than previous algorithms; it returns both positive and negative certificates; it is *lexicographic breadth first search* (LexBFS) based, introducing a new variant LexBFS$^-$; it is the foundation of recognition algorithms for other related families of graphs; and finally, it could potentially be generalized to a simple linear time graph modular decomposition algorithm.

Cographs or *complement reducible* graphs have the property of being defined both recursively and by a simple forbidden induced subgraph characterization, namely, no induced P_4 [9]. Cographs are the family of graphs constructed from a single vertex under the closure of the operations of union and complementation or, equivalently, union and join. As shown in [9] these operations uniquely define a tree representation referred to as a *cotree*. The certificate returned by our algorithm is either a cotree representation if the graph is a cograph or an induced P_4.

Cographs arise in applications such as examination scheduling problems [26], automatic clustering of index terms [17], and recently in the recognition of read-once functions [16]. Surprisingly, despite the structural simplicity of cographs, constructing linear time recognition algorithms has been challenging. The first linear time algorithm to recognize cographs and construct the cotree was achieved by Corneil, Perl, and Stewart [10] in 1985. The algorithm examines each vertex in turn, inserts it into the cotree if the graph is a cograph, and returns an induced P_4 otherwise. A

[*]Received by the editors July 10, 2006; accepted for publication (in revised form) March 26, 2008; published electronically July 25, 2008. The authors wish to thank the Natural Sciences and Engineering Council of Canada and the French ANR project *Graph Decompositions and Algorithms* for financial assistance in support of this research.
http://www.siam.org/journals/sidma/22-4/66469.html

[†]Department of Computer Science, University of Toronto, Toronto, ONT M5S 3G4, Canada (bretscher@utsc.utoronto.ca, dgc@cs.toronto.edu).
[‡]LIAFA, CNRS-Université Paris 7-Denis Diderot, Paris Cedex 05, F-75251 France (michel.habib@liafa.jussieu.fr).
[§]LIRMM, CNRS-Université Montpellier II, Montpellier Cedex 5, 34 392, France (paul@lirmm.fr).

[1]A variation of our algorithm, using three LexBFS sweeps instead of two, was outlined in [4].

parallel cograph recognition algorithm appeared in [13]. Recently, Habib and Paul [19] have used vertex splitting to produce a new linear time recognition algorithm for cographs. Despite being optimal, each of these algorithms requires more complex data structures or algorithmic techniques than our new algorithm.

The motivation for studying cographs lies not only in their simplicity but also in their extension to other families and to *modular decomposition*. In fact, our cograph algorithm has already been extended to recognize *distance hereditary*, P_4-*reducible*, and P_4-*sparse* graphs [3]. *Distance hereditary* graphs are characterized by the property that the distance between any two connected vertices of an induced subgraph equals their distance in the original graph. Their relationship to cographs stems from the fact that for any vertex v and integer k, the set of vertices at distance k from v induces a cograph. The family of P_4-*reducible* graphs requires that each vertex belongs to at most one induced P_4, and the P_4-*sparse* graphs are defined by the restriction that any set of five vertices induces at most one P_4. These two graph families belong to the P_4-*heirarchy*, which is a set of graph families defined by various restrictions on the number or types of allowed induced P_4s. Additional families in the P_4-heirarchy, parity graphs and house, hole, domino (HHD)-free graphs, are related to cographs and may also be recognizable by similar algorithms.

There are several methods for *decomposing* or revealing the structure of a graph. A graph is decomposed by splitting the graph into smaller subgraphs in a well-defined manner. The ultimate goal is to decompose a graph into its most basic building blocks or *prime subgraphs*. An important consequence of the decomposition process is the construction of a tree representation of the graph (see, for example, [22]). Such a tree representation of a graph can lead to the simplification of difficult graph related problems, such as transitive orientation [24] and NP-hard problems such as maximum size clique, maximum size stable set, minimum coloring, and minimum clique covering, for restricted families of graphs.

Modular decomposition is one such decomposition scheme. Unfortunately, algorithms for modular decomposition are notorious for being either linear time and impractical or less efficient. The optimal *theoretical* algorithms (see [23, 14, 24, 12]) are linear in time; however, the best *practical* algorithms are only $\mathcal{O}(|V| + |E| \log |V|)$ [14, 20] and $\mathcal{O}(|V| + |E| \cdot \alpha(|V|, |E|))$ [14], where $\alpha(|V|, |E|)$ is the inverse Ackermann function. Recent results are proving hopeful that a simple linear time algorithm may exist for finding the modular decomposition of a graph. The motivation for this discussion lies in the relationship between cographs and modular decomposition.

Cographs are exactly those graphs that are completely decomposable with respect to modular decomposition; i.e., they are the graphs whose prime subgraphs are single vertices. Additionally, the cotree is exactly the modular decomposition tree for cographs. The cotree is algorithmically significant as a tool for dynamic programming algorithms, thereby reducing NP-hard problems such as coloring, clique detection, hamiltonicity, etc., on arbitrary graphs to fast polynomial time problems on cographs [9]. As we will show, our results provide a very natural cotree construction algorithm from two LexBFS orderings of the vertices. Since the algorithm was easily extended to construct tree representations of distance hereditary, P_4-reducible, and P_4-sparse graphs, a natural question is whether there exists a generalized algorithm to construct the modular decomposition tree for any graph. Note that any linear time modular decomposition algorithm, such as [23, 14, 24, 12], immediately yields a somewhat complicated, linear time cograph recognition algorithm.

As previously noted, our algorithm is based on two LexBFS orderings of the vertex set. The real beauty of LexBFS lies in its straightforward implementation and linear running time. Not only can it be implemented in linear time for an input graph G, but using only the edges of G one can also do a LexBFS of the complement of G in time linear in the size of G. Furthermore, more sophisticated vertex selection methods can be easily implemented with minor modifications to the original LexBFS implementation. We will define one of these variations (LexBFS$^-$) in section 3. LexBFS based recognition algorithms fall into a simple algorithmic paradigm of "*order*" the vertex set and "*check*" for a graph characterizing property. The following algorithm overview for our cograph recognition algorithm illustrates the paradigm.

```
Algorithm sketch RECOGNIZE_COGRAPH($G$)
Input: Graph $G$.
Output: Cotree $T$ if $G$ is a cograph, an induced $P_4$ otherwise.
    Compute $\sigma$ a LexBFS of $G$
    Given $\sigma$, use LexBFS$^-$ to compute $\overline{\sigma}^-$, a LexBFS of $\overline{G}$
    if $\sigma, \overline{\sigma}^-$ satisfy the neighborhood subset property (NSP) on $G$, $\overline{G}$, respectively,
        return ( COTREE )
    else
        return ( REPORT_$P_4$ )
end RECOGNIZE_COGRAPH
```

This technique of using LexBFS to construct a vertex ordering that has a graph characterizing property was first introduced by Rose, Tarjan, and Lueker [25] in 1976 for chordal graph recognition. Their algorithm produces a vertex ordering that is then tested for a property specific to chordal graphs, namely, a perfect elimination ordering [25]. More recently, this paradigm has been used to recognize interval graphs [11, 18], unit interval graphs [7, 21], and bipartite permutation graphs [5, 21]. See [6] for a survey of these algorithms as well as other applications of LexBFS.

The remainder of this paper is organized as follows. We begin by defining cograph and cotree specific notation and theory, followed by a discussion of LexBFS and the newly defined variant LexBFS$^-$. The remaining sections discuss the algorithm and prove its correctness. The first of these sections proves the *neighborhood subset property* (NSP), a relationship between the neighborhoods of two sets drawn from the neighborhood and nonneighborhood with respect to a given vertex. This property leads naturally to the recognition algorithm, which is followed by an overview of the implementation details. We conclude by summarizing our results and indicating possible directions for future work.

2. Background.

We will assume standard graph theoretic notation and definitions as in [28]; however, here we will highlight those of particular importance to this paper.

All graphs in this paper are undirected. Let U be a subset of V. Then $V - U$ or $V \setminus U$ is the set of vertices belonging to V and not U. The *cardinality* of U is denoted by $|U|$ and $G[U]$ is the subgraph of G, induced by U. We will denote the complement graph of $G = (V, E)$ by $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{uv | u, v \in V, u \neq v, \text{ and } (u, v) \notin E\}$. The neighborhood $N(v)$, of vertex v, is the set of vertices adjacent to v, and the complement neighborhood $\overline{N}(v)$ is the set of vertices not adjacent to v in G.

We refer to a chordless path on k vertices by P_k. Suppose that $x_1 x_2 \ldots x_{k-1} x_k$ is a P_k. We say that x_1 and x_k are the endpoints of P_k and that $x_2, \ldots, x_{k-1}$ are the midpoints. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *union* of G_1 and G_2 is the graph $G = (V_1 \cup V_2, E_1 \cup E_2)$. The *join* of G_1 and G_2 is the graph $G = (V_1 \cup V_2, E_1 \cup E_2 \cup E)$, where $E = \{u_1u_2 | u_1 \in V_1 \text{ and } u_2 \in V_2\}$.

Given a rooted tree T and two vertices x and y in T, we say that x is an *ancestor* of y, and y is a *descendant* of x, if x lies on the path from y to the root of T. For a set of leaves S of T, we say that the *lowest common ancestor* (lca) of S is the internal node v of T such that v is the root of the smallest rooted subtree of T containing S. We say that the children of a node are *siblings*. A subset $M \subseteq V$ is a *module* of $G = (V, E)$ if for all vertices $x, y \in M$ and $v \in V - M$, $xv \in E$ if and only if $yv \in E$.

