Discrete Applied Mathematics 313 (2022) 147–178

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Cyclability in graph classes[☆,☆☆]

Christophe Crespelle[a], Petr A. Golovach[b,*]

[a] Université Claude Bernard Lyon 1, INRIA, Lyon, France
[b] Department of Informatics, University of Bergen, Norway

ARTICLE INFO

Article history:
Received 18 March 2021
Received in revised form 10 January 2022
Accepted 21 January 2022
Available online 15 March 2022

Keywords:
Cyclability
Interval graphs
Bipartite permutation graphs
Cographs
Certifying algorithms

ABSTRACT

A subset $T \subseteq V(G)$ of vertices of a graph G is said to be *cyclable* if G has a cycle C containing every vertex of T, and for a positive integer k, a graph G is k-*cyclable* if every set of vertices of size at most k is cyclable. The TERMINAL CYCLABILITY problem asks, given a graph G and a set T of vertices, whether T is cyclable, and the k-CYCLABILITY problem asks, given a graph G and a positive integer k, whether G is k-cyclable. These problems are generalizations of the classical HAMILTONIAN CYCLE problem. We initiate the study of these problems for graph classes that admit polynomial algorithms for HAMILTONIAN CYCLE. We show that TERMINAL CYCLABILITY can be solved in linear time for interval graphs, bipartite permutation graphs and cographs. Moreover, we construct certifying algorithms that either produce a solution, that is a cycle, or output a graph separator that certifies a no-answer. We use these results to show that k-CYCLABILITY can be solved in polynomial time when restricted to the aforementioned graph classes.

1. Introduction

A subset $T \subseteq V(G)$ of vertices of a graph G is said to be *cyclable* if G has a (simple) cycle C containing every vertex of T. In this case, C is said to *cover* T. It is assumed that a single-element set is cyclable. For a positive integer k, a graph G is k-*cyclable* if every set T of size at most k is cyclable. The *cyclability* of G, denoted $\mathsf{cyc}(G)$ (see Fig. 1 for a table summarizing the main notations we use), is the maximum $k \le |V(G)|$ such that G is k-cyclable. We consider the following generalizations of the classical HAMILTONIAN CYCLE problem.

TERMINAL CYCLABILITY

Input:	A graph G and a nonempty set $T \subseteq V(G)$ of *terminals*.
Task:	Decide whether T is cyclable.

☆ We thank Carl Feghali for fruitful discussions on the topic of this paper and participation in the preliminary publication in the conference proceedings.

☆☆ An extended abstract of this paper appeared in the proceedings of ISAAC 2019. This project has received funding from the Research Council of Norway[http://dx.doi.org/10.13039/501100005416][http://dx.doi.org/10.13039/501100005416] via the projects CLASSIS (grant no 249994) and MULTIVAL, Norway (grant no 263317) and the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 749022.

* Correspondence to: University of Bergen, PB 7803, N-5020, Bergen, Norway.
E-mail addresses: christophe.crespelle@inria.fr (C. Crespelle), petr.golovach@uib.no (P.A. Golovach).

https://doi.org/10.1016/j.dam.2022.01.021

T	a set of terminals
$c(G)$	number of connected components of G
$c_T(G)$	number of connected components of G containing a vertex of T
$\mathsf{cyc}(G)$	cyclablity of G, *i.e.* the maximum k such that G is k-cyclable
$\mathsf{sc}(G)$	scattering number of G, *i.e.* $\max\{c(G-S)-\lvert S\rvert \mid S$ is a separator of $G\}$
$\mathsf{sc}_T(G)$	T-scattering number of G, *i.e.* $\max\{c_T(G-S)-\lvert S\rvert \mid S$ is a T-separator$\}$
$\mathsf{seg}_T(G)$	T-cycle-segment-cover number of G, *i.e.* the minimum size of a T-cycle-segment cover
$\mathsf{sc}^k(G)$	k-scattering number of G, *i.e.* $\max\{c(G-S)-\lvert S\rvert \mid S$ is a separator of G s.t. $\lvert S\rvert \leq k-1\}$

Fig. 1. Nomenclature of the main notations we use throughout the paper.

k-Cyclability

Input: A graph G and a positive integer k.
Task: Decide whether G is k-cyclable.

