ELSEVIER

Discrete Applied Mathematics 89 (1998) 143–153

DISCRETE APPLIED MATHEMATICS

Partial and perfect path covers of cographs ☆

D.G. Kirkpatrick [a,*], K. Madhukar Reddy [b], C. Pandu Rangan [c], A. Srinivasan [d]

[a] *Department of Computer Science, University of British Columbia, 2366 Main Mall, Vancouver, BC, Canada, V6T 1Z1*

[b] *Department of Computer Science, University of Texas at Austin, Austin, Texas, USA*

[c] *Deptartment of Computer Science and Engineering, IIT-Madras, India*

[d] *Deptartment of Computer Science, University of Minnesota, Minneapolis, Minnesota, USA*

Received 13 October 1997; received in revised form 8 May 1998; accepted 18 May 1998

Abstract

A set $\mathscr{P}$ of disjoint paths in a graph G is called a (*complete*) *path cover* of G if every vertex of G belongs to one of the paths in $\mathscr{P}$. A path cover of any subgraph of G is called a *partial path cover* of G. For fixed $k>0$, a *k-blanket* of graph G is a partial path cover $\mathscr{P}$ of G, consisting of exactly k paths, that maximizes the size of the subgraph covered by $\mathscr{P}$. A *k-core* of graph G is a partial path cover $\mathscr{P}$ of G, consisting of exactly k paths, that minimizes the sum, over all vertices v of G, of the distance of v to its closest path in $\mathscr{P}$. The problems of finding a k-blanket or a k-core (for fixed k) of an arbitrary graph G as well as the dual minimum-path-cover problem (find a path cover of minimum size) are all NP-hard. A linear-time algorithm is known (C.J. Chang and D. Kuo, SIAM J. Discrete Math. 9 (1996) 309–316) for the minimum-path-cover problem on cographs (graphs that can be constructed from a collection of isolated vertices by union and complement operations). However, prior to this paper, polynomial-time algorithms for the k-core problem were known only for trees – and even then for $k=1,2$ only (Becker and Perl, Discrete Appl. Math. 11 (1985) 103–113; Morgan and Slater, SIAM J. Appl. Math. 37 (1979) 539–560). In this paper, we introduce a variant of a minimum path cover, called a *perfect path cover*. We show that every cograph has a perfect path cover, and we exploit this to obtain an $O(m+n\log n)$-time algorithm for finding, for any arbitrary k, a k-blanket or a k-core of a arbitrary cograph on n vertices and m edges. 1998 Published by Elsevier Science B.V. All rights reserved.

Keywords: Algorithm; Cograph; Core; Path cover

1. Introduction

All the graphs dealt with in this paper are simple, finite, undirected, and without multiple edges. A graph G with vertex set V and edge set E, is denoted by $G=(V,E)$. $|V|$ (respectively, $|E|$) is also denoted by n (respectively, m).

☆ This research was initiated while the second and fourth authors were students at IIT-Madras, India.

0166-218X/98/$19.00
PII: S0166-218X(98)00101-2

A *path* in G is a sequence of $t \geqslant 1$ distinct vertices $v_1, v_2, \ldots, v_t$ in V such that v_i and v_{i+1} are adjacent in G, for all $1 \leqslant i < t$. We say that such a path P *joins* vertex v_1 to vertex v_t, and define $span(P) = \{v_1, v_2, \ldots, v_t\}$. More generally, if $\mathscr{P}$ denotes a set of paths in G then $span(\mathscr{P}) = \bigcup_{P \in \mathscr{P}} span(P)$.

A set $\mathscr{P}$ of disjoint paths in G is called a (*complete*) *path cover* (respectively, a *partial path cover*) of G if $span(\mathscr{P}) = V$ (respectively, $span(\mathscr{P}) \subseteq V$). A partial path cover $\mathscr{P}$ has *size* $|\mathscr{P}|$. Obviously, every graph has a path cover (of size at most n). However, if the size is constrained a path cover might not exist; for example, having a path cover of size one is clearly equivalent to being Hamiltonian. Thus, several natural optimization problems arise in connection with path covers of a given graph. Specifically, the *minimum-path-cover problem* asks for the construction of a path cover of minimum size. Alternatively, one might fix the size of a partial path cover and maximize its span or minimize the worst-case or average-case distance of vertices from the partial cover.

