A Polynomial Solution to the k-Fixed-endpoint Path Cover Problem on Proper Interval Graphs

Katerina Asdre and Stavros D. Nikolopoulos

Department of Computer Science, University of Ioannina
P.O.Box 1186, GR-45110 Ioannina, Greece
{katerina, stavros}@cs.uoi.gr

Abstract: We study a variant of the path cover problem, namely, the k-fixed-endpoint path cover problem, or kPC for short. Given a graph G and a subset $\mathcal{T}$ of k vertices of $V(G)$, a k-fixed-endpoint path cover of G with respect to $\mathcal{T}$ is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the k vertices of $\mathcal{T}$ are all endpoints of the paths in $\mathcal{P}$. The kPC problem is to find a k-fixed-endpoint path cover of G of minimum cardinality; note that, if $\mathcal{T}$ is empty (or, equivalently, $k = 0$), the stated problem coincides with the classical path cover problem. The kPC problem generalizes some path cover related problems, such as the 1HP and 2HP problems, which have been proved to be NP-complete. Note that, the complexity status of both 1HP and 2HP problems on interval graphs remains an open question [9]. In this paper, we show that the kPC problem can be solved in linear time on the class of proper interval graphs, that is, in $O(n + m)$ time on a proper interval graph on n vertices and m edges. The proposed algorithm is simple, requires linear space, and also enables us to solve the 1HP and 2HP problems, on proper interval graphs within the same time and space complexity.

Keywords: perfect graphs, proper interval graphs, path cover, fixed-endpoint path cover, linear-time algorithms.

1 Introduction

A well studied problem with numerous practical applications in graph theory is to find a minimum number of vertex-disjoint paths of a graph G that cover the vertices of G. This problem, also known as the path cover problem (PC), finds application in the fields of database design, networks, code optimization among many others (see [1, 2, 30]); it is well known that the path cover problem and many of its variants are NP-complete in general graphs [11]. A graph that admits a path cover of size one is referred to as Hamiltonian. Thus, the path cover problem is at least as hard as the Hamiltonian path problem (HP), that is, the problem of deciding whether a graph is Hamiltonian. The path cover problem is known to be NP-complete even when the input is restricted to several interesting special classes of graphs; for example, it is NP-complete on planar graphs [12], bipartite graphs [13], chordal graphs [13], chordal bipartite graphs [21] and strongly chordal graphs [21]. Bertossi and Bonuccelli [6] proved that the Hamiltonian Circuit problem is NP-complete on several interesting classes of intersection graphs.

Several variants of the HP problem are also of great interest, among which is the problem of deciding whether a graph admits a Hamiltonian path between two points (2HP). The 2HP problem is the same as the HP problem except that in 2HP two vertices of the input graph G are specified, say, u and v, and we are asked whether G contains a Hamiltonian path beginning with u and ending with v. Similarly, the 1HP problem is to determine whether a graph G admits a Hamiltonian path starting from a specific vertex u of G, and to find one if such a path does exist. Both 1HP and 2HP problems are also NP-complete in general graphs [11].

The path cover problem as well as several variants of it have been extensively studied due to their wide applicability in many fields. Some of these problems, of both theoretical and practical importance, are in the context of communication and/or transposition networks [31]. In such problems, we are given a graph (network) G and two disjoint subsets $\mathcal{T}_1$ and $\mathcal{T}_2$ of vertices of G, and the objective is to determine whether G admits λ vertex-disjoint paths with several conditions on their endpoints with respect to $\mathcal{T}_1$ and $\mathcal{T}_2$, e.g., paths with both their endpoints in $\mathcal{T}_1 \cup \mathcal{T}_2$, paths with one endpoint in $\mathcal{T}_1$ and the other in $\mathcal{T}_2$, etc. [3, 31]; note that, the endpoints of a path P are the first vertex and the last vertex visited by P.

A similar problem that has received increased attention in recent years is in the context of communication networks. The only efficient way to transmit high volume communication, such as in multimedia applications, is through disjoint paths that are dedicated to pairs of processors. To efficiently utilize the network, one needs a simple algorithm that, with minimum overhead, constructs a large number of edge-disjoint paths between pairs of two given sets $\mathcal{T}_1$ and $\mathcal{T}_2$ of requests.

Furthermore, in the study of interconnection networks, the reliability of the interconnection network subject to node failures corresponds to the connectivity of an interconnection graph. It is well-known that the connectivity of a graph G is characterized in terms of vertex-disjoint paths joining a pair of vertices in G. Thus, one-to-many vertex-disjoint paths joining a vertex s (source) and k distinct vertices $t_1, t_2, \ldots, t_k$ (sinks) are required. A related work was presented by Park in [25].

Another related problem is the disjoint paths (DP) problem, which is defined as follows: Given a graph G and pairs (s_1, t_1), $(s_2, t_2), \ldots,$ (s_k, t_k) of vertices of G, the objective is to determine whether G admits k vertex-disjoint paths P_1, $P_2, \ldots,$ P_k in G such that P_i joins s_i and t_i $(1 \leq i \leq k)$. The problem was shown to be NP-complete by Karp [17] if k is a variable part of the input. For fixed k, however, the problem is more tractable; a polynomial-time algorithm was described by Robertson and Seymour [27]. Note that, for $k = 2$, there are several polynomial-time algorithms in the literature for the DP problem [28, 29]. In contrast, the corresponding question for directed graphs G where we seek directed paths $P_1, P_2, \ldots, P_k$ is NP-complete even for $k = 2$ [10].

In [9], Damaschke provided a foundation for obtaining polynomial-time algorithms for several problems concerning paths in interval graphs, such as finding Hamiltonian paths and circuits, and partitions into paths. In the same paper, he stated that the complexity status of both 1HP and 2HP problems on interval graphs remains an open question.

Motivated by the above issues we state a variant of the path cover problem, namely, the k-fixed-endpoint path cover problem (kPC), which generalizes both 1HP and 2HP, and also problem A.

Problem kPC: Given a graph G and a set $\mathcal{T}$ of k vertices of $V(G)$, a *k-fixed-endpoint path cover* of the graph G with respect to $\mathcal{T}$ is a path cover of G such that all vertices in $\mathcal{T}$ are endpoints of paths in the path cover; a *minimum k-fixed-endpoint path cover* of G with respect to $\mathcal{T}$ is a k-fixed-endpoint path cover of G with minimum cardinality; the *k-fixed-endpoint path cover problem* (kPC) is to find a minimum k-fixed-endpoint path cover of the graph G.