2.1. Cographs and cotrees. Figure 1 illustrates a cograph G and an embedding of the corresponding cotree T_G. Leaves of T_G represent the vertex set V, and each internal node signifies the union (0) or join (1) operations on the children. The significance of the 0(1) nodes is captured by the fact that $xy \in E$ if and only if the $lca(x, y)$ is a 1 node, as depicted in Figure 1. For the remainder of this section, assume that G refers to a cograph and T_G to its cotree.

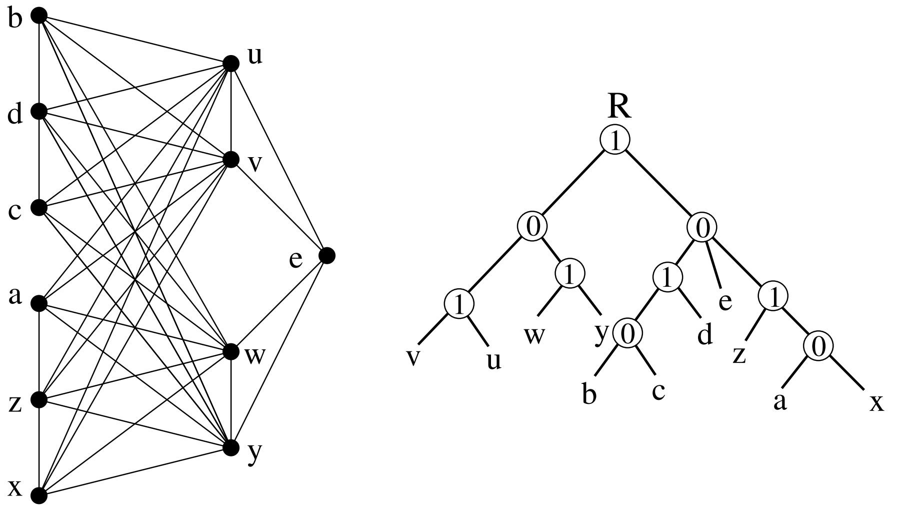

FIG. 1. *(Left) A cograph G. (Right) An embedding of the cotree T_G of G.*

The algorithm relies on the following properties of cographs:

1. *Complement property.* $\overline{G}$ is a cograph; i.e., cographs are closed under complementation.
2. *Complement cotree property.* The cotree $\overline{T}$ of $\overline{G}$ is exactly T with 0 and 1 nodes interchanged.
3. *Hereditary property.* Cographs are hereditary in the sense that any induced subgraph is also a cograph.
4. *Induced subtree property.* An induced rooted subtree t of T is the cotree of an induced subgraph of G.

Generally, given an internal 0 or 1 node u, we let T^u refer to the induced subtree rooted at u. However, we will need to be able to refer to more specific subtrees of a cotree. Consider the root R of T. Let x be a leaf of T. Then P_{xR} will denote the directed path of T from x to R. Note that with the exception of x, the path alternates between 0 and 1 nodes. Let $(0^x_1, \dots 0^x_k)$ (resp., $(1^x_1, \dots 1^x_{k'})$) refer to the 0 nodes (resp., the 1 nodes) from x to R on P_{xR}. Notice that each node 0_i^x or 1_i^x has one child on P_{xR} and at least one child not on P_{xR}.

We now define the subtrees rooted on a path P_{yR} for each y. Consider 0_i^y on P_{yR} and only the children of 0_i^y that do *not* lie on P_{yR}. We can define the set of subtrees T_{0i}^y to be the induced subtrees rooted at these children of 0_i^y. The analogous definition holds for T_{1j}^y and 1_j^y. Figure 2 highlights four such sets, namely, $T_{01}^x, T_{02}^x, T_{11}^x$, and T_{12}^x, for the cotree presented in Figure 1.

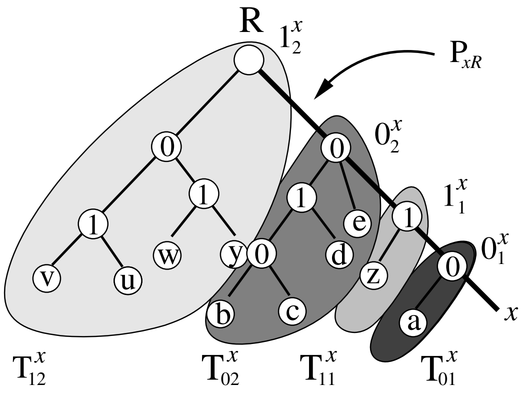

FIG. 2. *Each shaded subtree indicates a different subtree T_{0i}^x or T_{1j}^x rooted on the path P_{xR}.*

We conclude this section with an observation about the induced subtrees T_{0i}^y and T_{1i}^y of a cotree.

OBSERVATION 1. *For any vertex y of a cograph, the leaves of T_{0i}^y (or T_{1i}^y) define a module. More generally, a set M of vertices is a module if and only if there exists a set U of sibling nodes of the cotree such that $M = \cup_{u \in U}\{x \mid x$ is a leaf of $T^u\}$.*

3. LexBFS and LexBFS$^-$.

We begin by describing a lexicographic breadth first search (LexBFS) and then develop the new variant, LexBFS$^-$.

3.1. LexBFS.

For a graph $G = (V, E)$, a *LexBFS* of G, denoted *LexBFS(G)*, is a breadth first search in which preference is given to vertices whose neighbors have been visited *earliest* in the search [25]. We denote this ordering by the function $\sigma : V \to \mathbb{N}$, where $\sigma(y) = i$ indicates that y is the ith vertex of σ and inversely, $\sigma^{-1}(i) = y$. We use $u <_\sigma v$ to indicate that $\sigma(u) < \sigma(v)$.

There are several different paradigms for implementing LexBFS. The term *lexicographic* stems from the original *labeling* paradigm in which vertices pass a label to each unnumbered neighbor. The next vertex is chosen such that it has the lexicographically largest label. Another conceptualization of LexBFS involves pivots and partitioning [18]. The partitioning paradigm will prove more useful for illustrating the variant LexBFS$^-$. Algorithm LEXBFS below is a variation of the version appearing in [18].

The partitioning paradigm considers a pivot (the current vertex) and all unnumbered vertices. A vertex is considered to be visited if it has been a pivot, i.e., numbered. The algorithm begins with a list L of cells initialized to a single cell containing the vertex set. Each step consists of choosing a pivot α in the leftmost (first) cell of L. The pivot is removed from its cell and the unnumbered neighborhood of α is defined to be $N'(\alpha) = \{v | v \in N(\alpha)$ and v unnumbered$\}$. For each cell $P \in L$, all

```
Algorithm LexBFS($G, \tau$)
Input: A graph $G = (V, E)$ and an initial ordering $\tau$ of the vertices.
Output: An ordering $\sigma$ of the vertices of $G$.
1.  $L \leftarrow (V)$
2.  $i \leftarrow 1$
3.  while $\exists P_i \neq \emptyset$ in $L = (P_1, \dots, P_k)$ do
4.      Let $P_l$ be the leftmost nonempty cell
5.      Remove the first vertex $x$ (smallest with respect to $\tau$) from $P_l$
6.      $\sigma(x) \leftarrow i$
7.      $i \leftarrow i + 1$
8.      for each cell $P_j, j \geq l$ do
9.          Let $P' = \{v | v \in N(x) \cap P_j\}$;
10.         if $P'$ is nonempty and $P' \neq P_j$, then
11.             Remove $P'$ from $P_j$
12.             Insert $P'$ to the left of $P_j$ in $L$
13.     end for
14. end while
15. return ($\sigma$)
end LexBFS
```

members of $N'(\alpha)$ belonging to P are removed from P and inserted into a new cell P' positioned in L immediately to the left of P.

For example, if x is the first vertex selected by a LexBFS on G (see Figure 1 and Table 3.1), then the vertex set is split into two cells $\boxed{\text{y u v w z}}$ and $\boxed{\text{d e c a b}}$, where $\boxed{\text{y u v w z}}$ is moved ahead of $\boxed{\text{d e c a b}}$ since each vertex in $\{y, u, v, w, z\}$ is adjacent to x, whereas each vertex in $\{d, e, c, a, b\}$ is not adjacent to x.

The LexBFS terminates when all vertices have been visited. It outputs an ordering σ of the vertices. The numbering convention in this paper follows that of [11], where the ordering σ is the order in which vertices are visited during the LexBFS. Throughout the paper, when we refer to the "leftmost vertex" or "first" vertex in a set, the ordering is with respect to σ. Table 3.1 walks through a LexBFS of the cograph of Figure 1.

The following lemma is a very useful characterization of LexBFS orderings.

LEMMA 3.1 (see [15, 1]). Four point condition. *An ordering σ is a LexBFS of G if and only if for any $x <_\sigma y <_\sigma z$ such that $xz \in E$ but $xy \notin E$, there exists v such that $v <_\sigma x$, $vy \in E$, and $vz \notin E$.*

In addition, when restricted to cographs, LexBFS orderings can be shown to be *umbrella-free*. An *umbrella* of an ordering σ is a triple of vertices x, y, z such that $x <_\sigma y <_\sigma z$, where $xz \in E$, $xy \notin E$, and $yz \notin E$.

LEMMA 3.2. *Any LexBFS σ of a cograph G is umbrella-free (there does not exist $x <_\sigma y <_\sigma z$ such that $xz \in E$, $xy \notin E$, and $yz \notin E$).*

Proof. Let x, y, z be an umbrella such that x is leftmost in σ. By the four-point condition, there exist $u <_\sigma x$ such that $uy \in E$ but $uz \notin E$. Note that $ux \notin E$; otherwise, $\{u, x, y, z\}$ would induce a P_4. But then u, x, y is also an umbrella, contradicting the choice of x. ∎

We now introduce the notion of *slices*. Notice that when a pivot is selected from the leftmost cell, *any* vertex in the cell could be the pivot as each vertex in the cell

TABLE 3.1
Step by step LexBFS of G from Figure 1. The resulting ordering is $\sigma:$ ***x y w z u v a d c b e***.