Let $d(u, v)$ denote the number of edges in the shortest path joining the vertices u and v in G. If no such path exists then $d(u, v) = \infty$. Let P be a path in G. The *distance* of a vertex v from the path P, denoted by $d(v, P)$, is defined as

$$d(v, P) = \min\{d(v, j) \mid j \in span(P)\}.$$

More generally, if $\mathscr{P}$ denotes a set of paths in G then the *distance* of a vertex v from the set of paths $\mathscr{P}$, denoted by $d(v, \mathscr{P})$, is defined as

$$d(v, \mathscr{P}) = \min\{d(v, j) \mid j \in span(\mathscr{P})\}.$$

The *eccentricity* of a partial path cover $\mathscr{P}$ is defined by

$$ecc(\mathscr{P}) = \sum_{v \in V} d(v, \mathscr{P}).$$

For fixed $k > 0$, a *k-blanket* of graph G is a partial path cover $\mathscr{P}$ of G, consisting of exactly k paths, that maximizes $span(\mathscr{P})$. A *k-core* of graph G is a partial path cover $\mathscr{P}$ of G, consisting of exactly k paths, that minimizes $ecc(\mathscr{P})$.

The problems of finding a minimum size path cover, a k-blanket, or a k-core of an arbitrary graph G, for any fixed $k > 0$, are easily seen to be NP-hard, by a straightforward reduction from the *Hamiltonian path problem* [14]. (See [1] for some results on the minimum-path-cover problem, and [13, 16, 19, 21] for work related to the 1-blanket (longest path) problem.)

It is clear that these problems remain NP-hard when restricted to families of graphs for which the Hamiltonian path problem is NP-hard. However, the Hamiltonian path problem is polynomial-time solvable on several important graph families such as cocomparability graphs [12], suggesting the possibility of similar results for these generalizations of the Hamiltonian path problem. In fact, the minimum-path-cover problem is known (see [4]) to be linear-time solvable on cographs, a well-studied subfamily of cocomparability graphs (described in detail in the next section). See [15] for

material on cographs, cocomparability graphs, interval graphs, permutation graphs and other classes of perfect graphs.

The notion of a k-core is one of many that capture the essence of "centrality"of vertices or paths in graphs. The *median* of a graph G is a single vertex v which, viewed as a partial path cover, has minimum eccentricity. More generally, a *k-median* (or *multimedian*) of a graph is a set S of k vertices of minimum eccentricity. An application of the 1-core problem in routing a highway through a road network is mentioned in [22]. Other notions of central paths are considered in [18, 23, 24]. Algorithms for finding a median in a general graph and a multimedian in a tree are presented in [17, 20], respectively. A linear algorithm for finding a 1-core of a tree is presented in [22]. Becker and Perl gave an $O(n^2)$ algorithm for finding the 2-core of a tree [2].

In this paper, we introduce a strengthening of the notion of a minimum path cover, called a *perfect path cover*. We show that every cograph admits a perfect path cover and use this to obtain an $O(m+n\log n)$ time algorithm for finding, for any arbitrary k, a k-blanket or a k-core of an arbitrary cograph.

2. Cographs

The *complement* of a graph $G=(V,E)$, denoted $\overline{G}$, is the graph (V,E'), where $(u,v)\in E'$ if and only if $(u,v)\notin E$. A graph is *connected* if there exists a path between every pair of its vertices. The *connected components* of G are the maximal connected subgraphs of G.

Let $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ be two graphs. The *disjoint union* of G_1 and G_2, denoted by $G_1\cup G_2$, is defined as

$$G_1\cup G_2=(V_1\cup V_2,E_1\cup E_2).$$

The *product* of G_1 and G_2, denoted by $G_1\times G_2$, is defined as

$$G_1\times G_2=(V_1\cup V_2,E_1\cup E_2\cup\{(x,y)\,|\,x\in V_1,y\in V_2\}).$$

Note that $G=G_1\times G_2$ iff $G=\overline{\overline{G_1}\cup\overline{G_2}}$. It is easy to confirm that both disjoint union and product are associative operations on graphs.