The rational behind the definitions of Terminal Cyclability and k-Cyclability is to address the question of what can be done when the answer to the Hamiltonian Cycle problem is negative. Which subsets of vertices can one include in a single cycle? Is it possible to do so for all subsets of size k? The investigation of these two problems started in the 1960s with the pioneer work of Dirac [18] who proved that for each $k \geq 2$, every k-connected graph is k-cyclable. A number of related results followed, with the majority of them following the line of research initiated by Dirac [18], giving sufficient conditions for the existence of a cycle through a given set or number of vertices; we refer the reader to the survey paper of Gould [23] for results of this type.

From the computational complexity viewpoint, Terminal Cyclability and k-Cyclability are at least as hard as the Hamiltonian Cycle problem, which is well known to be NP-complete [19] (and remains so for several very restricted graph classes such as, for example, bipartite planar cubic graphs and split graphs [1,20]). Positive results can be found in the Parameterized Complexity framework (we refer to the recent book of Cygan et al. [12] for an introduction to the field). For instance, by the celebrated results of Robertson and Seymour [34] about the Disjoint Paths problem, Terminal Cyclability is fixed-parameter tractable (FPT) when parameterized by $|T|$. So far, the best known FPT (randomized) algorithm is due to Björklund, Husfeldt and Taslaman [3]. Golovach et al. [21] also proved that deciding if G is k-cyclable is co-W[1]-hard for split graphs and that k-Cyclability is FPT on planar graphs when parameterized by k.

There is also a long history of research on Hamiltonian Cycle and related problems for the classes of cographs, bipartite permutation graphs and interval graphs and some of their superclasses [5,7,8,11,13–16,25–27,29–31,33].

Let $c(G)$ denote the number of connected components of a graph G. Chvátal [9] observed that if there exists a vertex separator S of a graph G such that $c(G-S) > |S|$, then G has no Hamiltonian cycle. Hence, the condition that $c(G-S) \leq |S|$ holds for every separator of a graph G is a necessary Hamiltonicity condition. For interval graphs, bipartite permutation graphs and cographs that are connected and have at least three vertices, this condition turns out to be also sufficient [11,13,15]. Moreover, motivated by this necessary condition, Jung [28] defined the *scattering* number of a noncomplete graph G as

$$\mathsf{sc}(G) = \max\{c(G-S) - |S| \mid S \text{ is a separator of } G\}, \tag{1}$$

and a set S^* for which the maximum in (1) is achieved is called a *scattering* set. For a complete graph G, $\mathsf{sc}(G) = -\infty$. For the class of cocomparability graphs G with at least three vertices (that is a superclass of the classes of interval graphs and permutation graphs), the following duality was established in [15]. Firstly, the set of vertices of G can be covered by at most k vertex-disjoint paths if and only if $\mathsf{sc}(G) \leq k$. Secondly, G has a Hamiltonian cycle if and only if $\mathsf{sc}(G) \leq 0$.

From these equivalences, one can construct certifying polynomial time algorithms for Hamiltonian Path and Hamiltonian Cycle problems. We refer to the survey papers [2,32] for an introduction to certifying algorithms. A *certifying* algorithm for a decision problem is an algorithm that provides, together with each answer, a *certificate* (or *witness*) that demonstrates the correctness of the answer and that can be verified independently by another algorithm. Such an algorithm, called an *authentication* algorithm (or *checker*), takes as its input an instance of the considered problem as well as the output and the certificate provided by the certifying algorithm for this instance. It then verifies (independently of the original algorithm) whether the output is correct. The authentication should not involve solving the original problem and should be "simple" in some sense meaning that either it should be faster than algorithms solving the problem or it should be possible to apply the formal verification approach to check its correctness. The main advantage of certifying algorithms over standard ones is that their implementations are much more reliable and can be used without knowing the code, because one recognizes whether their outputs are correct even if the implementation is faulty, provided that the authentication algorithm is correct. Certifying algorithms for Hamiltonian Path and Hamiltonian Cycle that either output a Hamiltonian path or a Hamiltonian cycle certifying a yes-answer or produce a separator that certifies a no-answer have been given in [7,10,11,15].