$\sigma(\alpha)$	α	$N'(\alpha)$	Cells
			$\boxed{\text{x d y u e v w c a z b}}$
1	x	{y u v w z}	$\boxed{\text{y u v w z}}\ \boxed{\text{d e c a b}}$
2	y	{w z d e c a b}	$\boxed{\text{w z}}\ \boxed{\text{u v}}\ \boxed{\text{d e c a b}}$
3	w	{z d e c a b}	$\boxed{\text{z}}\ \boxed{\text{u v}}\ \boxed{\text{d e c a b}}$
4	z	{u v a}	$\boxed{\text{u v}}\ \boxed{\text{a}}\ \boxed{\text{d e c b}}$
5	u	{v a d e c b}	$\boxed{\text{v}}\ \boxed{\text{a}}\ \boxed{\text{d e c b}}$
6	v	{a d e c b}	$\boxed{\text{a}}\ \boxed{\text{d e c b}}$
7	a	{ }	$\boxed{\text{d e c b}}$
8	d	{c b}	$\boxed{\text{c b}}\ \boxed{\text{e}}$
9	c	{ }	$\boxed{\text{b}}\ \boxed{\text{e}}$
10	b	{ }	$\boxed{\text{e}}$
11	e	{ }	

has the same set of numbered neighbors. We say that these vertices form a *slice*. The following definitions formalize this notion.

Given an ordering σ of the vertices, for each vertex x, we can define subsets of the neighborhood of x,

$$N_i(x) = \{u \mid u \in N(x) \text{ and } \sigma(u) < i\},$$

which contain the neighbors of x appearing before the ith vertex in the ordering σ. When the ith pivot of a LexBFS is selected, $N_i(x)$ is the same for each vertex x in the leftmost cell P_l (the following definition establishes that these vertices form a slice).

DEFINITION 3.3. *A* slice *(or set of tied vertices) of a LexBFS ordering σ is a set of consecutive vertices $S = \{u \mid i \leqslant \sigma(u) \leqslant j\}$ such that for any $u \in S$, $N_i(u) = N_i(\sigma^{-1}(i))$, and for any v, $j < \sigma(v)$, or $\sigma(u) < i$, $N_i(v) \neq N_i(\sigma^{-1}(i))$.*

Examples of slices of a LexBFS are the whole set V and the neighborhood of the first vertex visited by the search. Notice that there are exactly $|V|$ slices, one for each vertex of the search, or one for each occurrence of a "leftmost" cell in the LexBFS procedure. Consider the LexBFS obtained in Table 3.1. The different bracket styles indicate the slices constructed during the sweep and highlight those at a common nesting depth:

$$\sigma:\ [\mathbf{x}(\mathbf{y}\{\mathbf{w}\langle\mathbf{z}\rangle\}\{\mathbf{u}\langle\mathbf{v}\rangle\})(\mathbf{a})(\mathbf{d}\{\mathbf{c}\langle\mathbf{b}\rangle\}\{\mathbf{e}\})]. \tag{3.1}$$

As shown by the following lemma, slices of a LexBFS ordering exhibit nice structural properties.

LEMMA 3.4. *Let S be a slice of a LexBFS σ of a graph G. Then σ restricted to the vertices of S is a LexBFS of the subgraph induced by those vertices.*

Since slices form the foundation of the recognition algorithm, we will need notation to refer to any slice of a LexBFS. The slice starting at vertex x will be denoted by $S(x)$. A *subslice* is a slice nested within a slice. In the following, some of the identified subslices may be empty. Notice that $S(x) \cap N(x)$ is a subslice of $S(x)$ which we denote by $S^A(x)$. Once the final vertex of $S^A(x)$ has been used as a pivot, the remaining vertices of $S(x)$ are grouped into cells $S(x) \cap \overline{N}(x) = S_1(x), S_2(x), \ldots,$ where all vertices in $S_i(x)$ have the same neighborhood in $S^A(x)$. The $S_i(x)$ are called *x-cells.* We let x_i be the first vertex, as chosen by σ, in $S_i(x)$. Note that $S_1(x)$ is a slice in

σ, whereas $S_j(x), j \geq 2$, could be the union of slices since there may be some edges from vertices in some $S_i(x), i < j$, to vertices in S_j. Since x_i is identified by σ, the four-point condition immediately establishes the following observation.

Observation 2. *Consider v-cell $S_j(v)$ as identified by σ. If some vertex in $S_j(v)$ other than v_j has a neighbor in $S_i(v)$ for $i < j$, then v_j also has a neighbor in $S_h(v), h \leq i$. Thus to find the leftmost vertex in $\overline{N}(v)$ that is adjacent to some vertex in $S_j(v)$, we have only to look at v_j's neighbors.*

For cographs, each $S_i(x)$ is a maximal slice of $S(x)$, since, as shown in Lemma 3.6 below, any edge between two x-cells in $S(x) \cap \overline{N}(x)$ implies the existence of a P_4. Equation (3.2) illustrates the subslices of $S(x)$ as defined by the LexBFS ordering of Table 3.1. (The $S_i(x)$ are slices since G is a cograph.) In particular, $S^A(x) = \{y, w, z, u, v\}$ and $S^N(x) = \langle S_1(x), S_2(x)\rangle$, where $S_1(x) = \{a\}$ and $S_2(x) = \{d, c, b, e\}$:

$$\mathbf{x}\ [\mathbf{y\ w\ z\ u\ v}]\ [\mathbf{a}]\ [\mathbf{d\ c\ b\ e}]. \tag{3.2}$$

Again it is noted that in a LexBFS ordering, the slice $S(x)$ is defined for each vertex x and that the sets $S^A(x)$ and $S^N(x)$ may be empty sets. Consider a, u, and c in (3.1); $S(a) = \{a\}$, so $S^A(a) = \emptyset$ and $S^N(a) = \emptyset$; $S^N(u)$ is empty in $S(u)$, while for vertex c the set $S^A(c)$ is empty.

Definition 3.5. *If S is a consecutive set of vertices in a LexBFS ordering σ and x is the first vertex of S, then the left neighborhood of S is*

$$N_<(S) = \{y | y <_\sigma x \text{ and } y \in N(z)\ \forall z \in S\}.$$

Lemma 3.6. *Let $G = (V, E)$ be a cograph and σ a LexBFS sweep of G. Then for any vertex v and $i < j$*

$$\forall x \in S_i(v), \forall y \in S_j(v), xy \notin E \text{and} \tag{3.3}$$

$$N_<(S_i(v)) \supset N_<(S_j(v)). \tag{3.4}$$

Proof. We begin by showing that for any $x \in S_i(v)$ and $y \in S_j(v)$, $xy \notin E$. Assume that $xy \in E$ and i is the smallest index such that there exist $j > i$, $y \in S_j(v)$, and $xy \in E$. As $y \notin S_i(v)$ there exists $u \in S^A(v)$ such that $ux \in E$ and $uy \notin E$. As both x and y are nonadjacent to v, $\{v, u, x, y\}$ induces a P_4, contradicting that G is a cograph, and thus (3.3) holds. Finally, Lemma 3.2 and (3.3) show that there does not exist $u <_\sigma x$ such that $uy \in E$ but $ux \notin E$, thereby proving (3.4). □

A property similar to Lemma 3.6, stated differently, has been observed in [13].

3.2. LexBFS$^-$.

In the above implementation of LexBFS, Algorithm LexBFS selects the pivot x from the leftmost cell, and for each cell P_j, the vertices in $N(x) \cap P_j$ are removed from P_j and inserted into a new cell P' that is inserted to the *left* of P_j. Notice that inserting P' to the *right* of P_j is equivalent to inserting the *nonneighbors* in P_j, $\overline{N}(x) \cap P_j$, to the left of the neighbors of x in P_j, i.e., equivalent to performing a LexBFS of the complement of G. Thus, a LexBFS of $\overline{G}$ can be accomplished in linear time *without* computing $\overline{G}$, i.e., using only the edge set of G and Algorithm LexBFS with a small modification to line 12, namely,

$$12^- \text{Insert } P' \text{ to be the right of } P_j \text{in } L.$$

A *LexBFS minus*, or LexBFS$^-$, of a graph G is a modified version of a LexBFS of $\overline{G}$. The modification arises in how the pivots are selected. The idea is to have an initial LexBFS ordering τ of G. LexBFS$^-(G, \tau)$ does a LexBFS of $\overline{G}$ *and* selects the next pivot to be the vertex in the leftmost cell that was numbered earliest in τ. To implement this "*tie-breaking mechanism*," simply pass τ as the initial ordering of the vertex set. Notice that the leftmost vertex in the leftmost cell *is* the vertex that was numbered earliest in τ. Therefore, a LexBFS$^-(G, \tau)$ can easily be implemented in linear time with minor modifications to Algorithm LexBFS. It follows that a minus sweep on $\overline{G}$ can also be accomplished in linear time. Such an ordering resulting from LexBFS$^-(G, \sigma)$ will be denoted $\overline{\sigma}^-$.