The class of *cographs* (also called complement-reducible graphs [7], CU graphs [6]) is an important subclass of recursively defined perfect graphs. Cographs are obtained from a collection of isolated vertices by repeated application of union and complement operations. More formally:

Definition 2.1 (Corneil et al. [7]). The class $\mathscr{C}$ of *cographs* is defined as follows:

1. The graph on one vertex (K_1) belongs to $\mathscr{C}$.
2. If $G_1\in\mathscr{C}$ and $G_2\in\mathscr{C}$ then $G_1\cup G_2\in\mathscr{C}$.
3. If $G_1\in\mathscr{C}$ and $G_2\in\mathscr{C}$ then $G_1\times G_2\in\mathscr{C}$.
4. No graph other than the ones generated by a finite number of applications of 1, 2 and 3 above belong to $\mathscr{C}$.

Cographs happen to be precisely the class of graphs with no induced paths on four vertices [7]. Hence, algorithms tailored to cographs have application in practical settings such as examination scheduling and automatic clustering of index terms in which the associated graphs are expected to have no vertex induced paths of length greater than three [10].

Another algorithmically significant property of cographs is that they have a rooted tree representation (called a *cotree*) that is unique up to isomorphism. The leaves of the cotree of a cograph G are the vertices of G, and its internal nodes are of two types, which are called *union* and *product* nodes.

The cotree T_G associated with cograph G is defined recursively as follows:

- If G is an isolated vertex labelled x, then T_G is the node labelled x.
- If G is disconnected, let $G_1,\ldots,G_c$ denote the connected components of G. Then the root of T_G is a *union* node whose $c>1$ children are the roots of the cotrees of G_1 through G_c.
- If G is connected, let $\overline{G}_1,\ldots,\overline{G}_c$ denote the connected components of $\overline{G}$. Then the root of T_G is a *product* node whose $c>1$ children are the roots of the cotrees of G_1 through G_c.

It is easily seen that the number of nodes in a cotree is a linear function of n, the number of vertices of the corresponding cograph. The importance of cographs stems in part from the fact that their recognition, along with the construction of the corresponding cotree, can be done in $O(n+m)$ time [10]. Cotrees form the basis for fast polynomial algorithms for problems such as isomorphism, colouring, clique detection, Hamiltonicity, treewidth and pathwidth, and dominating sets [3, 7–9] on cographs. All but the first of these problems are NP-hard on general graphs. In fact, Courcelle and Mosbah have shown [11] that all problems that can be formulated in monadic second-order logic (without quantification of edge sets) can be solved in linear time on cographs.

3. k-Cores and k-blankets in product graphs

Suppose that the graph G is a product graph, i.e. it can be expressed as $G_1 \times G_2$, where $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ are graphs. (This is certainly the case if G is a connected cograph on two or more vertices, in which case G_1 and G_2 are also cographs.)

Lemma 3.1. *If $\mathscr{P}$ is a k-core or a k-blanket of G then span($\mathscr{P}$) must include at least* $\min\{|V_1|,|V_2|+k\}$ *vertices of V_1 and at least* $\min\{|V_2|,|V_1|+k\}$ *vertices of V_2.*

Proof. Suppose that *span($\mathscr{P}$)* includes s vertices of V_1 and t vertices of V_2. If $s<|V_1|$, let x be some vertex in $V_1 - span(\mathscr{P})$. Then no path of $\mathscr{P}$ terminates in V_2 or contains two consecutive vertices of V_2. Otherwise, such a path P could be extended to include x, using edges between V_1 and V_2, producing a new path set $\mathscr{P}'$ satisfying

$ecc(\mathscr{P}') < ecc(\mathscr{P})$ (since $d(x,\mathscr{P}') = 0 < d(x,\mathscr{P})$), and $span(\mathscr{P}') = span(\mathscr{P}) \cup x$, contrary to our assumption that $\mathscr{P}$ is either a k-core or a k-blanket of G. Hence, $s \geqslant t + k$. By a symmetric argument, if $t < |V_2|$ then $t \geqslant s + k$. It follows that $s \geqslant \min\{|V_1|, |V_2| + k\}$ and $t \geqslant \min\{|V_2|, |V_1| + k\}$. □

It follows from Lemma 3.1 that the notions of k-core and k-blanket coincide for product graphs:

Theorem 3.1. *Let $G = G_1 \times G_2$. Then a set $\mathscr{P}$ forms a k-core for G if and only if it is a k-blanket of G.*

Proof. Suppose $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$. If $\mathscr{P}$ is a k-core or a k-blanket of G then both V_1 and V_2 have non-empty intersections with $span(\mathscr{P})$. Hence, by the definition of product, $d(v,\mathscr{P}) \leqslant 1$, for all $v \in V_1 \cup V_2$. Since path sets $\mathscr{P}$ satisfying $d(v,\mathscr{P}) \leqslant 1$, for all $v \in V_1 \cup V_2$, have $ecc(\mathscr{P}) = n - |span(\mathscr{P})|$, they minimize $ecc(\mathscr{P})$ if and only if they maximize $span(\mathscr{P})$. It follows that $\mathscr{P}$ is a k-core if and only if it is a k-blanket. □

Corollary 3.1. *If $G = G_1 \times G_2$, then P is a 1-core of G iff it is a longest path in G.*

Implicit in the proof of Theorem 3.1 is the following useful relationship between the span and eccentricity of blankets and cores:

Corollary 3.2. *If $\mathscr{P}$ is a k-core or a k-blanket of a product graph G then $ecc(\mathscr{P}) = n - |span(\mathscr{P})|$.*

The notions of k-blanket and k-core are *not* equivalent for disconnected cographs. This is because, any k-core (say $\mathscr{P}$) of G must contain at least one path from each component of G; otherwise $ecc(\mathscr{P}) = \infty$. Such a condition is not required for a k-blanket of G. In Section 6 we will describe how to compute a k-core or k-blanket for an arbitrary cograph.

4. Perfect path covers

In this section we introduce the concept of a *perfect path cover*, which generalizes the concepts of minimum path cover and k-blanket. In Section 5 we show that a perfect path cover for a cograph can be computed in $O(n \log n)$ time. This leads in turn to efficient algorithms for finding both k-cores and k-blankets in cographs, for any arbitrary k (Section 6).

Definition 4.1. A sequence $\langle P_1, P_2, \ldots, P_k \rangle$ is a *perfect path cover* for G if and only if $\{P_1, \ldots, P_k\}$ is a path cover (and hence a k-blanket) of G, and for all j, $1 \leqslant j < k$, $\{P_1, \ldots P_j\}$ is a j-blanket of G.

Of course, not every graph admits a perfect path cover. However, we shall now prove that all cographs do admit perfect path covers. Once a perfect path cover is known, j-blankets can be easily computed for all j.

Theorem 4.1. *Every cograph admits a perfect path cover.*

Proof. Isolated vertices clearly admit a perfect path cover. Given the recursive definition of cographs it suffices to prove that if G_1 and G_2 have perfect path covers then so do $G_1 \times G_2$ and $G_1 \cup G_2$. This is taken up in the next two subsections. □

4.1. Perfect path covers under product operations

Throughout this subsection, we suppose graph $G=(V,E)$ has the form $G_1 \times G_2$ where $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$. Without loss of generality, we assume that $|V_1| \leqslant |V_2|$.

Let $\widehat{\mathscr{Q}}=\langle Q_1,Q_2,\ldots,Q_r\rangle$ be any perfect path cover of G_2 and let $t=\min\{|V_1|+1,r\}$. We denote by $V_1 \triangleleft \widehat{\mathscr{Q}}$ the sequence $\langle R_1,\ldots,R_{r-t+1}\rangle$ formed by interleaving the vertices of V_1 with vertices of V_2 to "glue together" t of the paths in $\widehat{\mathscr{Q}}$ in the following way. First, specify a total order on the vertices of V_2 as follows: if v is the ith from last vertex of Q_a and w is the jth from last vertex of Q_b, then v preceeds w if (i,a) preceeds (j,b) in the usual lexicographic ordering. Choose some arbitrary ordering of the vertices of V_1. Now insert the ith vertex of V_1 immediately after the ith vertex of V_2 in its associated path. Let $\langle Q'_1,Q'_2,\ldots,Q'_r\rangle$ denote the resulting sequence of paths. (Note that (i) each path Q'_a starts with a vertex of V_2, (ii) paths Q'_a with $1 \leqslant a \leqslant \min\{|V_1|, r\}$ end with a vertex of V_1, and (iii) $\{Q'_1,Q'_2,\ldots,Q'_r\}$ is a path cover of G.) Finally, concatenate the first t paths, $Q'_1,\ldots,Q'_t$ into one path R_1, by appending path Q'_{a+1} to path Q'_a, for $1 \leqslant a < t$, and rename Q'_a as R_{a-t+1}, for $t < a \leqslant r$.