We continue the study of Terminal Cyclability and k-Cyclability from a computational complexity viewpoint by first showing that analogous dualities hold for these problems on interval graphs, bipartite permutation graphs and cographs (see Section 2 for the formal definitions of these graph classes). We will then show how to construct from these dualities polynomial time certifying algorithms for Terminal Cyclability on these graph classes. Further, we construct polynomial time algorithms for k-Cyclability.

In fact, for Terminal Cyclability we will consider a slightly more general problem called Cycle Segment Cover, which is defined as follows. Let G be a graph and let $T \subseteq V(G)$. A cycle or a family of paths containing the vertices of T is said to be a *T-cycle-segment cover*. The *size* of such a T-cycle-segment cover is defined to be zero if it is a cycle and the number of paths of the family otherwise. The *T-cycle-segment-cover number* $\mathsf{seg}_T(G)$ is the minimum size of a T-cycle-segment cover.

Cycle Segment Cover

Input: A graph G, a nonempty set $T \subseteq V(G)$ of *terminals* and an integer $r \geq 0$.
Task: Decide whether $\mathsf{seg}_T(G) \leq r$.

In particular, Terminal Cyclability is the restriction of Cycle Segment Cover with $r = 0$. Denote by $c_T(G)$ the number of components of G containing a vertex of T and say that $S \subseteq V(G)$ is a *T-separator* if $c_T(G-S) \geq 2$. It is easy to show (see Observation 3) that if G has a T-separator S with $c_T(G-S) - |S| > r$ (*i.e.* $\mathsf{sc}_T(G) > r$ from Definition 1), then $\mathsf{seg}_T(G) > r$, which allows us to use such a separator as a certificate of a no-answer for an instance of Cycle Segment Cover. In our first theorem, we show that we can solve in polynomial (and even linear) time Cycle Segment Cover on interval graphs, bipartite permutation graphs and cographs.

Theorem 1. *There is an algorithm that, given an instance (G, T, r) of* Cycle Segment Cover, *where G is either an interval graph, a bipartite permutation graph or a cograph and T is not a 2-clique, outputs in $\mathcal{O}(|V(G)| + |E(G)|)$ time either a T-cycle segment cover of size at most r or a T-separator S^* with $c_T(G - S^*) - |S^*| > r$ that certifies a no-answer.*

Notice that if T is a 2-clique, then this is a special case which is not covered by Theorem 1, because of no-instances (G, T, r) of Cycle Segment Cover that cannot be certified by any T-separator. Nevertheless, such a situation occurs only if $r = 0$ and the vertices of T are the end-vertices of a bridge, which can be recognized in linear time. In all other cases, (G, T, r) is a yes-instance whenever T is a clique.

We then use Theorem 1 to solve k-Cyclability for interval graphs, bipartite permutation graphs and cographs.

Theorem 2. *For an interval graph, a bipartite permutation graph or a cograph G, k-*Cyclability *can be solved in time $\mathcal{O}(|V(G)|^3)$.*

The paper is organized as follows. In Section 2, we present some auxiliary results. In Sections 3–5, we construct our algorithms for Cycle Segment Cover and k-Cyclability on, respectively, interval graphs, bipartite permutation graphs and cographs. We conclude the paper in Section 6 with some open problems.