The slice-cell notation for a LexBFS ordering $\overline{\sigma}$ of $\overline{G}$ follows that of a LexBFS of G. We will denote the slice starting at vertex x and its subslice of adjacent vertices by $\overline{S}(x)$, $\overline{S^A}(x)$, respectively. The remaining x-cells are $\overline{S^N}(x) = \langle \overline{S_1}(x), \dots \overline{S_k}(x) \rangle$. For example, given the graph G in Figure 1 and σ : *xywzuvadcbe*, LexBFS$^-$ yields

$$\overline{\sigma}^- : \mathbf{x}\ [\mathbf{a\ d\ e\ c\ b}\ ]\ [\mathbf{z}\ ]\ [\mathbf{y\ u\ v\ w}\ ],$$

where $\overline{S^A}(x) = \{a, d, e, c, b\}$ and $\overline{S_1}(x) = \{z\}$, $\overline{S_2}(x) = \{y, u, v, w\}$.

The next lemma highlights an important observation about modules and LexBFS$^-$ sweeps.

Lemma 3.7. *Let M be a module of a graph G, let σ be a LexBFS of G, and let $\overline{\sigma}^-$ be the ordering resulting from LexBFS$^-(G, \sigma)$. The first vertex of M in σ is v if and only if v is the first vertex of M in $\overline{\sigma}^-$. Furthermore, $S(v)$ and $\overline{S}(v)$ are the inclusion minimal slices of σ, respectively, $\overline{\sigma}^-$, containing M.*

Proof. Let v be the first vertex of M visited by $\overline{\sigma}^-$. At the stage v is selected in $\overline{\sigma}^-$, M is contained in the slice $\overline{S}(v)$. Since v breaks the tie, v must be the first vertex of $\overline{S}(v)$ to appear in σ and therefore the first vertex of M to appear in σ.

Assume v is the first vertex of M in σ. Let $\overline{S}$ be the inclusion minimal slice of $\overline{\sigma}^-$ containing M, and let u be the first vertex of $\overline{S}$. If $u \neq v$, then $u \in \overline{S} \setminus M$. But this implies that there exists a slice $\overline{S}' \subseteq \overline{S} \setminus \{u\}$ containing M, contradicting the minimality of $\overline{S}$. ∎

Motivated by Lemma 3.6, we wish to have a property appearing in σ or $\overline{\sigma}^-$ that indicates the presence of a P_4 in G. To set the stage for this, we are interested in whether the first vertex of a specific P_4 in a given LexBFS is an endpoint or a midpoint of the P_4. In particular, note that a LexBFS of a P_4 starting at a midpoint of the P_4 satisfies the conditions of Lemma 3.6, whereas a LexBFS starting at an endpoint does not. First, we prove a result about the presence of P_4s in an arbitrary LexBFS.

Lemma 3.8. *Let $S(v)$ be the minimal slice of LexBFS σ that contains P_4 p. If v does not belong to any P_4s in G, then $p \cap S^A(v)$ consists of the midpoints of p.*

Proof. By the minimality of $S(v)$, we know that v is not universal to p, since $S^A(v)$ is a slice of $S(v)$. If v is not adjacent to any vertices in p, then, again by the minimality of $S(v)$, there is some vertex $x \in S^A(v)$ that is adjacent to some, but not all, vertices in p. Thus there is an edge $yz \in p$ such that $xy \in E$, $xz \notin E$; but now we have a P_4 on v, x, y, z, contradicting v not belonging to a P_4. Thus v is adjacent to some, but not all, vertices of p. Now it is easy to check that for v to avoid being in a P_4, v must be adjacent to the midpoints of p, as required. ∎

We now use this lemma to study the relative starting points of a P_4 p in σ and of $\overline{p}$ in $\overline{\sigma}^-$.

Corollary 3.9. *Let σ be an arbitrary LexBFS of graph G, and let vertex w be the first vertex of σ that is in a P_4. Let p be an arbitrary P_4 that contains w. If w is an endpoint of p, then in $\overline{\sigma}^-$, the first vertex of $\overline{p}$ is a midpoint of $\overline{p}$.*

Proof. Let $\overline{S}(v)$ be the minimal slice of $\overline{\sigma}^-$ containing $\overline{p}$. If $v \in \overline{p}$, then $v = w$ and the corollary clearly holds. Otherwise, since w is the leftmost vertex in σ that belongs to a P_4, and $w \in \overline{S}(v)$, we know that $v <_\sigma w$ and thus v does not belong to a P_4. The result now follows by applying Lemma 3.8 to $\overline{\sigma}^-$. In particular, all such P_4s $\overline{p}$ in $\overline{S}(v)$, regardless of whether w is an endpoint of p, have a midpoint as their first vertex is $\overline{\sigma}^-$. ∎

The previous results deal only with the "first" P_4s. We now show that these results cannot be generalized to all P_4s. For example, not all P_4s with an endpoint as the first vertex in σ have a midpoint as the first vertex of the complement P_4 in $\overline{\sigma}^-$, as demonstrated by C_5 123451, σ : 1 2 5 3 4, and $\overline{\sigma}^-$: 1 3 4 5 2. The P_4 : 2345 in G and the P_4 : 3524 in $\overline{G}$ have endpoints 2 and 3, respectively, as the first vertex in σ and $\overline{\sigma}^-$, respectively. Futhermore, it is possible for a P_4 to have both endpoints as midpoints of p and $\overline{p}$, respectively. To see this, consider a "bull" (a vertex v adjacent to the midpoints of a P_4) and a LexBFS starting at v. Note that the "bull" is self-complimentary.

4. Correctness of the algorithm.

Recall that in the cograph recognition algorithm overview given previously, the algorithm consists of a LexBFS sweep σ, followed by a LexBFS$^-$ of G using σ as the input vertex ordering. A certificate in the form of a cotree or an induced P_4 is returned. This section first presents a characterization of cographs in terms of a property of σ and $\overline{\sigma}^-$. We then consider the case when the input graph G does not satisfy this property and a P_4 is returned, followed by the case where G does satisfy the property and a cotree is constructed. We show how LexBFS orderings can be used to characterize the family of cographs.

4.1. The neighborhood subset property.

Motivated by Lemma 3.6, this section introduces the neighborhood subset property (NSP) and a related theorem characterizing cographs.

DEFINITION 4.1. *The* local neighborhood *of a v-cell $S_i(v)$, denoted by $N^l(S_i(v))$, of a slice S(v) for a LexBFS σ is the set of vertices in $S(v)$ before v_i adjacent to at least one vertex in $S_i(v)$.*

Notice that in the case of cographs, it directly follows from Lemma 3.6, (3.3), that the above definition can be restricted.

OBSERVATION 3. *If σ is a LexBFS of a cograph, then $N^l(S_i(v)) = N_<(S_i(v)) \cap S^A(v)$.*

DEFINITION 4.2. *A LexBFS sweep satisfies the NSP if and only if*

$$\forall v \in V, \forall i < j, \text{ such that } S_j(v) \neq \emptyset,\ N^l(S_i(v)) \supset N^l(S_j(v)).$$

As pointed out in the previous section, a LexBFS of a P_4 starting at a midpoint of the P_4 does not violate the NSP. However, as shown in the following theorem, by looking at both σ and $\overline{\sigma}^-$ we are guaranteed to find a P_4 that does violate the property.

THEOREM 4.3 (neighborhood subset theorem). *Let σ be a LexBFS ordering of a graph $G = (V, E)$, and let $\overline{\sigma}^-$ be the sweep LexBFS$^-(G, \sigma)$. Then G is a cograph if and only if σ and $\overline{\sigma}^-$ satisfy the NSP.*

Proof. ($\Rightarrow$) Assume that G is a cograph. Lemma 3.6 and Observation 3 prove that σ satisfies the NSP. As $\overline{G}$ is a cograph, applying Lemma 3.6 and Observation 3 on $\overline{G}$ and $\overline{\sigma}^-$ proves that the NSP holds for $\overline{\sigma}^-$.

($\Leftarrow$) Assume that G is not a cograph; i.e., there exists an induced P_4. First notice that the complement $\overline{p}$ of a P_4 p is a P_4 and that the midpoints of p are precisely the endpoints of $\overline{p}$. The following properties are the core arguments of the proof.

Claim 1. *Let $p = abcd$ be an induced P_4 of G. Notice that the endpoints are a and d. If $S(a)$ contains p, then there exists a pair of integers i, j such that $S_i(a)$ and $S_j(a)$ fail the NSP.*

Proof. Since a is adjacent to b but not adjacent to c or d, $b \in S^A(a)$ and there exists a pair of integers i, j such that $c \in S_i(a)$ and $d \in S_j(a)$. Since b is adjacent to c but not to d, i and j are distinct integers. If $c <_\sigma d$ (i.e., $i < j$) as $cd \in E$, $bc \in E$, and $bd \notin E$, $N^l(S_i(a)) \not\supset N^l(S_j(a))$. Otherwise, $d <_\sigma c$ (i.e., $j < i$), and the four-point condition applied on $b <_\sigma d <_\sigma c$ implies the existence of a vertex $x \in S^A(a), x <_\sigma b$, such that $xd \in E$ and $xc \notin E$. Thereby $N^l(S_j(a)) \not\supset N^l(S_i(a))$, failing the NSP. □

Claim 2. *Let $p = abcd$ be an induced P_4 of G. If $S(v)$ with v not in any P_4 is the inclusion-minimal slice of LexBFS σ containing p, then there exists a pair of integers i, j such that $S_i(v)$ and $S_j(v)$ fail the NSP.*

Proof. By Lemma 3.8, $p \cap S^A(v) = \{b, c\}$; thus $a \in S_i(v)$ and $d \in S_j(v)$. Finally, since $b \in N(a) \setminus N(d)$ and $c \in N(d) \setminus N(a)$, the local neighborhoods $N^l(S_i(v))$ and $N^l(S_j(v))$ fail the NSP. □

We now return to the proof that if G is not a cograph, then the NSP fails. Since G is not a cograph, there exists a P_4 p in G or, equivalently, $\overline{p}$ in $\overline{G}$. We let w be the leftmost vertex in σ that is in some P_4 and let $p = abcd$ be any P_4 containing w. Examine $\overline{p}$ in $\overline{G}$ and let $\overline{S}(v)$ be the inclusion-minimal slice containing $\overline{p}$. Three distinct cases have to be considered:

1. *Vertex v is an endpoint of $\overline{p}$.* Then Claim 1 applies.
2. *Vertex v is a midpoint, say, a, of $\overline{p}$.* Thus a is the leftmost vertex of p in σ and is an endpoint of p. Let $S(u)$ be the inclusion minimal slice of σ containing p. If $u = w = a$, Claim 1 applies; otherwise, by the choice of w, u is not in any P_4, and Claim 2 applies.
3. *Vertex v does not belong to $\overline{p}$.* By the LexBFS$^-$ rule, v is to the left of $\{a, b, c, d\}$ in σ and thus is not in any P_4, by our choice of p. Thus Claim 2 applies. □

We now show that the NSP for σ can be tested in linear time.