In this paper, we study the complexity status of the k-fixed-endpoint path cover problem (kPC) on the class of proper interval graphs, and show that this problem can be solved in polynomial time when the input is a proper interval graph [4, 7, 8]. A *proper interval graph* is an interval graph that has an interval representation where no interval is properly contained in another. Proper interval graphs arise naturally in applications such as DNA sequencing [14]. The proposed algorithm runs in time linear in the size of the input graph G on n vertices and m edges, that is, in $O(n + m)$

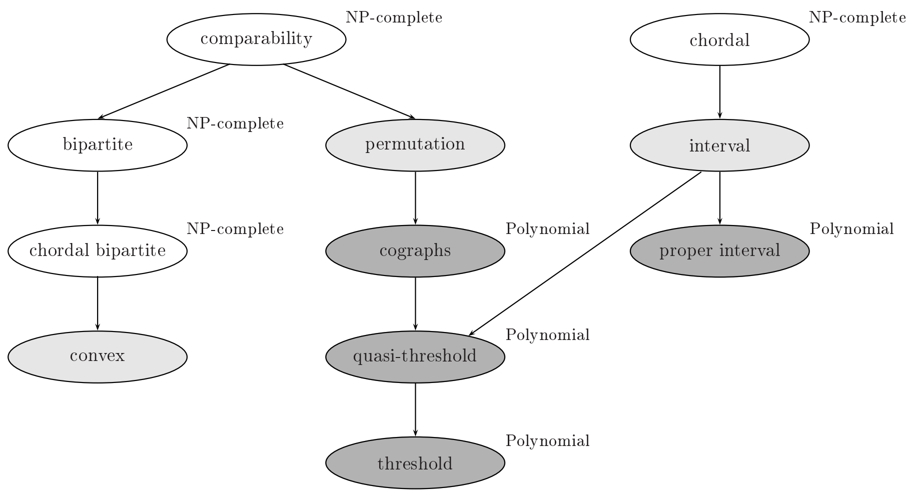

Figure 1: The complexity status (NP-complete, unknown, polynomial) of the kPC problem for some graph subclasses of comparability and chordal graphs. $A \to B$ indicates that class A contains class B.

time, and requires linear space. The proposed algorithm for the kPC problem can also be used to solve the 1HP and 2HP problems on proper interval graphs within the same time and space complexity. Figure 1 shows a diagram of class inclusions for a number of graph classes, subclasses of comparability and chordal graphs, and the current complexity status of the kPC problem and, thus, of 1HP and 2HP as well on these classes. Note that, if problems 1HP and 2HP are polynomially solvable on interval graphs, then they are also polynomially solvable on convex graphs [21]. The definitions of the classes appearing in the diagram, are given in Section 2 (see also [7, 13]).

The class of proper interval graphs has been extensively studied in the literature [13, 26] and several linear-time algorithms are known for their recognition and realization [8, 24]. Both Hamiltonian Circuit (HC) and Hamiltonian Path (HP) problems are polynomially solvable for the class of interval and proper interval graphs. Keil introduced a linear-time algorithm for the HC problem on interval graphs [18] and Arikati and Rangan [2] presented a linear-time algorithm for the minimum path cover problem on interval graphs. Bertossi [5] proved that a proper interval graph has a Hamiltonian path if and only if it is connected. He also gave an $O(n \log n)$ algorithm for finding a Hamiltonian circuit in a proper interval graph. Panda and Pas [24] presented a linear-time algorithm to detect if a proper interval graph is Hamiltonian. Recently, Asdre and Nikolopoulos [3] studied the k-fixed-endpoint path cover problem (kPC) on cographs. Based on the tree representation of a cograph (co-tree) and using operations similar to those described in this paper, they proposed an algorithm which solves the kPC problem in linear time. A unified approach to solving the Hamiltonian problems on distance-hereditary graphs was presented in [16], while Hsieh [15] presented an efficient parallel strategy for the 2HP problem on the same class of graphs. Nakano et al. [22] proposed an optimal parallel algorithm which finds and reports all the paths in a minimum path cover of a cograph in $O(\log n)$ time using $O(n/\log n)$ processors on a PRAM model. Moreover, recently Nikolopoulos [23] solved the Hamiltonian problem on quasi-threshold graphs (a subclass of cographs) in $O(\log n)$ time using $O(n+m)$ processors on a PRAM model.

The paper is organized as follows. In Section 2 we establish the notation and related terminology, and we present background results. In Section 3 we describe our linear-time algorithm for the kPC problem, while in Section 4 we prove its correctness and compute its time and space complexity. Finally, in Section 5 we conclude the paper and discuss possible future extensions.

2 Theoretical Framework

We consider finite undirected graphs with no loops or multiple edges. Let G be a graph; we denote its vertex set by $V(G)$ and its edge set by $E(G)$. Let $N(v) = \{w \in V(G) | vw \in E(G)\}$ be the set of neighbors of v and $N[v] = N(v) \cup \{v\}$. Let S be a subset of the vertex set of a graph G. Then, the subgraph of G induced by S is denoted by $G[S]$. If $G[N(v)]$ is a complete subgraph, then v is called a *simplicial vertex* of G.

We study the k-fixed-endpoint path cover problem (kPC) on proper interval graphs. A graph is a proper interval graph if and only if it is an interval graph with no induced subgraph isomorphic to the claw, which is the graph $K_{1,3}$ consisting of one vertex adjacent to three pairwise non-adjacent vertices [26]. A graph G is an *interval graph* if its vertices can be put in a one-to-one correspondence with a family F of intervals on the real line such that two vertices are adjacent in G if and only if their corresponding intervals intersect. The graph classes of proper interval graphs, interval graphs, and chordal graphs are *hereditary*: if G is in the class, then every induced subgraph of G is in the same class. Recall that a graph G is *chordal* if it contains no induced cycles larger than triangles and, a graph G is *chordal bipartite* if it is a bipartite graph in which every induced cycle is a C_4 (chordless cycle on 4 vertices). Furthermore, a bipartite graph $G(X, Y; E)$ is called X-*convex* (or simply convex) if there exists an ordering of its vertices such that for all $y \in Y$ the vertices of $N(y)$ are consecutive [21].

Figure 1 shows a diagram of class inclusions for the above classes of graphs, and the current complexity status of the kPC problem on these classes. Recall that a graph G is called *cograph*, if G contains no induced subgraph isomorphic to P_4 (chordless path on 4 vertices) [7, 13]. A graph G is called *quasi-threshold* if G contains no induced subgraph isomorphic to P_4 or C_4 [7, 13]. Finally, a graph G is a *threshold graph* if and only if G does not contain $2K_2$, P_4 or C_4 as induced subgraphs [13].

The above graph classes are intersection graphs; in fact, all graphs are intersection graphs [20]. Let $\mathcal{F} = \{S_1, S_2, \ldots, S_n\}$ be any family of sets. The intersection graph of $\mathcal{F}$, denoted $\Omega(\mathcal{F})$, is the graph having $\mathcal{F}$ as vertex set with S_i adjacent to S_j iff $i \neq j$ and $S_i \cap S_j \neq \emptyset$. A graph G is an intersection graph if there exists a family $\mathcal{F}$ such that $V(G) = \{v_1, v_2, \ldots, v_n\}$ with each v_i corresponding to S_i and $v_i v_j \in E(G)$ iff $S_i \cap S_j \neq \emptyset$.

2.1 Structural Properties of Proper Interval Graphs

Let G be an interval graph and let $\{I_v = [a_v, b_v]\}$ be an interval representation of G; here, a_v and b_v ($a_v \leq b_v$) are referred to as the left and right endpoint of the interval I_v. The graph G is called a *unit* interval graph if all the intervals in the representation have unit length [19]. The family $\{I_V\}_{v \in V(G)}$ is the interval representation of a *proper* interval graph if no interval is properly contained in another. Clearly, unit interval graphs are proper interval graphs. Roberts [26] has proved the following fundamental result that shows that unit interval graphs, proper interval graphs, and indifference graphs are synonyms.

Proposition 2.1. [26]: *For a graph G, the following statements are equivalent:*

(i) G is a unit interval graph;

(ii) G is a proper interval graph;

(iii) G is an interval graph with no induced claw;

(iv) G is an indifference graph.