2. Preliminaries

We consider only finite undirected simple graphs. We use n to denote the number of vertices and m the number of edges of the considered graphs unless it creates confusion. For $U \subseteq V(G)$, we write $G[U]$ to denote the subgraph of G induced by U. We write $G - U$ to denote the graph $G[V(G) \setminus U]$; for a single-element $U = \{u\}$, we write $G - u$. Similarly, for a set of edges S, $G - S$ denotes the graph obtained from G by the deletion of the edges of S; we write $G - e$ instead of $G - \{e\}$ for a single-element set. A set of vertices $U \subseteq V(G)$ is *connected* if $G[U]$ is a connected graph. For a vertex v, we denote by $N_G(v)$ the *(open) neighborhood* of v in G, i.e., the set of vertices that are adjacent to v in G. For a set $U \subseteq V(G)$, $N_G(U) = (\bigcup_{v \in U} N_G(v)) \setminus U$. We denote by $N_G[v] = N_G(v) \cup \{v\}$ the *closed neighborhood* of v and $N_G[U] = \bigcup_{v \in U} N_G[v]$ for a set of vertices U. A vertex v is *universal* if $N_G[v] = V(G)$. The *degree* of a vertex v is $d_G(v) = |N_G(v)|$. We say that a set of vertices $X \subset V(G)$ is a *separator* or *cut-set* of a graph G if $G - X$ has more components than G. It is also convenient for us to assume that the empty set is a separator of a disconnected graph. A vertex u is a *cut-vertex* if $\{u\}$ is a separator. A connected graph G is *2-connected* if it has no cut-vertex. A *block B* of a graph G is an inclusion maximal connected subgraph which does not contain a cut-vertex. Clearly, a block of a connected graph with at least one edge is either a single edge called a *bridge* of G, or is a 2-connected subgraph with at least three vertices, and we say that the block is nontrivial in this case. A *clique* of a graph G is a set of pairwise adjacent vertices. Recall that a *path P* in a graph G is a connected subgraph whose vertices except at most two of them, called the *end-vertices*, have degree two and the end-vertices have degree one or zero if P is a single-vertex path (called *trivial*). We refer to the vertices of degree two of a path as *internal*. We write $P = v_1 \cdots v_s$ to denote the path with the vertices $v_1, \ldots, v_s$ such that $v_{i-1}v_i \in E(P)$ for $i \in \{2, \ldots, s\}$. Note that this notation defines an ordering of vertices and we say that $\langle v_1, \ldots, v_s \rangle$ is the *path ordering* of $V(P)$. We denote by P_1P_2 the concatenation of two vertex-disjoint paths P_1 and P_2. A *subpath* is a connected subgraph of a path P. A *cycle C* in a graph G is a connected subgraph where each vertex has degree two. We write $C = v_0 \cdots v_s$ to denote that C is the cycle consisting of distinct vertices $v_1, \dots, v_s$ such that $v_0 = v_s$ and $v_{i-1}v_i \in E(C)$ for $i \in \{1, \dots, s\}$. A path $P = v_1 \cdots v_s$ (a cycle $C = v_0 \cdots v_s$) in G is *Hamiltonian* if $\{v_1, \dots, v_s\} = V(G)$.

A graph G is an *interval* graph if there is a family $\mathcal{I}$ of closed intervals of the line (called *interval model* or *representation*) such that G is isomorphic to the intersection graph of $\mathcal{I}$. A graph G is a *permutation* graph if it has an intersection model consisting of straight segments between two parallel lines. Equivalently, G is a permutation graph if there is an ordering $\langle v_1, \dots, v_n\rangle$ of its vertices and a permutation $\pi : \{1, \dots, n\} \to \{1, \dots, n\}$ such that for $1 \le i < j \le n$, v_i and v_j are adjacent in G if and only of $\pi(i) > \pi(j)$. A graph is a *bipartite permutation* graph if it is both a bipartite graph and a permutation graph. A graph G is a *cograph* if it has no induced subgraph isomorphic to the path on four vertices. We refer to [6,22] for detailed introductions to these graph classes.