Lemma 4.4. *Given the list of local neighborhoods, $N^l(S_i(v))$, $i > 0$, for each vertex v, sorted with respect to the same ordering (say, σ), the NSP of σ can be tested in $\mathcal{O}(n + m)$.*

Proof. For each vertex v and for each i, first check that $|N^l(S_{i+1}(v))| < |N^l(S_i(v))|$; if not, report failure. Otherwise, simultaneously walk the lists $N^l(S_i(v))$ and $N^l(S_{i+1}(v))$. If the current vertex x of $N^l(S_{i+1}(v))$ is not the current vertex of $N^l(S_i(v))$, then walk $N^l(S_i(v))$ until x is reached. If x is not reached, then the inclusion $N^l(S_{i+1}(v)) \subset N^l(S_i(v))$ is not satisfied and failure is reported. If $N^l(S_{i+1}(v))$ is walked without reporting failure, then report success.

Since each local neighborhood is searched at most twice, the above algorithm runs in time linear in the total size of the local neighborhoods. Since there are at most n slices, there are at most n local neighborhoods. The size of each local neighborhood of a v-cell $S_i(v)$ is bounded by the number of "left" edges from vertices in $S_i(v)$. Therefore, summing the sizes of all the neighborhood sets results in $\mathcal{O}(n+m)$ complexity. □

4.2. Reporting a P_4 from σ or $\overline{\sigma}^-$ (when G fails the NSP).

We now consider the certificate returned by the algorithm, an induced P_4, when the input graph is not a cograph.

Lemma 4.5. *Let σ be a LexBFS ordering failing the NSP for v-cells of $S^N(v)$. If j is the smallest integer such that $S_j(v)$ and $S_{j+1}(v)$ fail the NSP, then one of the* *following induced P_4s is in G:*

$$vwyv_{j+1} \quad \text{or } v_jwyv_{j+1} \quad \text{or } yvwv_j,$$

where v_i is the first vertex chosen by σ in $S_i(v)$, $w <_\sigma v_j, w \in (N(v_j) \setminus N(v_{j+1})) \cap S^A(v)$, and $y <_\sigma v_{j+1}, y \in (N^l(v_{j+1}) \setminus N^l(v_j))$.

Proof. By the definition of v-cells, there exists $w \in S^A(v)$ such that $w \in N(v_j) \setminus N(v_{j+1})$. We now show that there exists $y <_\sigma v_{j+1}$ such that $y \in N^l(v_{j+1}) \setminus N^l(v_j)$. Since $S_j(v), S_{j+1}(v)$ fail the NSP, there exists $x \in S_{j+1}(v)$ adjacent to $z \in N^l(v_{j+1}) \setminus N^l(v_j)$. If $v_{j+1}z \in E$, then set $y = z$. Otherwise, by the four-point condition on z, v_{j+1}, x, there exists $z' <_\sigma z$ such that $z'v_{j+1} \in E, z'x \notin E$. Thus by the definition of $S_{j+1}(v)$, $z' \in \overline{N}(v)$. If $z' \in S_j(v)$, then set $y = z'$. If $z' \in N^l(v_j)$ and thus $z' <_\sigma v_j$, we have contradicted the choice of $j, j+1$. Without loss of generality, we assume y is the *rightmost* such vertex. There are two cases; y is either a neighbor of v or a nonneighbor.

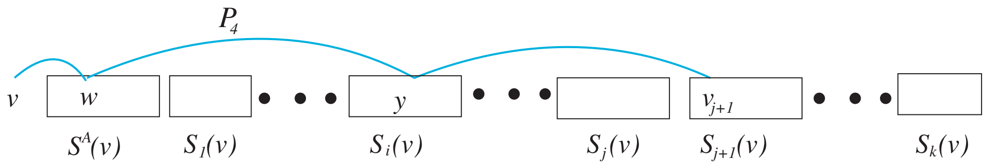

FIG. 3.

1. *y is a nonneighbor of v.* There exists $i < j+1$ such that $y \in S_i(v)$ (see Figure 3). Since j is minimum, w adjacent to v_j implies that w is also adjacent to y. Note that v is a neighbor of w; v is not a neighbor of y or v_{j+1}; w is a neighbor of y but not v_{j+1}; and y is a neighbor of v_{j+1}. Hence, $\{v, w, y, v_{j+1}\}$ is an induced P_4.
2. *y is a neighbor of v.* Then $y \in S^A(v)$, and since y is rightmost, v_j and v_{j+1} are not adjacent (see Figure 4). It follows that the subgraph induced by the set of vertices $\{v, w, y, v_j, v_{j+1}\}$ contains an induced P_4. More precisely, if w and y are adjacent, v_jwyv_{j+1} is an induced P_4; otherwise v_jwvyv_{j+1} is an induced P_5, containing the induced P_4 $yvwv_j$, thereby completing the proof. ▯

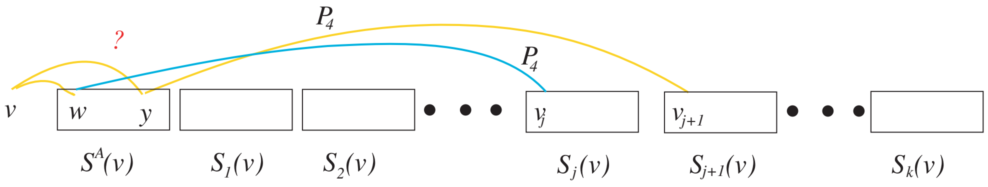

FIG. 4.

We now have the following simple algorithm Report_P_4 to output an induced P_4 once the neighborhood subset test returns a vertex v and an index $j \geq 1$ such that the first pair of v-cells failing the NSP are $S_j(v)$ and $S_{j+1}(v)$. Note that our description is for σ only; see the comments at the end of section 5 for the minor modifications needed to produce a P_4, if the NSP failure is in $\overline{\sigma}^-$.

```
Algorithm Report_P_4($\sigma, \overline{\sigma}^-$)
Input: Vertex $v$ and smallest index $j$ such that $S_j(v), S_{j+1}(v)$ fail the NSP in $\sigma$
Output: An induced $P_4$.
    Choose $w \in (N^l(v_j) \setminus N^l(v_{j+1})) \cap S^A(v)$
    Choose $y$ to be the rightmost vertex such that $y \in N^l(v_{j+1}) \setminus N^l(v_j)$
    if $yv \notin E$
            return ( $vwyv_{j+1}$ )
    else if $wy \in E$
        return ( $v_jwyv_{j+1}$ )
    else
        return ( $yvwv_j$ )
end Report_P_4
```

LEMMA 4.6. *Reporting a P_4, given a pair of v-cells $S_j(v)$ and $S_{j+1}(v)$ of σ failing the NSP, takes at most $\mathcal{O}(\Delta)$ time, where Δ is the maximum degree of the graph.*

Proof. Assume that the local neighborhoods are sorted according to the LexBFS ordering σ. To locate vertices $w \in (N^l(v_j) \setminus N^l(v_{j+1})) \cap S^A(v)$ and $y \in N^l(v_{j+1}) \setminus N^l(v_j)$, simply walk the local neighborhoods $N^l(v_j)$ and $N^l(v_{j+1})$, each of which are no larger than Δ. The adjacency test between y and w takes Δ time simply by walking the adjacency list of w or of y. □

4.3. Constructing the cotree from $\sigma, \overline{\sigma}^-$ (when G satisfies the NSP). We now assume that the graphs we are operating on are cographs. We will refer to an arbitrary cograph as G and its cotree as T with root R. We begin by relating slices $S_i(x)$ to modules of G and subtrees T^x_{0i} of G. This relationship results in a simple cotree construction algorithm.

Let x be an arbitrary vertex of G and consider the LexBFS(G) starting at x, resulting in σ, and the ordering $\overline{\sigma}^-$ produced by LexBFS$^-(G, \sigma)$. The following observation follows directly from the neighborhood subset theorem.

OBSERVATION 4. *Each $S_i(x)$ and $\overline{S}_j(x)$ induces a module in G.*

Furthermore, there is a well-defined relationship between the leaves of an induced subtree in T and the vertices in each $S_i(x)$ or $\overline{S}_j(x)$.