Proper interval graphs are characterized by an ordering of their vertices [19]:

Theorem 2.1. (Looges and Olariu [19]): *A graph G is a proper interval graph if and only if there exists a linear order π on $V(G)$ such that for every choice of vertices u, v, w,*

$$u \prec_\pi v \prec_\pi w, \quad \text{and} \quad uw \in E(G) \quad \text{implies} \quad uv, vw \in E(G). \tag{1}$$

Let G be a proper interval graph on n vertices and m edges; an ordering π of the vertices of G satisfying (1) is referred to as *canonical* and it can be constructed in $O(n+m)$ time [19]. Theorem 2.1 implies the following result.

Corollary 2.1. *Let G be a proper interval graph and let π be a linear order on the vertex set of G satisfying (1). For every choice of subscripts i, j with $(1 \leq i < j \leq n)$ and $v_i v_j \in E(G)$, the vertices $v_i, v_{i+1}, \ldots, v_j$ are pairwise adjacent.*

2.2 Proper Interval Graphs and the kPC Problem

We next present the definition of the k-fixed-endpoint path cover problem, or kPC, for short.

Definition 2.1. Let G be a graph and let $\mathcal{T}$ be a set of k vertices of $V(G)$. A *k-fixed-endpoint path cover* of the graph G with respect to $\mathcal{T}$ is a path cover of G such that all vertices in $\mathcal{T}$ are endpoints of paths in the path cover; a *minimum k-fixed-endpoint path cover* of G with respect to $\mathcal{T}$ is a k-fixed-endpoint path cover of G with minimum cardinality; the *k-fixed-endpoint path cover problem* (kPC) is to find a minimum k-fixed-endpoint path cover of the graph G.

Let G be a graph with vertex set $V(G)$ and edge set $E(G)$, $\mathcal{T}$ be a set of k vertices of $V(G)$, and let $\mathcal{P}_\mathcal{T}(G)$ be a minimum k-fixed-endpoint path cover of G with respect to $\mathcal{T}$ of size $\lambda_\mathcal{T}$; note that the size of $\mathcal{P}_\mathcal{T}(G)$ is the number of paths it contains. The vertices of the set $\mathcal{T}$ are called *terminal* vertices, and the set $\mathcal{T}$ is called the *terminal set* of G, while those of $V(G) - \mathcal{T}$ are called *non-terminal* vertices. Thus, the set $\mathcal{P}_\mathcal{T}(G)$ contains three types of paths, which we call *terminal*, *semi-terminal*, and *non-terminal* paths:

(i) a *terminal path* P_t consists of at least two vertices and both its endpoints, say, u and v, are terminal vertices, that is, $u, v \in \mathcal{T}$;

(ii) a *semi-terminal path* P_s is a path having one endpoint in $\mathcal{T}$ and the other in $V(G) - \mathcal{T}$; if P_s consists of only one vertex (trivial path), say, u, then $u \in \mathcal{T}$;

(iii) a *non-terminal path* P_f is a path having both its endpoints in $V(G) - \mathcal{T}$; if P_f consists of only one vertex, say, u, then $u \in V(G) - \mathcal{T}$.

Note that all the internal vertices of the paths of $\mathcal{P}_\mathcal{T}(G)$ are non-terminal vertices. Moreover, a semi-terminal path may consist of only one vertex which is a terminal vertex, while a terminal path contains at least two vertices. The set of the non-terminal paths in a minimum kPC of the graph G is denoted by N, while S and T denote the sets of the semi-terminal and terminal paths, respectively. Thus, we have

$$\lambda_\mathcal{T} = |N| + |S| + |T|. \tag{2}$$

From the Definition 2.1 of the k-fixed-endpoint path cover problem (kPC), we can easily conclude that the number of paths in a minimum kPC cannot be less than the number of terminal vertices divided by two. Furthermore, since each semi-terminal path contains one terminal vertex and each terminal path contains two, the number of terminal vertices is equal to $|S| + 2|T|$. Thus, the following proposition holds:

Proposition 2.2. *Let G be a graph and let $\mathcal{T}$ be a terminal set of G. Then $|\mathcal{T}| = |S| + 2|T|$ and $\lambda_\mathcal{T} \geq \lceil \frac{|\mathcal{T}|}{2} \rceil$.*

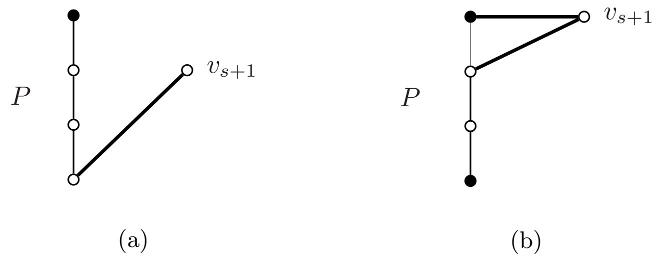

Figure 2: Illustrating (a) connect and (b) insert operations; $P \in \mathcal{P}_\mathcal{T}(G[\{v_1, v_2, \ldots, v_s\}])$.

Clearly, the size of a kPC of a graph G, as well as the size of a minimum kPC of G, is less than or equal to the number of vertices of G, that is, $\lambda_\mathcal{T} \leq |V(G)|$. Let $F(V(G))$ be the set of the non-terminal vertices of G. Thus, the following proposition holds:

Proposition 2.3. *Let G be a graph and let $\mathcal{T}$ be a terminal set of G. If $\lambda_\mathcal{T}$ is the size of a minimum kPC of G, then $\lambda_\mathcal{T} \leq |F(V(G))| + |\mathcal{T}|$.*

Section 3 describes our algorithm which takes as input a proper interval graph G and a subset $\mathcal{T}$ of its vertices and finds the paths of a minimum kPC of G. It is based on a greedy principle, visiting the intervals according to the canonical ordering. From now on, by v_i, $1 \leq i \leq n$, we mean the vertex of G that is numbered with integer i. We say that $v_i \prec_\pi v_j$ if $i < j$, $1 \leq i, j \leq n$. Let $G[S]$ be the induced subgraph of G consisting of vertices $S = \{v_1, v_2, \ldots, v_s\}$, $1 \leq s < n$ and let $\mathcal{P}_\mathcal{T}(G[S])$ be a minimum kPC of $G[S]$. Before describing our algorithm, we define two operations on paths of a minimum kPC $\mathcal{P}_\mathcal{T}(G[S])$ of a proper interval graph $G[S]$ and a vertex $v_{s+1} \in V(G)$, namely *connect* and *insert* operations; these operations are illustrated in Fig. 2.

- *Connect* operation: Let P be a non-terminal or a semi-terminal path of $\mathcal{P}_\mathcal{T}(G[S])$, where $G[S]$ is the induced subgraph of G consisting of vertices $S = \{v_1, v_2, \ldots, v_s\}$, $1 \leq s < n$, and let v_{s+1} be a non-terminal or a terminal vertex. We say that we *connect* vertex v_{s+1} to the path P, if we add an edge which joins vertex v_{s+1} with a non-terminal endpoint of P.