For the proof of Theorem 1, we need the following definition.

Definition 1. For $T \subseteq V(G)$ that is not a clique, the *T-scattering* number of G is given by

$$\mathsf{sc}_T(G) = \max\{c_T(G - S) - |S| \mid S \text{ is a } T\text{-separator}\}, \tag{2}$$

and a set S^* for which the maximum in (2) is achieved is called a T-scattering set.

Note that if T is a clique, then G has no T-separator.

We use the following observation.

Observation 3. *Let $T \subseteq V(G)$ be a set of vertices that is not a clique, Then $\mathsf{sc}_T(G) \le \mathsf{seg}_T(G)$.*

Proof. Let $r = \mathsf{seg}_T(G)$ and let S^* be a T-scattering set. By definition, $G - S^*$ has $p = |S^*| + \mathsf{sc}_T(G) \ge 2$ connected components $G_1, \dots, G_p$ containing vertices of T.

Assume first that $r = 0$, that is, G has a cycle C that contains the vertices of T. Clearly, every cycle visiting $G_1, \dots, G_p$ has at least p vertices in S^*, since S^* separates these connected components of $G - S^*$. Hence, $|S^*| \ge |S^* \cap V(C)| \ge p = |S^*| + \mathsf{sc}_T(G)$ and $\mathsf{sc}_T(G) \le 0 = \mathsf{seg}_T(G)$.

Now, if $r \ge 1$, there exist r vertex-disjoint paths $P_1, \dots, P_r$ covering T in G. One can then add r edges to graph G in order to link the end-vertices of these r paths and form a cycle. The resulting graph G' therefore satisfies $\mathsf{seg}_T(G') = 0$ and from what precedes, we get $\mathsf{sc}_T(G') \le 0$. Finally, observe that adding an edge in a graph can decrease its T-scattering number by at most one. This implies that $\mathsf{sc}_T(G) \le r = \mathsf{seg}_T(G)$. □

To prove Theorem 2, we also need the following definition.

Definition 2. For a positive integer k, define the *k-scattering* number of a graph G as

$$\mathsf{sc}^k(G) = \max\{c(G - S) - |S| \mid S \text{ is a separator of } G \text{ s.t. } |S| \le k - 1\}, \tag{3}$$

and $\mathsf{sc}^k(G) = -\infty$ if it has no separator of size at most $k-1$ (we assume that the empty set is a separator of a disconnected graph).

We have the following observation that generalizes the necessary Hamiltonicity condition of Chvátal [9].

Observation 4. *If G is a graph with at least 3 vertices that is k-cyclable for some $k \ge 3$, then $\mathsf{sc}^k(G) \le 0$.*

Proof. By contrapose, let G be a graph such that $\mathsf{sc}^k(G) > 0$. From Definition 2, G contains some separator of size at most $k - 1$, since otherwise $\mathsf{sc}^k(G) = -\infty < 0$. Consequently, from Definition 2 again, G contains a separator S of size at most $k - 1$ such that $c(G - S) > |S|$. Form a set $T \subseteq V(G)$ by picking exactly $|S| + 1$ vertices, each of them being chosen from a different component of $G - S$. We have that T is not a clique, S is a T-separator of G, and $c_T(G - S) > |S|$. This gives $\mathsf{sc}_T(G) > 0$ and so $\mathsf{seg}_T(G) > 0$, from Observation 3, implying that T is not cyclable in G. Consequently, since $|T| = |S| + 1 \le k$, G is not k-cyclable. □

We establish a number of auxiliary results that will allow us to dispense with easy instances of our problems. We start with the case $|T| \le 2$ for Cycle Segment Cover.