LEMMA 4.7. *Let x be the first vertex of a LexBFS σ of G. For any i, the slice $S_i(x)$ contains exactly the leaves of T^x_{0i}.*

Proof. Consider σ and the stage of the sweep where x and each vertex of $S^A(x)$ have been numbered. Since x is the first vertex of σ, only nodes adjacent to x have been numbered. If a vertex in $S^A(x)$ is adjacent to the leaves of T^x_{0i}, then by the neighborhood subset theorem, the vertex is adjacent to all leaves in T^x_{0j}, where $j < i$; see Figure 2. Therefore, the leaves of T^x_{01} must belong to the slice $S_1(x)$. By the neighborhood subset theorem, no leaf in T^x_{0i} is adjacent to any leaf in $T^x_{0j}, j \neq i$. Consequently, we can use the same argument to show that the next slice $S_2(x)$ contains exactly the leaves of T^x_{02} and, inductively, that the ith slice $S_i(x)$ contains the leaves of the subtree T^x_{0i}. □

COROLLARY 4.8. *Let x be the first vertex of a LexBFS σ of a cograph G, and let τ be any LexBFS of $\overline{G}$ that starts at x (for example, $\overline{\sigma}^-$). Then, in τ, for any j, $\overline{S}_j(x)$ contains exactly the leaves of T^x_{1j}.*

Proof. Since the cotree of $\overline{G}$ is exactly the cotree of G with 1 and 0 nodes interchanged, Lemma 4.7 applied to τ implies that $v \in \overline{S}_i(x)$ if and only if $v \in$

T^x_{1i}. □

Lemma 4.7 and Figure 2 motivate the following corollary.

COROLLARY 4.9. *Given σ and $\overline{\sigma}^-$ of a cograph G, if x is the first vertex of σ, then the path P_{xR} can be reconstructed from the sequences $S^N(x)$ and $\overline{S^N}(x)$.*

Proof. Lemma 4.7 applied to σ and to $\overline{\sigma}^-$ shows that each $S_i(x)$ and $\overline{S}_i(x)$ slice defines the set of leaves of each T^x_{0i} and T^x_{1i} subtree. To embed P_{xR}, it remains only to determine which of 0^x_1 and 1^x_1 is the parent of the leaf x in the cotree. Let u (resp., v) be the first vertex of $S_1(x)$ (resp., $\overline{S_1}(x)$). It is straightforward to see that 0^x_1 is the parent of leaf x if and only if u is adjacent to v. □

These results, along with the hereditary property of cographs, motivate the following preliminary recursive cotree construction algorithm.

```
Algorithm TREE (module $M$, $G$)
Input: A module $M$ of cograph $G$.
Output: A Cotree $T$ of $M$.
  if $M = \{v\}$
    return ( $\{v\}$ )
  for any $v \in M$, compute:
    $[v, S^A(v), S_1(v), \ldots, S_k(v)]$ with respect to $\sigma$, an arbitrary LexBFS of $G[M]$
    $[v, \overline{S^A}(v), \overline{S}_1(v), \ldots, \overline{S}_h(v)]$ with respect to $\overline{\sigma}^-$, the LexBFS$^-$ of $\overline{G}[M]$
    with respect to $\sigma$
  if $S^A(v) = \emptyset$ ($\overline{S^A}(v)$, resp.),
    return ( $v⓪\text{TREE}(S_1(v), G)$ ) (resp., ( $v①\text{TREE}(\overline{S}_1(v), G)$ ))
  else for any $a \in S_1(v)$ and $b \in \overline{S}_1(v)$
    if $ab \in E$, then
($\star$)     return ( $v⓪\text{TREE}(S_1(v), G)①\text{TREE}(\overline{S}_1(v), G)⓪\text{TREE}(S_2(v), G)\ldots$ )
    else
($\star\star$)    return ( $v①\text{TREE}(\overline{S}_1(v), G)⓪\text{TREE}(S_1(v), G)①\text{TREE}(\overline{S}_2(v), G)\ldots$ )
```

Note that the notation "$x①\ T_1\ ⓪\ T_2\ldots$" is parsed as "$((x①T_1)\ ⓪\ T_2)\ldots$" and means that x and T_1 are children of a 1 node, this 1 node and T_2 are the children of a 0 node, etc. In addition, we point out that if T_1 is rooted at a 1 node, the ① operation will *merge* the two 1 nodes together and likewise for the 0 node case. The correctness of the algorithm follows directly from Lemma 4.7, Observation 4, Corollary 4.9, and the properties of cographs.

LEMMA 4.10. *If G is a cograph, Algorithm* TREE$(V(G), G)$ *returns T, the cotree of G.*

Proof. The proof is a simple induction argument. Certainly, a single leaf is the cotree of the graph on one vertex. So now consider when M is the entire vertex set $V(G)$. Corollary 4.9 proves that v and the ① and ⓪ nodes in line ($\star$) or ($\star\star$) construct the path P_{vR} for the cotree of the cograph G. The remainder of the cotree consists of the subtrees whose parent nodes lie on P_{vR}. Since each slice $S_i(v)$ or $\overline{S}_i(v)$ is a module and furthermore, by the hereditary property of cographs, induces a cograph, by the inductive assumption, TREE$(S_i(v), G)$ will return the cotree representing $G[S_i(v)]$. In addition, by Lemma 4.7, the cotree returned by TREE$(S_i(v), G)$ is exactly the subtree T^v_{0i} of T. An analogous argument using Corollary 4.8 holds for $\overline{S}_j(v)$, completing the proof. □

For complexity reasons, it is impractical to compute LexBFS and LexBFS$^-$ orderings of each module in G. To meet the linear time bound, TREE(V(G), G) needs to be altered to use only the two orderings σ and $\overline{\sigma}^-$, a LexBFS of G and a LexBFS$^-$ of G using σ. Lemma 4.11 and Corollary 4.12 show that in certain situations, the two sweeps are sufficient.

LEMMA 4.11. *Given σ and $\overline{\sigma}^-$ of G and $S_i(v)$, for arbitrary v and i, if $S_i(v) = S(y)$ contains exactly the leaves of T^v_{0i}, then $S_j(y)$ contains exactly the leaves of T^y_{0j} for all j.*

Proof. Since $S(y) = S_i(v) = T^v_{0i}$, $S_i(v)$ induces a module and a cograph. Therefore, applying Lemma 4.7 to $S(y)$ implies that $S_j(y)$ contains exactly the leaves of T^y_{0j} for all j. ☐

Complementing Lemma 4.11 results in Corollary 4.12.

COROLLARY 4.12. *Given σ and $\overline{\sigma}^-$ of G and $\overline{S}_i(v)$ for arbitrary v and i, if $\overline{S}_i(v) = \overline{S}(y)$ contains exactly the leaves of T^v_{1i}, then $\overline{S}_j(y)$ contains exactly the leaves of T^y_{1j} for all j.*

This raises the following questions: If $S(y) = S_i(v) = T^v_{0i}$, does $\overline{S}_j(y)$ contain exactly the leaves of T^y_{1j} for each j? Or, if $\overline{S}(y) = \overline{S}_i(v) = T^v_{1i}$, does $S_j(y)$ contain exactly the leaves of T^y_{0j} for each j? We now answer these questions with the following observation and lemmas.

Since the leaves of an induced subtree of a cotree induce a module, the next observation follows immediately from Lemma 3.7.

OBSERVATION 5. *Given σ and $\overline{\sigma}^-$ of G, if z is the leftmost vertex with respect to σ in T^y_{0i} then $T^y_{0i} \subseteq S(z)$ and $T^y_{0i} \subseteq \overline{S}(z)$. The same is true for z, the leftmost vertex with respect to σ in T^y_{1j}.*

Lemma 4.13 claims that if $S_i(v)$ contains the leaves of a subtree T^v_{0i}, then in $\overline{\sigma}^-$, the only vertices in the inclusion minimal slice containing $S_i(v)$ either belong to $S_i(v)$ or are independent of $S_i(v)$.

LEMMA 4.13. *Consider σ and $\overline{\sigma}^-$ of G and any vertex v such that $S(v)$ is a module and any index i such that $S_i(v)$ contains exactly the leaves of T^v_{0i}. Let $\overline{S}(u)$ be the inclusion-minimal slice containing $S_i(v)$; then $\overline{S}(u) - S_i(v)$ is a subset of $\overline{S^A}(u)$ and for all $z \in \overline{S}(u) - S_i(v)$, z is independent of the vertices in $S_i(v)$.*

Proof. By Lemma 3.7 and the fact that $S(v)$ induces a module, $S(v) \subseteq \overline{S}(v)$. Since each vertex in $S_i(v)$ is not adjacent to v and $S_i(v) \subseteq \overline{S}(u) \subset \overline{S}(v)$, every vertex in $\overline{S}(u)$ must be uniform with respect to v. Therefore, for all $y \in \overline{S}(u)$, y and v cannot be adjacent, implying that $lca(y, v) = 0$. Now consider any vertex $z \in \overline{S}(u) - S_i(v)$. Since $lca(z, v) = 0$, then $z \in T$, where T is rooted at the child of a 0 node (not 0^v_i) on P_{vR}. Hence, the lca of z with any leaf in T^v_{0i} is a 0 node. Since $u \in S_i(v)$ and every vertex in $\overline{S}(u) - S_i(v)$ is not adjacent to u, it must be the case that $\overline{S}(u) - S_i(v) \in \overline{S^A}(u)$ and for all $z \in \overline{S}(u) - S_i(v)$, z is independent of the vertices in $S_i(v) = T^v_{0i}$. ☐

Lemma 4.13 indicates that we can recurse on $\overline{S}(u)$ to construct the 1-rooted subtrees of T^v_{0i}. Similarly, Corollary 4.14 shows that the complement case holds as well, i.e., that if $\overline{S}_j(v)$ contains exactly the leaves of T^v_{1j}, then we can recurse on the inclusion-minimum slice containing $\overline{S}_i(v)$ in σ to construct the 0 rooted subtrees of T^v_{1j}. The proof of the corollary is very similar to that of the lemma.