- *Insert* operation: Let $P = [t, p, \ldots, p', t']$ be a terminal or a semi-terminal path of $\mathcal{P}_\mathcal{T}(G[S])$, where $G[S]$ is the induced subgraph of G consisting of vertices $S = \{v_1, v_2, \ldots, v_s\}$, $1 \leq s < n$, and let v_{s+1} be a non-terminal vertex. We say that we *insert* vertex v_{s+1} into P, if we replace an edge of P, say, the edge tp, with the path $[t, v_{s+1}, p]$. Thus, the resulting path is $P = [t, v_{s+1}, p, \ldots, p', t']$.

Let P be a path of $\mathcal{P}_\mathcal{T}(G)$ and let v_i and v_j be its endpoints. We next define the left and the right endpoint of the path P as well as the available endpoint of a path. Moreover, we define an ordering on the paths in $\mathcal{P}_\mathcal{T}(G)$.

Definition 2.2. Let P be a path of $\mathcal{P}_\mathcal{T}(G)$ and let v_i and v_j be its endpoints. If $v_i \prec_\pi v_j$ then vertex v_i is the *left endpoint* and v_j is the *right endpoint* of the path P. Furthermore, vertex v_i is an *available endpoint* of P if (i) $v_i \in V(G) - \mathcal{T}$, that is, v_i is a non-terminal endpoint, or (ii) $v_i \in \mathcal{T}$ and $P = [v_i]$.

Definition 2.3. Let $\mathcal{P}_\mathcal{T}(G)$ be a minimum kPC of the graph G and let $P_i \in \mathcal{P}_\mathcal{T}(G)$ and $P_j \in \mathcal{P}_\mathcal{T}(G)$ such that v_a and v_b are the left and right endpoints of P_i, respectively, and v_c and v_d are the left and right endpoints of P_j, respectively. We define an ordering of the paths in $\mathcal{P}_\mathcal{T}(G)$ as follows: $P_i < P_j$, if $v_b \prec_\pi v_c$.

As shown in Section 3, if $v_c \prec_\pi v_b$ then the ordering of the endpoints of P_i and P_j is as follows: $v_c \prec_\pi v_d \prec_\pi v_a \prec_\pi v_b$.

3 The Algorithm

In this section we present an algorithm for the kPC problem on proper interval graphs. Our algorithm takes as input a proper interval graph G and a subset $\mathcal{T}$ of its vertices and finds the paths of a minimum kPC of G in linear time. The algorithm is based on a greedy principle to extend a path of a minimum kPC using operations on its paths and properties of the graph $G[\{v_1, v_2, \ldots, v_i\}]$, $1 \leq i \leq n$; if a vertex sees the two endpoints of only one non-terminal path P, it is connected to the left endpoint of the path P. Note that, according to Theorem 2.1, if $N_{G[\{v_1,v_2,\ldots,v_{i-1}\}]}(v_i) = \emptyset$, then the graph G is disconnected. Specifically, the algorithm works as follows:

Algorithm Minimum_kPC

Input: a proper interval graph G on n vertices and m edges and a set $\mathcal{T} \subseteq V(G)$;

Output: a minimum kPC $\mathcal{P}_{\mathcal{T}}(G)$ of the proper interval G;

1. Construct the canonical ordering π of the vertices of G;

2. Execute the subroutine *process*(π); the minimum kPC $\mathcal{P}_{\mathcal{T}}(G)$ is the set of paths returned by the subroutine;

where the description of the subroutine *process*(π) is presented at Fig. 3.

Let P be a path in the kPC $\mathcal{P}_{\mathcal{T}}(G)$ constructed by Algorithm Minimum_kPC and let v_i and v_j be its left and right endpoints, respectively (see Definition 2.2). Then, there is no path $P' \in \mathcal{P}_{\mathcal{T}}(G)$ having an endpoint between vertices v_i and v_j. Hence, we prove the following lemma:

Lemma 3.1. *Let P_s, $1 \leq s \leq \lambda_{\mathcal{T}}$ be a path of the kPC $\mathcal{P}_{\mathcal{T}}(G)$ of G constructed by Algorithm Minimum_kPC and let v_i and v_j be its left and right endpoints, respectively. Then, there is no path $P_t \in \mathcal{P}_{\mathcal{T}}(G)$, $1 \leq t \leq \lambda_{\mathcal{T}}$, $t \neq s$, such that $v_i \prec_\pi v_k \prec_\pi v_j$ or $v_i \prec_\pi v_\ell \prec_\pi v_j$, where v_k and v_ℓ are the left and right endpoints of P_t, respectively.*

Proof. Let P_s, $1 \leq s \leq \lambda_{\mathcal{T}}$ be a path of the kPC $\mathcal{P}_{\mathcal{T}}(G)$ constructed by Algorithm Minimum_kPC and let v_i and v_j be its left and right endpoints, respectively. Let $P_t \in \mathcal{P}_{\mathcal{T}}(G)$, $1 \leq t \leq \lambda_{\mathcal{T}}$, $t \neq s$, and let v_k and v_ℓ be its left and right endpoints, respectively. Suppose that $v_i \prec_\pi v_k \prec_\pi v_j$. Since v_i and v_j are the endpoints of P_s and $v_i \prec_\pi v_k \prec_\pi v_j$, the path P_s contains at least one edge, say, $v_a v_b$, such that $v_a \prec_\pi v_k \prec_\pi v_b$. Clearly, vertices v_a and v_b are non-terminal vertices. Since $v_a v_b \in E(G)$, we also have $v_a v_k \in E(G)$ and $v_k v_b \in E(G)$. Then, according to Algorithm Minimum_kPC, when vertex v_b is processed, it is either connected to the subpath of P_s which is already constructed having v_a as an endpoint or it is included to P_s through an edge $v_{a'} v_a$. If v_a is an endpoint when v_b is processed, it was also an endpoint when v_k was processed, and thus the edge $v_a v_k$ would appear to the path P_s, a contradiction. On the other hand, if v_b is included to P_s through the edge $v_{a'} v_a$, we have $v_{a'} \prec_\pi v_b$, and, thus, $v_{a'} v_k \in E(G)$; then v_k would appear to the path P_s, a contradiction. Similarly, we can prove that $v_\ell \prec_\pi v_i$ or $v_j \prec_\pi v_\ell$. ∎

Similarly, we can prove that, if P is a path of the kPC $\mathcal{P}_{\mathcal{T}}(G)$ constructed by Algorithm Minimum_kPC with v_i and v_j being its leftmost and rightmost vertices, respectively, there is no vertex between vertices v_i and v_j belonging to another path $P' \in \mathcal{P}_{\mathcal{T}}(G)$. Hence, we have the following lemma:

Lemma 3.2. *Let P_s, $1 \leq s \leq \lambda_{\mathcal{T}}$ be a path of the kPC $\mathcal{P}_{\mathcal{T}}(G)$ of G constructed by Algorithm Minimum_kPC and let v_i and v_j be its leftmost and rightmost vertices, respectively. Then, there is no vertex v_k belonging to another path $P_t \in \mathcal{P}_{\mathcal{T}}(G)$, $1 \leq t \leq \lambda_{\mathcal{T}}$, $t \neq s$, such that $v_i \prec_\pi v_k \prec_\pi v_j$.*