Theorem 4.2. *If $|V_1| \leqslant |V_2|$ and $\widehat{\mathscr{Q}}$ is a perfect path cover of G_2 then $V_1 \triangleleft \widehat{\mathscr{Q}}$ is a perfect path cover of G.*

Proof. Suppose $\widehat{\mathscr{Q}}=\langle Q_1,Q_2,\ldots,Q_r\rangle$ and $V_1 \triangleleft \widehat{\mathscr{Q}}=\langle R_1,\ldots,R_s\rangle$. Note that $span(\{R_1,\ldots,R_j\}) = V_1 \cup span(\{Q_1,\ldots,Q_{r-s+j}\})$, for $1 \leqslant j \leqslant s$. In particular, $\{R_1,\ldots,R_s\}$ forms a path cover (and hence an s-blanket) of G. Thus, it remains to show that $\mathscr{R}^k=\{R_1, R_2,\ldots,R_k\}$ is a k-blanket of G, for all k, $1 \leqslant k < s$.

If $r \leqslant |V_1|+1$ then $s=1$ and there is nothing to show, so suppose $r > |V_1|+1$ and hence $s=r-|V_1|>1$. Let $\mathscr{S}=\{S_1,S_2,\ldots,S_k\}$ be any k-blanket of G. By Lemma 3.1, $V_1 \subseteq span(\mathscr{S})$. By removing the vertices of V_1 from $\mathscr{S}$, we get a set $\mathscr{S}'$ of $k' \leqslant k+|V_1|$ disjoint paths from G_2. Since $\langle Q_1,Q_2,\ldots,Q_r\rangle$ is a perfect path cover of G_2, the set $\mathscr{Q}^{k'}=\{Q_1,Q_2,\ldots,Q_{k'}\}$ is a k'-blanket of G_2 and hence $|span(\mathscr{Q}^{k'})| \geqslant |span(\mathscr{S}')|$. Thus $|span(\mathscr{R}^k)| = |V_1| + |span(\mathscr{Q}^{|V_1|+k})| \geqslant |V_1| + |span(\mathscr{Q}^{k'})| \geqslant |V_1| + |span(\mathscr{S}')| = |span(\mathscr{S})|$. Hence, $\mathscr{R}^k$ is a k-blanket of G, for all k, $1 \leqslant k < s$. □

Theorem 4.2 above demonstrates that a product graph has a perfect path cover whenever its larger factor does. This asymmetry is exploited in our algorithm for constructing perfect path covers in cographs.

4.2. *Perfect path covers under union operations*

In this subsection, we suppose graph $G=(V,E)$ has the form $G_1 \cup G_2$, where $G_1 = (V_1,E_1)$ and $G_2=(V_2,E_2)$.

Let $\widehat{\mathscr{P}} = \langle P_1,P_2,\ldots,P_a\rangle$ be a perfect path cover for G_1 and let $\widehat{\mathscr{Q}} = \langle Q_1,Q_2,\ldots,Q_b\rangle$ be a perfect path cover for G_2. Define $\widehat{\mathscr{P}} \sqcup \widehat{\mathscr{Q}} = \langle R_1,R_2,\ldots,R_{a+b}\rangle$, where R_i denotes the ith longest path in the set $\{P_1,P_2,\ldots,P_a\} \cup \{Q_1,Q_2,\ldots,Q_b\}$.

Theorem 4.3. *Let $\widehat{\mathscr{P}}$ be a perfect path cover for G_1 and let $\widehat{\mathscr{Q}}$ be a perfect path cover for G_2. Then $\widehat{\mathscr{P}} \sqcup \widehat{\mathscr{Q}}$ is a perfect path cover for G.*

Proof. Suppose $\widehat{\mathscr{P}} = \langle P_1,P_2,\ldots,P_a\rangle$, $\widehat{\mathscr{Q}} = \langle Q_1,Q_2,\ldots,Q_b\rangle$, and $\widehat{\mathscr{P}} \sqcup \widehat{\mathscr{Q}} = \langle R_1,R_2,\ldots, R_{a+b}\rangle$. Clearly $\widehat{\mathscr{P}} \sqcup \widehat{\mathscr{Q}}$ is a path cover of G.