COROLLARY 4.14. *Consider σ and $\overline{\sigma}^-$ of G and any vertex v such that $\overline{S}(v)$ is a module and any index j such that $\overline{S}_j(v)$ contains exactly the leaves of T^v_{1j}. Let $S(u)$ be the inclusion-minimal slice containing $\overline{S}_j(v)$; then $S(u) - \overline{S}_j(v)$ is a subset of $S^A(u)$ and for all $y \in S(u) - \overline{S}_j(v)$, y is universal to the vertices in $\overline{S}_j(v)$.*

LEMMA 4.15. *Consider σ and $\overline{\sigma}^-$ of G. For any vertex v in V, $S_i(v)$ contains exactly the leaves of T^v_{0i} and $\overline{S}_j(v)$ contains exactly the leaves of T^v_{1j}.*

Proof. Throughout this proof we will apply Observation 4 to conclude that the various $S_j(v), \overline{S}_j(v)$ are modules. By Lemma 4.7 and Corollary 4.8, if v is the first vertex of σ, the claim holds. The proof will be by induction on the nested depth of the subslices. Assume that $S(y)$ is a module and that $S_i(y)$ contains exactly the leaves of T^y_{0i} and $\overline{S}_j(y)$ contains exactly the leaves of T^y_{1j}. We will consider each case separately.

Let v be the first vertex of $S_i(y)$, so $S(v) = S_i(y)$. By Lemma 4.11, each $S_k(v)$ contains exactly the vertices of the subtrees T^v_{0k} of T^y_{0i}. We now consider the 1-rooted subtrees of T^y_{0i}. Since v is the first vertex in $S_i(y)$, Observation 5 implies that $S_i(y) \subseteq \overline{S}(v)$. Let the set of vertices in $\overline{S}(v)$ but *not* in $S_i(y)$ be denoted by X. Lemma 4.13 applied to $S(y)$ and $S_i(y)$ implies that $X \in \overline{S^A}(v)$ and, furthermore, that every vertex in X is independent of the vertices in $\overline{S}(v) \cap S_i(y)$. Therefore, we can use an argument similar to that in the proof of Corollary 4.8 to show that each $\overline{S}_j(v)$ contains exactly the leaves of the 1-rooted subtrees T^v_{1j} of T^y_{0i}.

Similarly, if v is the first vertex of $\overline{S}_j(y)$, then by Corollary 4.12, each $\overline{S}_k(v)$ contains exactly the leaves of the subtrees T^v_{1k} of T^y_{1j}. Observation 5 implies that given $\overline{S}_j(y) = T^y_{1j}$ and that v is the first vertex of $\overline{S}_j(y)$, $\overline{S}_j(y) \subseteq S(v)$. Corollary 4.14 requires that every vertex in $S(v) - \overline{S}_j(y)$ belongs to $S^A(v)$ and is universal to the vertices in $\overline{S}_j(y)$. Therefore, we can use an argument analogous to that in the proof of Lemma 4.7 to show that each $S_m(v)$ contains exactly the leaves of the 0-rooted subtrees T^v_{0m} of T^y_{1j}. ☐

We now have the tools to prove that a recursive cotree construction algorithm can be achieved using only σ and $\overline{\sigma}^-$.

LEMMA 4.16. *Given σ and $\overline{\sigma}^-$ of a cograph G, the cotree T of G can be constructed.*

Proof. Certainly the claim holds for the trivial case of a single vertex. We will argue inductively by considering the largest slice possible and showing that the recursion holds.

Let x be the first vertex of σ and therefore of $\overline{\sigma}^-$ as well, and consider each $S_i(x)$. By Lemma 4.7, $S_i(x)$ contains exactly the leaves of T^x_{0i}. In addition, since $S(x) = V$, $S(x)$ induces a module. Let u be the first vertex of $S_i(x)$. By Lemmas 4.11 and 4.15, each $S_j(u)$ contains the leaves of the subtree T^u_{0j}, and each $\overline{S}_h(u)$ contains exactly the leaves of T^u_{1h}. Therefore, we can build the path P_u, from u to the root of T^x_{0i} as in the proof of Corollary 4.9 and by induction, build each subtree T^u_{0j} and T^u_{1h} rooted on P_u.

The argument for the complement is nearly identical. By Corollary 4.8, $\overline{S}_m(x)$ contains exactly the leaves of T^x_{1m}. If u is the first vertex of $\overline{S}_m(x)$, then by Corollary 4.12 and Lemma 4.15, each $\overline{S}_j(u)$ contains the leaves of the subtree T^u_{1j} and each $S_h(u)$ contains exactly the leaves of T^u_{0h}. Therefore, again, we can build the path from u to the root of T^x_{1m} as in the proof of Corollary 4.9 and by induction, build each subtree T^u_{1j} and T^u_{0h}. ☐

Notice that at each stage of building a path in the cotree and recursing on the subtrees, for each vertex v, we need only to know the first vertex of each of the $S_i(v)$ and $\overline{S}_j(v)$ subslices. We will denote the first vertex of the ith subslice by $v[i]$ and the first vertex of the jth subslice in the complement by $\overline{v}[j]$. Each list of first vertices is generated during the appropriate LexBFS sweep at no extra cost to LexBFS complexity. COTREE illustrates the algorithm.

```
**Algorithm** COTREE (vertex $v$)
**Input:** Vertex $v$ of a slice $S(v)$ of cograph $G$ with respect to $\sigma$.
**Output:** A Cotree $T$ of $v \bigcup_{\forall i} S_i(v) \bigcup_{\forall j} \overline{S}_j(v)$.
    if ($v[1] =$ nil and $\overline{v}[1] =$ nil),
        return ( $\{v\}$ )
    if ($v[1] =$ nil),
        return ( $v$ ① COTREE($\overline{v}[1]$) )
    if ($\overline{v}[1] =$ nil),
        return ( $v$ ⓪ COTREE($v[1]$) )
    if $v[1]\overline{v}[1] \in E$, then
($\star$)   return ( $v$⓪COTREE($v[1]$)①COTREE($\overline{v}[1]$)⓪COTREE($v[2]$)... )
    else
($\star\star$)  return ( $v$①COTREE($\overline{v}[1]$)⓪COTREE($v[1]$)①COTREE($\overline{v}[2]$)... )
```

LEMMA 4.17. *Given $v[i]$ and $\overline{v}[j]$ for each vertex v and for all i, j, Algorithm* COTREE*(v) belongs to $\mathcal{O}(n+m)$.*

Proof. Notice that each vertex v is passed to COTREE at most once and that for each v, the size of the lists $v[i]$ and $\overline{v}[i]$ is bounded by the degree of v. Further, one iteration of the algorithm requires $O(deg(v))$ time to execute line ($\star$) or ($\star\star$). The test that $v[1]\overline{v}[1] \in E$ can be completed in constant time. To see this, note that $v[1]$ must belong to $\overline{S^A(v)}$ and, further, by the LexBFS$^-$ tie breaking mechanism, must be the first vertex in $\overline{S^A}(v)$. This implies that $v[1]$ will be the first vertex of $N^l(\overline{S}_1(v))$ (with respect to the adjacencies of G) if and only if $v[1]\overline{v}[1] \in E$. The local neighborhood for each $\overline{S}_i(v)$ with respect to the adjacencies of G can be generated during the LexBFS$^-$ at no extra cost.

Therefore, summing for all v, the complexity is $\mathcal{O}(n+m)$. ∎

COROLLARY 4.18. *Let x be the first vertex of a pair of LexBFS σ, $\overline{\sigma}^-$ of a cograph G. Then Algorithm* COTREE(x) *produces a cotree of G in $\mathcal{O}(n+m)$ time.*

5. Putting things together.

Theorem 4.3 ensures the correctness of Algorithm RECOGNIZE_COGRAPH. The subroutines Algorithm COTREE and Algorithm REPORT_P$_4$ have already been described.

```
**Algorithm** RECOGNIZE_COGRAPH($G$)
**Input**: *Graph $G$.*
**Output**: *Cotree $T$ if $G$ is a cograph, an induced $P_4$ otherwise.*
    Let $\tau$ be an arbitrary ordering of $V$
    Compute LexBFS($G, \tau$) resulting in $\sigma$ of $G$, with first vertex $x$
    Given $\sigma$, compute LexBFS$^-$($G, \sigma$), a LexBFS of $\overline{G}$, resulting in $\overline{\sigma}^-$
    if $\sigma, \overline{\sigma}^-$ satisfy the NSP on $G$, $\overline{G}$, respectively,
        return ( COTREE($x$) )
    else
        return ( REPORT_$P_4$($\sigma, \overline{\sigma}^-$) )
end RECOGNIZE_COGRAPH
```

To show the linearity of the above algorithm, it remains to explain how the NSP can be checked in linear time. Although it can be done during the LexBFS search itself as explained in [3], it is simpler to do the test as a postprocessing. This test is divided into two steps.

```
Step 1
Sort the adjacency list of every vertex in $G$ with respect to $\sigma$
Step 2
for each $v \in V$ (following $\sigma$ increasing) where $v_2 \neq NIL$ do
    $A \leftarrow N_<(v_1) \cap S^A(v)$
    $i \leftarrow 1$
    while $v_{i+1} \neq NIL$ do
        $B \leftarrow N_<(v_{i+1}) \cap S(v)$
        if $B$ not included in $A$, then
            return ( NSP fails on $\sigma$, $v$, $i$ )
        else $A \leftarrow B$; $i \leftarrow i + 1$
    end while
end for
return ( NSP succeeds on $\sigma$ )
```

5.1. Complexity analysis. First we note, by Observation 2, that it suffices to consider $N_<(v_{i+1})$ instead of $N^l(S_{i+1}(v))$. Sorting the adjacency lists according to σ as specified in step 1 takes $\mathcal{O}(|E|)$ time; simply create a new set of adjacency lists, walk through the ordering, and for each vertex v, append v to the end of the adjacency list of each vertex $y \in N(v)$, i.e., for each y in the original adjacency list of v.