If $S = \{v_1, v_2, \ldots, v_{i-1}\}$, then $\mathcal{P}_{\mathcal{T}}(G[S])$ is the kPC of $G[S]$ constructed by the algorithm after processing $i-1$ vertices; let s be the size of $\mathcal{P}_{\mathcal{T}}(G[S])$, where $1 \leq s \leq \lambda_{\mathcal{T}}$. Let P_s be the path

```
process ($\pi$)
Input: the canonical ordering of the vertices of $G$.
Output: a minimum kPC $\mathcal{P}_\mathcal{T}(G)$ of $G$.
$P_1 \leftarrow [v_1]$; $i \leftarrow 2$; $\lambda_\mathcal{T} \leftarrow 1$;
$p_\ell \leftarrow v_1$;      {the left endpoint of the path $P_{\lambda_\mathcal{T}}$}
$p_r \leftarrow v_1$;      {the right endpoint of the path $P_{\lambda_\mathcal{T}}$}
while $i \leq n$ do
    if $v_i p_\ell \in E(G)$ then
        case 1:  if $p_\ell = p_r$ or $p_\ell \notin \mathcal{T}$ then
                     connect $v_i$ to $P_{\lambda_\mathcal{T}}$ through vertex $p_\ell$;
                     $p_\ell \leftarrow p_r$; $p_r \leftarrow v_i$;
        case 2:  if $p_\ell \in \mathcal{T}$ and $p_r \notin \mathcal{T}$ then
                     connect $v_i$ to $P_{\lambda_\mathcal{T}}$ through vertex $p_r$;
                     $p_r \leftarrow v_i$;
    else-if $v_i p_r \in E(G)$ and $p_r \notin \mathcal{T}$ then
        connect $v_i$ to $P_{\lambda_\mathcal{T}}$ through $p_r$;
        $p_r \leftarrow v_i$;
    else-if $v_s, v_t$ ($s < i$, $t < i$) are the rightmost successive
            neighbors of $v_i$ in $P_{\lambda_\mathcal{T}}$ then
        insert $v_i$ into $P_{\lambda_\mathcal{T}}$ through the edge $v_s v_t$;
    else
        $\lambda_\mathcal{T} \leftarrow \lambda_\mathcal{T} + 1$;
        $P_{\lambda_\mathcal{T}} \leftarrow v_i$;
        $p_\ell \leftarrow v_i$; $p_r \leftarrow v_i$;
    $i \leftarrow i + 1$;
endwhile;
$\mathcal{P}_\mathcal{T}(G) = \{P_1, \ldots, P_{\lambda_\mathcal{T}}\}$.
```

Figure 3: The subroutine *process*(π) of the Algorithm Minimum_kPC.

such that $P_i < P_s$, $\forall i \in [1, s)$ (see Definition 2.3), and, v_k and v_ℓ are its endpoints, where $k < \ell$. Let P_{s-1} be the path such that $P_i < P_{s-1}$, $\forall i \in [1, s-1)$, and, v_r and v_t are its endpoints, where $r < t$. Then, according to Lemma 3.1, $v_r \prec_\pi v_t \prec_\pi v_k \prec_\pi v_\ell$ and vertex v_i sees at least one of v_k and v_ℓ. Suppose that v_i can also be connected to the path P_{s-1}. If $v_i v_t \in E(G)$ and $v_t \notin \mathcal{T}$, then $v_k v_t \in E(G)$ and v_k is connected to the path P_{s-1}, a contradiction. If $v_t \in \mathcal{T}$ and $v_i v_r \in E(G)$ and $v_r \notin \mathcal{T}$, then $v_k v_r \in E(G)$ and v_k is connected to the path P_{s-1}, a contradiction. Consequently, if vertex v_i can be connected to the path P_s then it is the only path it can be connected to, and the following holds:

Corollary 3.1. *Let $\mathcal{P}_\mathcal{T}(G[S])$ be the kPC of $G[S]$ constructed by Algorithm Minimum_kPC, where $S = \{v_1, v_2, \ldots, v_{i-1}\}$ and let s be the size of $\mathcal{P}_\mathcal{T}(G[S])$, where $1 \leq s \leq \lambda_\mathcal{T}$. If vertex v_i can be connected to the path P_s, where $P_s > P_j$, $\forall j \in [1, s)$, then P_s is the only path that v_i can be connected to.*

Next, we prove the following lemma.

Lemma 3.3. *Let $\mathcal{P}_\mathcal{T}(G[S])$ be the kPC of $G[S]$ constructed by Algorithm Minimum_kPC, where $S = \{v_1, v_2, \ldots, v_{i-1}\}$. Unless vertices $v_{i-2}, v_{i-1} \in \mathcal{T}$, at least one of the following holds:*

(i) v_{i-2}, v_{i-1} are successive in a path in $\mathcal{P}_\mathcal{T}(G[S])$;

(ii) at least one of v_{i-2}, v_{i-1} is an available endpoint.

Proof. We use induction on i, $i \geq 3$. The basis $i = 3$ is trivial. Suppose that the hypothesis holds for $i - 1$: let $\mathcal{P}_{\mathcal{T}}(G[S'])$ be the kPC of $G[S']$ constructed by Algorithm Minimum_kPC, where $S' = \{v_1, v_2, \ldots, v_{i-2}\}$ and let s' be the size of $\mathcal{P}_{\mathcal{T}}(G[S'])$, where $1 \leq s' \leq \lambda_{\mathcal{T}}$; then, unless vertices $v_{i-3}, v_{i-2} \in \mathcal{T}$, at least one of the following holds: (i) v_{i-3}, v_{i-2} are successive in a path in $\mathcal{P}_{\mathcal{T}}(G[S])$; (ii) at least one of v_{i-3}, v_{i-2} is an available endpoint.

We show that, if $\mathcal{P}_{\mathcal{T}}(G[S])$ is the kPC of $G[S]$ constructed by Algorithm Minimum_kPC, where $S = \{v_1, v_2, \ldots, v_{i-1}\}$, and s is the size of $\mathcal{P}_{\mathcal{T}}(G[S])$, where $1 \leq s \leq \lambda_{\mathcal{T}}$, then, unless vertices $v_{i-2}, v_{i-1} \in \mathcal{T}$, at least one of the following holds: (i) v_{i-2}, v_{i-1} are successive in a path in $\mathcal{P}_{\mathcal{T}}(G[S])$; (ii) at least one of v_{i-2}, v_{i-1} is an available endpoint. Clearly, $v_{i-2}v_{i-1} \in E(G)$, otherwise $G[S]$ is disconnected. We distinguish the following cases:

Case 1: When vertex v_{i-2} was processed, it was connected to a path or it constructed a new one; let P be the path containing v_{i-2}. Then, vertex v_{i-1} is either connected to the path P or it is inserted to P, or a new path is constructed containing only vertex v_{i-1}. If v_{i-1} is connected to the path P or a new path is constructed containing only v_{i-1}, then at least one of the following holds: (i) v_{i-2}, v_{i-1} are successive in P; (ii) v_{i-1} is an available endpoint.

Consider the case where v_{i-1} is inserted into P, that is, v_{i-2} is not an available endpoint; then, unless $v_{i-3}, v_{i-2} \in \mathcal{T}$, by the induction hypothesis, at least one of the following holds: (i) v_{i-3}, v_{i-2} are successive in P; (ii) v_{i-3} is an available endpoint.

If $v_{i-3}, v_{i-2} \in \mathcal{T}$, they are either successive in P, or, according to Lemma 3.2, v_{i-2}, v_{i-4} are successive in P. Thus, if v_{i-1} is inserted into P, then v_{i-2} and v_{i-1} are successive. Consequently, at least one of the following holds: (i) v_{i-2}, v_{i-1} are successive in P; (ii) at least one of v_{i-2}, v_{i-1} is an available endpoint.