Let $\mathscr{T}$ be any k-blanket of G and suppose that $\mathscr{T}$ contains exactly i (respectively $k-i$) paths from G_1 (respectively G_2). As $\widehat{\mathscr{P}}$ and $\widehat{\mathscr{Q}}$ are perfect path covers for G_1 and G_2 respectively, it follows that $\mathscr{T}' = \{P_1,\ldots,P_i,Q_1,\ldots,Q_{k-i}\}$ is also a k-blanket in G. But, by construction, $\mathscr{R}^k = \{R_1,R_2,\ldots,R_k\}$ satisfies $span(\mathscr{R}^k) \geqslant span(\mathscr{T}')$, and hence $\mathscr{R}^k$ is also a k-blanket in G. □

5. Efficient computation of perfect path covers in cographs

Let $G=(V,E)$ be an arbitrary cograph and suppose that its corresponding cotree T_G is given. (As noted earlier, T_G can be constructed from a standard representation of G in $O(n+m)$ time. Hence, if T_G is not given the $O(n \log n)$ time bounds of this and subsequent sections should be read as $O(m + n \log n)$.)

In this section we first present an $O(n \log n)$-time algorithm for computing just the set of lengths associated with the paths in a perfect path cover of G. Later we show how to obtain the actual paths in a perfect path cover. We maintain a collection of sets of path lengths. (Since all paths are disjoint, the total number of lengths appearing in all sets is at most n.) Each set is represented by a standard mergeable priority queue (supporting *make_set*, *insert*, *extract_max* and *merge* operations in $O(\log n)$ time each) [5].

The computation begins with a preprocessing step in which T_G is converted into a binary tree (exploiting the associativity of both the product and union operations) and each node x of the resulting tree is labelled by $size(x)$, the number of leaves in the subtree rooted at x. Recall that our construction of a perfect path cover of a product graph depends only on the perfect path cover of its largest factor. For this reason, we also label each non-root node x as *non-critical* if the parent of x is non-critical or the

parent of x is a product node and x is not the largest (size) child of its parent (with ties broken arbitrarily). It should be obvious that this preprocessing can be carried out in time linear in the size of T_G.

The computation continues in a standard bottom up fashion, processing leaves first, and internal nodes only when both of their children have been processed. Processing a non-critical node involves nothing. Processing a critical node x involves the construction of S_x the set of lengths associated with the the paths in some perfect path cover of the cograph G_x, whose cotree is just the subtree of T_G rooted at x. We denote $|S_x|$ by s_x. There are three cases:

- (x is a leaf node.) We set $S_x := make\_set(1)$ and $s_x := 1$.
- (x is a union node.) Let y and z be the children of x in the T_G. Then, following Theorem 4.3, we set $S_x := merge(S_y, S_z)$ and $s_x := s_y + s_z$.
- (x is a product node.) Let y and z be the children of x in the cotree and assume, without loss of generality, that $size(y) \leqslant size(z)$ (and hence z is critical). Following Theorem 4.2, we consider two subcases separately.
  - ($s_z \leqslant size(y) + 1$.)
    In this case, as seen in Theorem 4.2, G_x has an Hamiltonian path.
    Hence, we set $S_x := make\_set(size(y) + size(z))$ and $s_x := 1$.
  - ($s_z > size(y) + 1$.)
    Using the idea of Theorem 4.2, S_x is constructed from S_z as follows.
    $total := size(y)$.
    for $i = 1$ to $size(y) + 1$ **do**
    $next := extract\_max(S_z)$;
    $total := total + next$;
    endfor
    $S_x := insert(S_z, total)$;
    $s_x := s_z - size(y)$.