Proposition 5. *There is an algorithm for* Cycle Segment Cover *for $|T| \le 2$ that in linear time*[1] *does the following:*

- *if $|T| = 1$, it returns a trivial solution that is the single vertex of T;*
- *if $r \ge 2$ and $|T| = 2$, it returns a trivial solution made of two trivial paths, each of them containing one single vertex that belongs to T;*
- *if $r = 1$ and $|T| = 2$, it either returns a path containing both elements of T or reports that the elements of T are in distinct components of G;*

[1] Throughout the paper, linear time means $O(n + m)$ time, *i.e.* linear in the size of the adjacency lists of the input graph.

- *if $r = 0$ and $|T| = 2$, it either returns a cycle containing T, or a set $S \subseteq V(G)$ with $|S| \leq 1$ such that the vertices of T are in distinct components of $G - S$, or a trivial block of G containing the elements of T.*

Proof. The first three items are obvious. The last claim follows from the fact that a graph has a cycle containing two distinct vertices u and v if and only if u and v are in the same nontrivial block (see, e.g., [17]). Recall that all the cut-vertices and blocks of a graph can be found in linear time by the algorithm of Hopcroft and Tarjan [24]. Hence, it can be decided in linear time whether u and v are on a same cycle. Note that the algorithm of Hopcroft and Tarjan can be easily modified to find such a cycle in linear time if it exists. Otherwise, if u and v are not in the same nontrivial block, they are either end-vertices of the edge composing a trivial block or there is a cut-vertex x such that u and v are in distinct components of $G - x$. $\square$

Proposition 6. *k-Cyclability can be solved in linear time if $k \leq 2$.*

Proof. Every graph is, by definition, 1-cyclable and from Menger's theorem (see, e.g. [17]), it follows that a graph G is 2-cyclable if and only if G is a 2-connected graph with at least three vertices. Since 2-connectedness can be checked in linear time [24], the result follows. $\square$

Similarly, Cycle Segment Cover is trivial for the special case when T is a clique.

Observation 7. *An instance (G, T, r) of Cycle Segment Cover, where T is a clique and $|T| \neq 2$, is a yes-instance with a solution that is an arbitrary Hamiltonian cycle of the complete graph $G[T]$ (or a trivial solution if $|T| = 1$).*

If G is a complete graph, then k-Cyclability is also trivial.

Observation 8. *An instance (G, k) of k-Cyclability, where G is a complete graph distinct from K_2 is a yes-instance with a solution that is an arbitrary Hamiltonian cycle of the complete graph G (or a trivial solution if $|V(G)| = 1$).*

We conclude this section with a lemma (Lemma 9) that will be useful in solving k-Cyclability for the considered graph classes. We say that a graph class $\mathcal{C}$ is *cycle-scattering dual* if the duality theorem between the T-cycle-segment-cover number and the T-scattering number holds on class $\mathcal{C}$, as expressed formally in the following definition.

Definition 3. $\mathcal{C}$ is *cycle-scattering dual* if for every $G \in \mathcal{C}$ and nonempty $T \subseteq V(G)$ that is not a clique, $\mathsf{seg}_T(G) = \max\{\mathsf{sc}_T(G), 0\}$.

Lemma 9. *Let $\mathcal{C}$ be a cycle-scattering dual graph class. For each $k \geq 3$, a graph $G \in \mathcal{C}$ with at least three vertices is k-cyclable if and only if $\mathsf{sc}^k(G) \leq 0$.*

Proof. If G is k-cyclable, then $\mathsf{sc}^k(G) \leq 0$ by Observation 4. Conversely, suppose that $\mathsf{sc}^k(G) \leq 0$. Assume, without loss of generality, that G contains at least k vertices. Let $T \subseteq V(G)$ have size k. If T is a clique, then T is cyclable by Observation 8. Suppose that T is not a clique. Then there is a T-separator S in G. If $|S| \geq k$, then $c_T(G - S) \leq k \leq |S|$. If $|S| \leq k - 1$, then $c_T(G - S) \leq c(G - S) \leq |S|$, because $\mathsf{sc}^k(G) \leq 0$. Therefore, $\mathsf{sc}_T(G) \leq 0$. Moreover, since $\mathcal{C}$ is a cycle-scattering dual graph class, $\mathsf{seg}_T(G) = \max\{\mathsf{sc}_T(G), 0\} = 0$ and so T is cyclable. But as T was an arbitrary set of k vertices, G is k-cyclable. $\square$

3. Interval graphs

In this section, we prove Theorems 1 and 2 for interval graphs.