Case 2: When vertex v_{i-2} was processed, it was inserted into a path, say, P. Consequently, $v_{i-2} \in V(G) - \mathcal{T}$ and it has at least two neighbors. Vertex v_{i-1} does not see any endpoint since, if it did, v_{i-2} would also see it. Thus, v_{i-1} either constructs a new path or it is inserted into the path P. By the induction hypothesis, v_{i-3} and v_{i-2} are successive into P or v_{i-3} is an available endpoint. Thus, at least one of the following holds: (i) v_{i-2}, v_{i-1} are successive in P; (ii) at least one of v_{i-2}, v_{i-1} is an available endpoint.

Thus, the lemma follows. ∎

Generalizing the above lemma, we obtain the following:

Lemma 3.4. *Let $\mathcal{P}_{\mathcal{T}}(G[S])$ be the kPC of $G[S]$ constructed by Algorithm Minimum_kPC, where $S = \{v_1, v_2, \ldots, v_{i-1}\}$, and let $|N_{G[S]}(v_i)| > 2$. At least one of the following holds:*

(i) there exists a neighbor of v_i being an available endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G[S])$;

(ii) there exist two vertices $v_a, v_b \in N_{G[S]}(v_i)$ that are successive in a path in $\mathcal{P}_{\mathcal{T}}(G[S])$.

Proof. We use induction on i, $i \geq 4$. The basis $i = 4$ is trivial. Suppose that the hypothesis holds for $i-1$: let $\mathcal{P}_{\mathcal{T}}(G[S'])$ be the kPC of $G[S']$ constructed by Algorithm Minimum_kPC, where $S' = \{v_1, v_2, \ldots, v_{i-2}\}$, let s' be the size of $\mathcal{P}_{\mathcal{T}}(G[S'])$, where $1 \leq s' \leq \lambda_{\mathcal{T}}$, and $|N_{G[S']}(v_{i-1})| > 2$; then, at least one of the following holds: (i) there exists a neighbor of v_{i-1} being an available endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G[S'])$; (ii) there exist two vertices $v_{a'}, v_{b'} \in N_{G[S']}(v_{i-1})$ that are successive in a path in $\mathcal{P}_{\mathcal{T}}(G[S'])$.

We show that, if $\mathcal{P}_{\mathcal{T}}(G[S])$ is the kPC of $G[S]$ constructed by Algorithm Minimum_kPC, where $S = \{v_1, v_2, \ldots, v_{i-1}\}$, s is the size of $\mathcal{P}_{\mathcal{T}}(G[S])$, where $1 \leq s \leq \lambda_{\mathcal{T}}$, and $|N_{G[S]}(v_i)| > 2$, then, at least one of the following holds: (i) there exists a neighbor of v_i being an available endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G[S])$; (ii) there exist two vertices $v_a, v_b \in N_{G[S]}(v_i)$ that are successive in a path in $\mathcal{P}_{\mathcal{T}}(G[S])$. If v_{i-1} and v_{i-2} are not both terminal vertices, according to Lemma 3.3, the lemma holds. In the case where $v_{i-1}, v_{i-2} \in \mathcal{T}$, we distinguish the following cases:

Case 1: Vertices v_{i-1} and v_{i-2} belong to different paths. According to Lemma 3.1, vertex v_{i-1} belongs to a path consisting of only one vertex; thus, the lemma holds.

Case 2: Vertices v_{i-1} and v_{i-2} belong to a path consisting of only two vertices. Then, the lemma holds.

Case 3: Vertices v_{i-1} and v_{i-2} belong to a path P consisting of more than two vertices. Then, vertex v_{i-3} belongs to P (see Lemma 3.2), that is, $v_{i-3} \in V(G) - \mathcal{T}$, and, according to Lemma 3.3, when v_{i-1} is processed, v_{i-3} is an available endpoint or v_{i-3}, v_{i-2} are successive in P.

Consequently, the lemma follows. ∎

The above lemma also shows that, when vertex v_i is inserted into a path P, it is inserted through the edge $v_{i-2}v_{i-1}$ or $v_{i-3}v_{i-1}$.

4 Correctness and Time Complexity

Let G be a proper interval graph on n vertices and m edges and let $\mathcal{T}$ be a subset of $V(G)$. In order to prove the correctness of Algorithm Minimum_kPC, we use induction on n. We also prove a property of the minimum kPC $\mathcal{P}_{\mathcal{T}}(G)$ of G constructed by our algorithm: if $v_i \in N[v_n]$, $1 < i \leq n$ is the rightmost available endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G)$, then there is no other minimum kPC $\mathcal{P}'_{\mathcal{T}}(G)$ having a rightmost available endpoint $v_j \in N[v_n]$, $1 < j \leq n$, such that $v_i \prec_\pi v_j$. Hence, we prove the following theorem.

Theorem 4.1. *Let G be a proper interval graph on n vertices and m edges and let $\mathcal{T}$ be a subset of $V(G)$. Algorithm Minimum_kPC computes a minimum k-fixed-endpoint path cover $\mathcal{P}_{\mathcal{T}}(G)$ of G such that if $v_i \in N[v_n]$, $1 < i \leq n$, is the rightmost available endpoint of the paths in $\mathcal{P}_{\mathcal{T}}(G)$, then there is no minimum k-fixed-endpoint path cover $\mathcal{P}'_{\mathcal{T}}(G)$ having a rightmost available endpoint $v_j \in N[v_n]$, $1 < j \leq n$ such that $v_i \prec_\pi v_j$.*

Proof. We use induction on n. The basis $n = 1$ is trivial. Assume that Algorithm Minimum_kPC computes a minimum kPC $\mathcal{P}_{\mathcal{T}}(G[S])$ of every proper interval graph $G[S]$, $S = \{v_1, v_2, \ldots, v_{n-1}\}$, with at most $n-1$ vertices having $v_i \in N[v_{n-1}]$, $1 < i \leq n-1$, as the rightmost available endpoint of its paths, such that there is no other minimum kPC $\mathcal{P}'_{\mathcal{T}}(G[S])$ having a rightmost available endpoint $v_j \in N[v_{n-1}]$, $1 < j \leq n-1$, such that $v_i \prec_\pi v_j$; let $\lambda_{\mathcal{T}}(G[S])$ be the size of $\mathcal{P}_{\mathcal{T}}(G[S])$. We show that the algorithm computes a minimum kPC $\mathcal{P}_{\mathcal{T}}(G)$ of every proper interval graph G with n vertices having $v_k \in N[v_n]$, $i \leq k \leq n$, as the rightmost available endpoint of its paths, such that there is no other minimum kPC $\mathcal{P}'_{\mathcal{T}}(G)$ having a rightmost available endpoint $v_\ell \in N[v_n]$, $i \leq \ell \leq n$, such that $v_k \prec_\pi v_\ell$; let $\lambda_{\mathcal{T}}(G)$ be the size of $\mathcal{P}_{\mathcal{T}}(G)$.