5.1. Complexity

As stated in Section 2, $|T_G|$ is $O(n)$. The number of *make_set*, *merge* and *insert* operations, added together, is at most the number of nodes of T_G. To bound the number of *extract_max* operations, note that in processing any given (critical) product node x the number of *extract_max* operations is at most one more than the size of the non-critical child of x. Hence, the total number of *extract_max* operations is bounded by the number of critical product nodes plus the total size of all maximal non-critical nodes (non-critical nodes with critical parents). Since the latter is just the number of non-critical leaves, it follows that the total number of *extract_max* operations is $O(n)$.

Since the cost of processing any node is, up to constants, dominated by the cost of manipulating the appropriate priority queues, it follows that the entire algorithm runs in $O(n \log n)$ time.

5.2. Finding the paths

It is straightforward to modify the algorithm of the previous subsection to construct the paths in a perfect path cover of G in $O(n \log n)$ time. A path is represented by a singly linked list of its vertices (in reverse order). For a path P, $head(P)$ and $tail(P)$ point to the first and last vertex of P, respectively. Clearly, this representation suffices to permit concatenation of two paths in constant time.

For each non-critical node x in the cotree T_G, we need only compute the set of leaves of the subtree rooted at x. For each critical node x in the cotree T_G, we now compute not only the set S_x but also a corresponding set of paths. The required path manipulations (other than initialization) are as described in the proof of Theorem 4.2. As with the *extract_max* operation count above, it is easy to bound the number of elementary path manipulations by the total size of all maximal non-critical nodes. As previously noted, this is $O(n)$.

6. k-Cores and k-blankets in cographs

By the definition of perfect path cover, our $O(n \log n)$ algorithm for finding a perfect path cover constructs a minimum path cover of an arbitrary cograph given by its cotree representation. In addition, it provides a k-blanket for all suitable values of k. By Theorem 3.1, it follows that k-cores too can be constructed efficiently, provided the given cograph is connected (and hence a product graph). It remains to show how to compute the k-core of a disconnected cograph, for arbitrary k.

Let G be a disconnected cograph with connected components $G_1, G_2, \ldots, G_r$, for some $r > 1$. Obviously, if $r > k$, G does not have a partial path cover, consisting of k paths, with finite eccentricity. Hence, we assume that $r \leqslant k$. For each i, $1 \leqslant i \leqslant r$, let $\widehat{\mathscr{P}}_i$ denote a perfect path cover for G_i. Let $\mathscr{R}$ denote the partial path cover formed by the longest path from each of $\widehat{\mathscr{P}}_1, \widehat{\mathscr{P}}_2, \ldots, \widehat{\mathscr{P}}_r$ together with the $k - r$ longest of the remaining paths.

Theorem 6.1. *Assuming $k \geqslant r$, the set $\mathscr{R}$ described above is a k-core of G.*

Proof. Let $\mathscr{S}$ be any k-core of G. For each i, $1 \leqslant i \leqslant r$, let a_i denote the number of paths from G_i in $\mathscr{S}$. Clearly $a_i \geqslant 1$ and $\sum_{1 \leqslant i \leqslant r} a_i = k$.

Since $\widehat{\mathscr{P}}_i$ is a perfect path cover of G_i, it follows that the partial path cover $\mathscr{S}'$, formed by choosing, for each i, $1 \leqslant i \leqslant r$, the a_i longest paths in $\widehat{\mathscr{P}}_i$ (which, by Theorem 3.1, forms both an a_i-blanket and an a_i-core in G_i) satisfies $ecc(\mathscr{S}') \leqslant ecc(\mathscr{S})$. Hence $\mathscr{S}'$ also forms a k-core of G. But, by the construction of $\mathscr{R}$, we are guaranteed that $span(\mathscr{R}) \geqslant span(\mathscr{S}')$. Since $\mathscr{R}$ and $\mathscr{S}'$ both induce a blanket in each component of G, it follows from Corollary 3.2 that $ecc(\mathscr{R}) = n - span(\mathscr{R}) \leqslant n - span(\mathscr{S}') = ecc(\mathscr{S}')$. Hence $\mathscr{R}$ also forms a k-core of G. □

It should be clear from the above how to compute, in $O(n \log n)$ time, a k-core of given cograph G, for arbitrary k, using a perfect path cover of each of the components of G.

7. Conclusion

We have presented an $O(n \log n)$-time algorithm for finding a perfect path cover of an arbitrary cograph given by its cotree representation. From this $O(n \log n)$-time algorithms for constructing minimum path covers, k-blankets (including as a special case the longest path) and k-cores in cographs are easily formulated. To our knowledge these are the first polynomial algorithms for the k-blanket and k-core problems (for arbitrary k) on any non-trivial class of graphs.