Our algorithms use a specific interval representation of the input graph. A *clique path* of a graph G is a sequence of cliques $C_1, \ldots, C_s$ of G such that

(i) $C_1 \cup \cdots \cup C_s = V(G)$,
(ii) for every $uv \in E(G)$, there is $i \in \{1, \ldots, s\}$ such that $u, v \in C_i$,
(iii) for every $v \in V(G)$, if $v \in C_i \cap C_j$ for some $1 \leq i < j \leq s$, then $v \in C_h$ for $h \in \{i, \ldots, j\}$.

It is usually assumed in the definition of a clique path (see, e.g., [6,22]) that $C_1, \ldots, C_s$ are maximal cliques of G. Here, we relax the standard definition and do not require the cliques to be inclusion-wise maximal. In particular, some cliques may be identical or empty. It is well-known [6,22] that a graph is an interval graph if and only if it has a clique path. The classical recognition algorithm for interval graphs of Booth and Lueker [4] constructs a clique path in time $\mathcal{O}(n + m)$. So we can assume from now on that the input graph is given with its clique path.

For a vertex $v \in V(G)$, we let $\ell_v = \min\{i \in \{1, \ldots, s\} \mid v \in C_i\}$ and $r_v = \max\{i \in \{1, \ldots, s\} \mid v \in C_i\}$. We say that ℓ_v and r_v are the *left bound* and *right bound* of v respectively. Notice that the intervals $[\ell_v, r_v]$ of the real line for $v \in V(G)$ form an interval representation of G. For $1 \leq i \leq j \leq s$, we denote $C_{i,j} = \bigcup_{h=i}^{j} C_h$.

We use the following well-known observation about separators of interval graphs that results from the definition of a clique path (see, e.g., [6,22]).

Observation 10. *Let G be a connected interval graph with a clique path $C_1, \ldots, C_s$. If $X = C_{1,i} \setminus C_{i+1} \neq \emptyset$ and $Y = C_{i+1,s} \setminus C_i \neq \emptyset$ for some $i \in \{1, \ldots, s-1\}$, then $C_i \cap C_{i+1}$ is a separator of G such that X and Y are in distinct components of $G - (C_i \cap C_{i+1})$.*

For clique paths composed of maximal cliques, we have a stronger property.

Proposition 11 ([6,22]). *Let G be an interval graph with a clique path $C_1, \ldots, C_s$ where $C_1, \ldots, C_s$ are pairwise distinct maximal cliques of G. Let also $u \in C_i$ and $v \in C_j$ be nonadjacent vertices for $1 \leq i < j \leq s$. Then $X \subseteq V(G)$ is an inclusion minimal $\{u, v\}$-separator if and only if $X = C_h \cap C_{h+1}$ for some h such that $r_u \leq h < l_v$.*

In Section 3.1 we solve Cycle Segment Cover for interval graphs. In Section 3.2, we show how to compute the k-scattering number for interval graphs and use this result to solve k-Cyclability.

3.1. *Certifying algorithm for* Terminal Cyclability *and* Cycle Segment Cover

In this subsection we construct an algorithm for Terminal Cyclability and then explain how to obtain an algorithm for Cycle Segment Cover as a corollary. By Observation 7, it is sufficient to solve Terminal Cyclability for a set of terminals T that is not a clique.