Clearly, the size $\lambda'_{\mathcal{T}}(G)$ of a minimum kPC of G is $\lambda_{\mathcal{T}}(G[S])$ or $\lambda_{\mathcal{T}}(G[S]) + 1$. We distinguish the following cases:

Case 1: when the algorithm processes vertex v_n, it connects it to an existing path. Thus, $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S])$ and, consequently, $\mathcal{P}_{\mathcal{T}}(G)$ is a minimum kPC of G, that is, $\lambda'_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G)$. Furthermore, if $v_n \notin \mathcal{T}$ then $\mathcal{P}_{\mathcal{T}}(G)$ contains a path having vertex v_n as an available endpoint, which is clearly the rightmost available endpoint in $N[v_n]$ that any other minimum kPC $\mathcal{P}'_{\mathcal{T}}(G)$ can contain. Consider the case where $v_n \in \mathcal{T}$. Suppose that the rightmost available endpoint in $N[v_n]$ of $\mathcal{P}_{\mathcal{T}}(G)$ is the same as the rightmost available endpoint in $N[v_n]$ of $\mathcal{P}_{\mathcal{T}}(G[S])$, that is, vertex v_i. Let $\mathcal{P}'_{\mathcal{T}}(G)$ be a kPC of G having a rightmost available endpoint $v_\ell \in N[v_n]$, $i \leq \ell \leq n$, such that $v_i \prec_\pi v_\ell$. Clearly, $v_\ell \neq v_n$. Removing v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a minimum kPC of $G[S]$ having $v_\ell \in N[v_n]$ as an available endpoint; a contradiction since v_i is the rightmost endpoint of any minimum kPC of $G[S]$. Now suppose that vertex v_n is connected to vertex v_i. Then, according to the algorithm, there exists no available endpoint in $N[v_n]$ in $\mathcal{P}_{\mathcal{T}}(G)$. We show that there is no other minimum kPC $\mathcal{P}'_{\mathcal{T}}(G)$ containing an available endpoint in $N[v_n]$. Indeed, let $\mathcal{P}'_{\mathcal{T}}(G)$ be a minimum kPC containing a path P' with an available endpoint in $N[v_n]$, say v_e. Note that, $v_e \prec_\pi v_i$. Clearly, v_n belongs to path $P \in \mathcal{P}'_\mathcal{T}(G)$ containing more than one vertex and v_i is not an available endpoint of a path $P'' \in \mathcal{P}'_\mathcal{T}(G)$; let $P = [v_n, v_k, \ldots]$ and $v_{i'}$ be a vertex that v_i is connected to in P''. If $P \neq P'$ then removing v_n from $\mathcal{P}'_\mathcal{T}(G)$, and since $v_k v_e \in E$, results to a kPC of size $\lambda_\mathcal{T}(G[S]) - 1$, a contradiction. Suppose now that $P = P'$. Clearly, $v_{i'} v_e \in E(G)$ and $v_{i'} v_k \in E(G)$. Then, if we remove v_n from $\mathcal{P}'_\mathcal{T}(G)$, we obtain a kPC of size $\lambda_\mathcal{T}(G[S]) - 1$, a contradiction.

Case 2: when the algorithm processes vertex v_n, it inserts it into an existing path. Thus, $\lambda_\mathcal{T}(G) = \lambda_\mathcal{T}(G[S])$ and consequently, $\mathcal{P}_\mathcal{T}(G)$ is a minimum kPC of G, that is, $\lambda'_\mathcal{T}(G) = \lambda_\mathcal{T}(G)$. Furthermore, there exists no available endpoint in $N[v_n]$ in $\mathcal{P}_\mathcal{T}(G)$. Let $\mathcal{P}'_\mathcal{T}(G)$ be a minimum kPC of G having an available endpoint $v_\ell \in N[v_n]$, $i \leq \ell \leq n$. Clearly, v_n belongs to path P of $\mathcal{P}'_\mathcal{T}(G)$ containing more than one vertex. If $v_\ell = v_n$, then removing v_n from $\mathcal{P}'_\mathcal{T}(G)$ results to a minimum kPC of $G[S]$ having a neighbor of v_n, say v_t, as an available endpoint, a contradiction. If v_n is an internal node of P, let v_a and v_b be the vertices it is connected to. Clearly, $v_a v_b \in E(G)$ and, thus, removing v_n from $\mathcal{P}'_\mathcal{T}(G)$ results to a minimum kPC of $G[S]$ having v_ℓ as an available endpoint, a contradiction.

Case 3: when the algorithm processes vertex v_n, it creates a new path, that is, $P_{\lambda_\mathcal{T}(G)} = [v_n]$. Thus, $\lambda_\mathcal{T}(G) = \lambda_\mathcal{T}(G[S]) + 1$. Since the algorithm creates a new path, there is no path in $\mathcal{P}_\mathcal{T}(G[S])$ having an available endpoint $v_a \in N(v_n)$ or containing an edge $v_a v_b$ such that $v_a, v_b \in N(v_n)$. Let $\mathcal{P}'_\mathcal{T}(G)$ be a minimum kPC of size $\lambda'_\mathcal{T}(G) = \lambda_\mathcal{T}(G[S])$. If v_n is the only vertex of a path in $\mathcal{P}'_\mathcal{T}(G)$, then removing v_n from $\mathcal{P}'_\mathcal{T}(G)$ results to a kPC of $G[S]$ of size $\lambda_\mathcal{T}(G[S]) - 1$, a contradiction. If v_n is an endpoint of a path in $\mathcal{P}'_\mathcal{T}(G)$, then removing v_n from $\mathcal{P}'_\mathcal{T}(G)$ results to a minimum kPC of $G[S]$ having a neighbor of v_n, say v_t, as an available endpoint; a contradiction. On the other hand, if v_n is an internal node of a path in $\mathcal{P}'_\mathcal{T}(G)$, then it has at least two neighbors, that is $v_{n-1}, v_{n-2} \in N(v_n)$ and $v_{n-1} v_{n-2} \in E(G)$. According to Lemma 3.3, unless vertices v_{n-1} and v_{n-2} are terminal vertices, at least one of the following holds:
(i) v_{n-2}, v_{n-1} are successive in a path in $\mathcal{P}_\mathcal{T}(G[S])$,
(ii) at least one of v_{n-2}, v_{n-1} is an available endpoint.
Since the algorithm constructs a new path, $v_{n-1}, v_{n-2} \in \mathcal{T}$ and they are endpoints at the same path (see Corollary 3.1) which contains at least one more vertex, that is, v_{n-2} is the left endpoint of $P_{\lambda_\mathcal{T}(G[S])}$ and v_{n-1} is its right endpoint. Consequently, if we apply the algorithm to $G[S] - \{v_{n-1}, v_{n-2}\} = G - \{v_{n-1}, v_n, v_{n-2}\}$, we obtain a kPC of size $\lambda_\mathcal{T}(G[S])$, which, by the induction hypothesis, is minimum. Suppose that $N(v_n) = \{v_{n-1}, v_{n-2}\}$. Then, the minimum kPC $\mathcal{P}'_\mathcal{T}(G)$ contains the path $P' = [v_{n-1}, v_n, v_{n-2}]$. Consequently, removing the vertices v_{n-1}, v_n, v_{n-2} from $\mathcal{P}'_\mathcal{T}(G)$ results to a kPC of $G - \{v_{n-1}, v_n, v_{n-2}\}$ of size $\lambda_\mathcal{T}(G[S]) - 1$, a contradiction. Now let $|N(v_n)| > 2$. In this case, according to Lemma 3.4, the algorithm either connects v_n to a path or inserts it into a path and thus, $\lambda_\mathcal{T}(G) = \lambda_\mathcal{T}(G[S])$, a contradiction.