Each time the neighborhood of a vertex is used in the previous procedure, the edges to visit are at the beginning of its adjacency list and can be deleted from this list after having been processed. The computations of the sets A and B can be done easily using standard partition refinement techniques as developed in [20, 18]. Hence the overall complexity of this procedure is $O(m)$.

Therefore, testing the NSP on σ can be done in $O(n + m)$ time complexity. Observing that the NSP of a LexBFS ordering $\overline{\sigma}^-$ on $\overline{G}$ can be checked on G by reversing the order inclusion to be tested, it is straightforward to check the NSP of $\overline{\sigma}^-$ on $\overline{G}$ in linear time $\mathcal{O}(n + m)$. Similar arguments hold for finding a P_4 in G when $\overline{\sigma}^-$ on $\overline{G}$ does not fulfill the NSP. It can be done in $\mathcal{O}(\Delta)$ time, where Δ is the maximum degree of the graph G.

6. Concluding remarks.

We conclude by first stating the major result of this paper followed by a discussion of future research directions directly related to these results.

MAIN RESULT. *There exists a simple, linear time, LexBFS based recognition algorithm for the family of cographs that returns a certificate in the form of a cotree or an induced P_4.*

While optimal cograph recognition algorithms already exist, we feel our new LexBFS based recognition algorithm improves upon previous algorithms for several reasons. It follows a simple paradigm: *order* the vertices, *check* for a characterizing property, and *return* an appropriate certificate. In addition, the algorithm is easily implemented and returns a certificate in both the positive and negative cases. Finally we feel that the algorithm has potential to be generalized and extended to other families and, possibly, modular decomposition. We realize that our new algorithm does not have one very nice feature of the original linear time cograph recognition algorithm [10], in that it is not incremental. Unfortunately, LexBFS does not lend itself to incremental algorithms, and, as such, our algorithm cannot be used in dynamic situations.

Concerning the algorithm itself, it is natural to wonder if LexBFS is the only sweep that can be used in this way to recognize cographs. First we note that BFS is too weak since it does not guarantee the nesting of the local neighborhoods of the $\{S_i(v)\}$, and thus with such an algorithm, a cograph might not satisfy the NSP. If we try to generalize LexBFS to other maximal neighborhood searches (MNS) such as LexDFS (see [8]) and maximum cardinality search (MCS), we risk having some vertex of $S^N(v)$ before a vertex of $S^A(v)$. Note that using the 4-vertex condition of MNS [8] (namely, an ordering σ is an MNS of G if and only if, for any $x <_\sigma y <_\sigma z$ such that $xz \in E$ but $xy \notin E$, there exists v such that $v <_\sigma y$, $vy \in E$, and $vz \notin E$), it is easy to show that Lemma 3.1 holds, namely, every MNS of a cograph is umbrella-free.

Similar to P_4-reducible and P_4-sparse graphs, other families in the P_4-hierarchy such as P_4-lite, P_4-extendible, and P_4-tidy (see [2] for definitions) may be well suited to a recognition algorithm that is an extension of our cograph recognition algorithm. In addition, distance hereditary graphs have a simple recognition algorithm that is again based on this cograph algorithm, motivating the question of whether families of graphs related to distance hereditary graphs, such as parity graphs, $(k,+)$-distance hereditary graphs, or HHD-free graphs, have similar recognition algorithms.

Finally, the most interesting and most challenging question is whether there exists a modular decomposition algorithm that generalizes our cograph recognition algorithm. This is a natural question to ask since cographs are the family of completely decomposable graphs with respect to modular decomposition.

Note added in proof. The cograph recognition algorithm presented in this paper does extend to a simple, linear time algorithm for modular decomposition; however, an even easier linear time modular decomposition algorithm has recently appeared (see [27]).

Acknowledgments. We are grateful to the anonymous referees and Marty Golumbic for their comments and suggestions.

REFERENCES

[1] A. Brandstädt, F.F. Dragan, and F. Nicolai, *LexBFS orderings and powers of chordal graphs*, Discrete Math., 171 (1997), pp. 27–42.

[2] A. Brandstädt, V.B. Le, and J.P. Spinrad, *Graph Classes: A Survey*, SIAM Monogr. Discrete Math. Appl. 3, SIAM, Philadelphia, 1999.

[3] A. Bretscher, *LexBFS Based Recognition Algorithms for Cographs and Related Graphs Families*, Ph.D. thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 2004.

[4] A. Bretscher, D.G. Corneil, M. Habib, and C. Paul, *A simple linear time LexBFS cograph recognition algorithm*, in Graph Theoretical Concepts in Computer Science (WG), Lecture Notes in Computer Science 2880, 2003, pp. 119–130.

[5] J.-M. Chang, C.-W. Ho, and M.-T. Ko, *LexBFS ordering in asteroidal triple-free graphs*, in Proceedings of the 10th International Symposium on Algorithms and Computation (ISAAC), Lecture Notes in Comput. Sci. 1741, Springer-Verlag, Berlin, 1999, pp. 163–172.

[6] D.G. Corneil, *Lexicographic breadth first search—a survey*, in Proceedings of the 30th International Workshop on Graph Theory (WG2004), Lecture Notes in Comput. Sci. 3353, Springer-Verlag, Berlin, 2004, pp. 1–19.

[7] D.G. Corneil, *A simple* 3*-sweep LBFS algorithm for the recognition of unit interval graphs*, Discrete Appl. Math., 138 (2004), pp. 171–179.

[8] D.G. Corneil and R. Krueger, *A unified view of graph searching*, SIAM J. Discrete Math., 22 (2008), pp. 1259–1276.

[9] D.G. Corneil, H. Lerchs, and L.K. Stewart Burlingham, *Complement reducible graphs*, Discrete Appl. Math., 3 (1981), pp. 163–174.

[10] D.G. Corneil, Y. Perl, and L.K. Stewart, *A linear time recognition algorithm for cographs*, SIAM J. Comput., 14 (1985), pp. 926–934.

[11] D. G. Corneil, S. Olariu, and L. Stewart, *The ultimate interval graph recognition algorithm?*, in Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 1998, pp. 175–180.

[12] A. Cournier and M. Habib, *A new linear time algorithm for modular decomposition*, in Proceedings of the 19th International Colloquium on Trees in Algebra and Programming (CAAP), Lecture Notes in Comput. Sci. 787, Springer-Verlag, Berlin, 1994, pp. 68–84.

[13] E. Dahlhaus, *Efficient parallel recognition algorithms for cographs and distance hereditary graphs*, Discrete Appl. Math., 57 (1995), pp. 29–45.

[14] E. Dahlhaus, J. Gustedt, and R.M. McConnell, *Efficient and practical modular decomposition algorithm*, in Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 1997, pp. 26–35.

[15] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, 2nd ed., Ann. Discrete Math. 57, Elsevier Science B.V., Amsterdam, 2004.

[16] M.C. Golumbic, A. Mintz, and U. Rotics, *Factoring and recognition of read-once functions using cographs and normality, and the readability of functions associated with partial k-trees*, Discrete Appl. Math., 154 (2006), pp. 1465–1677.

[17] C.C. Gotlieb and S. Kumar, *Semantic clustering of index terms*, J. ACM, 15 (1968), pp. 493–513.

[18] M. Habib, R.M. McConnell, C. Paul, and L. Viennot, *LexBFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing*, Theoret. Comput. Sci., 234 (2000), pp. 59–84.

[19] M. Habib and C. Paul, *A simple linear time algorithm for cograph recognition*, Discrete Appl. Math., 145 (2005), pp. 183–197.

[20] M. Habib, C. Paul, and L. Viennot, *Partition refinement: An interested algorithmic tool-kit*, Internat. J. Found. Comput. Sci., 10 (1999), pp. 147–170.

[21] P. Hell and J. Huang, *Certifying LexBFS recognition algorithms for proper interval graphs and proper interval bigraphs*, SIAM J. Discrete Math., 18 (2005), pp. 554–570.

[22] B. Jamison and S. Olariu, *P-components and the homogeneous decomposition of graphs*, SIAM J. Discrete Math., 8 (1995), pp. 448–463.

[23] R.M. McConnell and J.P. Spinrad, *Linear-time modular decomposition and efficient transitive orientation of comparability graphs*, in Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 1994, pp. 536–545.

[24] R.M. McConnell and J.P. Spinrad, *Modular decomposition and transitive orientation*, Discrete Math., 201 (1999), pp. 536–545.

[25] D.J. Rose, R.E. Tarjan, and G.S. Lueker, *Algorithmic aspects on vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.

[26] S. Shew, *A Cograph Approach to Examination Scheduling*, Master's thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1986.

[27] M. Tedder, D. Corneil, M. Habib, and C. Paul, *Simpler linear-time modular decomposition via recursive factorizing permutations*, in Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 5125, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 634–645.

[28] D.B. West, *Introduction to Graph Theory*, 2nd ed., Prentice Hall, Upper Saddle River, NJ, 2001.