The notion of a perfect path cover might be worth studying in connection with other graph families. It is perhaps worth remarking that our $O(n \log n)$ time algorithm for finding a perfect path cover, the central result of this paper, can be easily extended to apply to vertex-weighted cographs (with the natural generalization of path length).

References

[1] S.R. Arikati, C. PanduRangan, Linear algorithm for optimal path cover problem on interval graphs, Inform. Process. Lett. 35 (1990) 149–153.
[2] R.I. Becker, Y. Perl, Finding the two-core of a tree, Discrete Appl. Math. 11 (1985) 103–113.
[3] H.L. Bodlaender, R.H. Mohring, Treewidth and pathwidth of cographs, SIAM J. Discrete Math. 6 (1993) 181–188.
[4] G.J. Chang, D. Kuo, The $l(2,1)$-labelling problem on graphs, SIAM J. Discrete Math. 9 (1996) 309–316.
[5] T. Cormen, C.E Leiserson, R.L Rivest, Introduction to Algorithms, MIT Press and McGraw-Hill, New York, 1990.
[6] D.G. Corneil, D.G. Kirkpatrick, Families of recursively defined perfect graphs, Congr. Numer. 39 (1983) 237–246.
[7] D.G. Corneil, H. Lerchs, L.S. Burlingham, Complement reducible graphs, Discrete Appl. Math. 3 (1981) 163–174.
[8] D.G. Corneil, Y. Perl, Clustering and domination in perfect graphs, Discrete Appl. Math. 9 (1984) 27–39.
[9] D.G. Corneil, Y. Perl, L.K. Stewart, Cographs: recognition, applications and algorithms, Congr. Numer. 43 (1984) 249–258.
[10] D.G. Corneil, Y. Perl, L.K. Stewart, A linear recognition algorithm for cographs, SIAM J. Comput. 14 (1985) 926–934.
[11] B. Courcelle, M. Mosbah, Monadic second-order evaluations on tree-decomposable graphs, Theoret. Comput. Sci. 109 (1993) 49–82.
[12] J.S. Deogun, G. Steiner, Polynomial algorithms for hamiltonian cycle in cocomparability graphs, SIAM J. Comput. 23 (1994) 520–552.
[13] J.A. Ellis, M. Mata, G. MacGillivray, A linear time algorithm for longest (s,t)-paths in weighted outerplanar graphs, Inform. Process. Lett. 32 (1989) 199–204.
[14] M.R. Garey, D.S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, Freeman, San Fransisco, 1979.
[15] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.
[16] G. Gutin, Finding a longest path in a complete multipartite digraph, SIAM J. Discrete Math. 6 (1993) 270–273.

[17] S.L. Hakimi, Optimal location of switching centers and the associated centers and medians of a graph, Oper. Res. 12 (1964) 450–459.
[18] M.S. Hedetneimi, E.J. Cockayne, S.T. Hedetneimi, Linear algorithms for finding the jordan center and path center of a tree, Transportation Sci. 15 (1981) 98–114.
[19] D. Karger, R. Motwani, Ramkumar, On approximating the longest path in a graph. Workshop on Algorithms and Data structures, Lecture Notes in Computer Science, 709, 1993, pp. 421–430.
[20] O. Kariv, S.L. Hakimi, An algorithmic approach to network location problems, ii: The p-medians, SIAM J. Appl. Math. 37 (1979) 539–560.
[21] N. Megiddo, A. Tamir, E. Zemel, R. Chandrasekaran, An $O(n \log^2 n)$ algorithm for the kth longest path in a tree with applications to location problems, SIAM J. Comput. 10 (1981) 328–337.
[22] C.A. Morgan, P.J. Slater, A linear algorithm for the core of a tree, J. Algorithms 1 (1980) 247–258.
[23] P.J. Slater, Centrality of paths and vertices in a graph: cores and pits, Proc. 4th Internat. Conf. Theory and Appl. of Graphs, 1980, pp. 529–542.
[24] P.J. Slater, Locating central paths in graphs, Transportation Sci. 16 (1982) 1–18.