Consequently, $\mathcal{P}_\mathcal{T}(G)$ is a minimum kPC of G, that is, $\lambda'_\mathcal{T}(G) = \lambda_\mathcal{T}(G) = \lambda_\mathcal{T}(G[S]) + 1$. Finally, the rightmost available endpoint in $N[v_n]$ of the kPC constructed by the algorithm is vertex v_n and, clearly, it is the rightmost available endpoint in $N[v_n]$ that any other minimum kPC $\mathcal{P}'_\mathcal{T}(G)$ can contain.

Thus, the theorem follows. ∎

Let G be a proper interval graph on n vertices and m edges and let $\mathcal{T}$ be a terminal set. Then, Algorithm Minimum_kPC computes a minimum kPC $\mathcal{P}_\mathcal{T}(G)$ of G. The time complexity is $O(n + m)$ since vertices p_ℓ and p_r of the algorithm can be obtained in constant time and the operations of the algorithm can be performed in $O(1)$ time for each $v_i, 1 \leq i \leq n$. Note that, no additional space is required and, thus, the space complexity is also linear to the size of the input graph. Furthermore, the canonical ordering is constructed in $O(n + m)$ time [19]. Hence, we can state the following result.

Theorem 4.2. *Let G be a proper interval graph on n vertices and m edges and let $\mathcal{T}$ be a subset of $V(G)$. A minimum k-fixed-endpoint path cover $\mathcal{P}_{\mathcal{T}}(G)$ of G can be computed in $O(n+m)$ time and space.*

5 Concluding Remarks

This paper presents a simple linear-time algorithm for the k-fixed-endpoint path cover problem on proper interval graphs. Our algorithm can be used to solve the 1HP and 2HP problems on proper interval graphs; recall that the Hamiltonian path problem is NP-complete on chordal graphs. An interesting open question would be to see if the k-fixed-endpoint path cover problem can be polynomially solved on other classes of graphs, such as interval graphs. Note that the complexity status of simpler problems, such as 1HP and 2HP, is unknown for interval graphs. Furthermore, our work could probably be used in solving other related problems, such as finding a longest path between two specified vertices of a proper interval graph or computing the longest path on an interval graph.

References

[1] G.S. Adhar and S. Peng, Parallel algorithm for path covering, Hamiltonian path, and Hamiltonian cycle in cographs, *Int'l Conference on Parallel Processing*, Vol. III: Algorithms and Architecture, Pennsylvania State University Press, pp. 364–365, 1990.

[2] S.R. Arikati and C.P. Rangan, Linear algorithm for optimal path cover problem on interval graphs, *Inform. Process. Lett.* **35** (1990) 149–153.

[3] K. Asdre and S.D. Nikolopoulos, A linear-time algorithm for the k-fixed-endpoint path cover problem on cographs, *Networks* **50** (2007) 231–240.

[4] J. Bang-Jensen, J. Huang, and L. Ibarra, Recognizing and representing proper interval graphs in parallel using merging and sorting, *Discrete Appl. Math.* **155** (2007) 442–456.

[5] A.A. Bertossi, Finding Hamiltonian circuits in proper interval graphs, *Inform. Process. Lett.* **17** (1983) 97–101.

[6] A.A. Bertossi and M.A. Bonuccelli, Finding Hamiltonian circuits in interval graph generalizations, *Inform. Process. Lett.* **23** (1986) 195–200.

[7] A. Brandstädt, V.B. Le, and J. Spinrad, *Graph classes – A survey*, SIAM Monographs in Discrete Mathematics and Applications, SIAM, Philadelphia, 1999.

[8] D.G. Corneil, H. Kim, S. Nataranjan, S. Olariu, and A.P. Sprague, Simple linear time recognition of unit interval graphs, *Inform. Process. Lett.* **55** (1995) 99–104.

[9] P. Damaschke, Paths in interval graphs and circular arc graphs, *Discrete Math.* **112** (1993) 49–64.

[10] S. Fortune, J.E. Hopcroft, and J. Wyllie, The directed subgraph homeomorphism problem, *J. Theoret. Comput. Sci.* **10** (1980), 111–121.

[11] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W.H. Freeman, San Francisco, 1979.

[12] M.R. Garey, D.S. Johnson, and R.E. Tarjan, The planar Hamiltonian circuit problem is NP-complete, *SIAM J. Comput.* **5** (1976) 704–714.

[13] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980. Second edition, Annals of Discrete Math. **57**, Elsevier, 2004.

[14] P. Hell, R. Shamir, and R. Sharan, A fully dynamic algorithm for recognizing and representing proper interval graphs, *SIAM J. Comput.* **31** (2001) 289–305.

[15] S.Y. Hsieh, An efficient parallel strategy for the two-fixed-endpoint Hamiltonian path problem on distance-hereditary graphs, *J. Parallel Distrib. Comput.* **64** (2004) 662–685.

[16] R.W. Hung and M.S. Chang, Linear-time algorithms for the Hamiltonian problems on distance-hereditary graphs, *Theoret. Comput. Sci.* **341** (2005) 411–440.

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

[18] J.M. Keil, Finding Hamiltonian circuits in interval graphs, *Inform. Process. Lett.* **20** (1985) 201–206.

[19] P.J. Looges and S. Olariu, Optimal greedy algorithms for indifference graphs, *Comput. Math. Appl.* **25** (1993) 15–25.

[20] T.A. McKee and F.R. McMorris, *Topics in Intersection Graph Theory*, SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, 1999.

[21] H. Müller, Hamiltonian circuits in chordal bipartite graphs, *Discrete Math.* **156** (1996) 291–298.

[22] K. Nakano, S. Olariu, and A.Y. Zomaya, A time-optimal solution for the path cover problem on cographs, *Theoret. Comput. Sci.* **290** (2003) 1541–1556.

[23] S.D. Nikolopoulos, Parallel algorithms for Hamiltonian problems on quasi-threshold graphs, *J. Parallel Distrib. Comput.* **64** (2004) 48–67.

[24] B.S. Panda and S.K. Das, A linear time recognition algorithm for proper interval graphs, *Inform. Process. Lett.* **87** (2003) 153–161.

[25] J.H. Park, One-to-Many disjoint path covers in a graph with faulty elements, *Proc. 10th Internat. Computing and Combinatorics Conference (COCOON'04)*, LNCS 3106 (2004), 392–401.

[26] F.S. Roberts, Graph theory and its applications to problems of society, SIAM Press, Philadelphia, PA, 1978.

[27] N. Robertson and P.D. Seymour, Graph minors. XIII. The disjoint paths problem, *J. Combinatorial Theory*, Series B, **63** (1995), 65–110.

[28] P.D. Seymour, Disjoint paths in graphs, *Discrete Math.* **29** (1980), 293–309.

[29] Y. Shiloach, A polynomial solution to the undirected two paths problem, *J. Assoc. Comput. Mach.* **27** (1980), 445–456.

[30] R. Srikant, R. Sundaram, K.S. Singh, and C.P. Rangan, Optimal path cover problem on block graphs and bipartite permutation graphs, *Theoret. Comput. Sci.* **115** (1993) 351–357.

[31] Y. Suzuki, K. Kaneko, and M. Nakamori, Node-disjoint paths algorithm in a transposition graph, *IEICE Trans. Inform. & Syst.* **E89-D** (2006) 2600–2605.