Theorem 12. *There is an algorithm that, given an instance (G, T) of* Terminal Cyclability *where G is an interval graph and T is not a clique, in time $\mathcal{O}(n+m)$ returns either a cycle of G covering T or a T-separator S^* with $c_T(G - S^*) - |S^*| > 0$ that certifies a no-answer.*

The next part of the subsection contains the proof of Theorem 12. We construct an algorithm that tries to finds a cycle of a graph G that covers T. If it fails to do it, we use the information obtained by the algorithm to construct a T-separator. The algorithm is inspired by the algorithm for finding a Hamiltonian cycle in interval graphs of Keil [29]. For us, it is more convenient to use a specially tailored variant of Algorithm 1 in [7] for a more general problem as this allows us to use some results of [7] as black boxes. For this, we need some auxiliary results.

Let G be an interval graph given together with its clique path $C_1, \ldots, C_s$, and let $T \subseteq V(G)$ such that T is not a clique. If G has at least two distinct components containing vertices of T, then G has no cycle covering T and the algorithm returns $S^* = \emptyset$. We can thus assume that the vertices of T are in the same component, so we can discard the other components if they exist. Clearly, all this can be done in linear time. So we can safely assume from now on that G is connected. Our algorithm (Algorithm 1) scans the clique path of G and selects vertices depending on their bounds. In order to break ties between vertices having the same right bound (Lines 4 and 9) or the same left bound (Line 14), we use a pre-decided arbitrary total order π on the vertices of G and always select the least possible vertex in π; in the first iteration, when $P_1 = P_2$, the algorithm chooses P_1. We define $p = \min\{r_v \mid v \in T\}$ and $q = \max\{\ell_v \mid v \in T\}$. Let w_b be the minimum vertex of T with respect to π such that $r_{w_b} = p$ and analogously, let w_e be the maximum vertex of T with respect to π such that $\ell_{w_e} = q$. Since T is not a clique, it follows that $p < q$, $w_b \neq w_e$ and $w_b w_e \notin E(G)$.

Algorithm 1 tries to construct two (w_b, w_e)-paths P_1 and P_2 that are internally vertex-disjoint such that $T \subseteq V(P_1) \cup V(P_2)$. If the algorithm succeeds, then the concatenation of P_1 and P_2 is a cycle covering T. Initially, $P_1 = P_2 = w_b$. Along the algorithm, new vertices are attached to the two paths. Each of these new vertices is attached to the end-vertex of only one of the two paths, which we call the *extremity* of the path. For each P_i, the initial extremity of P_i is w_b and whenever we append a new vertex to the path, this vertex becomes the new extremity.

Lemma 13. *If Algorithm 1 returns P_1 and P_2, then P_1 and P_2 are internally vertex-disjoint (w_b, w_e)-paths that contain all the vertices of T.*

Proof. Assume that Algorithm 1 returned P_1 and P_2. Since P_1 and P_2 are constructed by attaching vertices that have not been included in P_1 and P_2 in previous steps, P_1 and P_2 are paths. Notice that both P_1 and P_2 initially contain one single vertex which is w_b (Line 2) and that w_e is attached to both paths at Line 14 of the algorithm. Hence, P_1 and P_2 are (w_b, w_e)-paths. Because for each $t \in \{p, \ldots, q\}$, the vertices $x \in T$ such that $r_x = t$ that have not been previously attached to some path are attached to a path at Line 5, $T \subseteq V(P_1) \cup V(P_2)$. □

Our next aim is to show that if Algorithm 1 reports that T is not cyclable, then there is a T-separator S^* such that $c_T(G - S^*) > |S^*|$. The main observation that we shall use to construct the set S^* is that for $T = V(G)$, Algorithm 1 is precisely an algorithm for finding a Hamiltonian cycle in an interval graph. Our algorithm can be interpreted, to a large extent, as a variant of Keil's algorithm [29] or of Algorithm 1 of Broersma et al. [7] (the main difference between our algorithm and theirs is that our algorithm does not try to include, in the constructed paths, all vertices that it encounters). In particular, in [7] an explicit construction is given of a separator S of G such that $c(G-S) > |S|$ for the case when G has no Hamiltonian cycle. We adapt their approach by first altering our graph so as to allow the use of some of their results. The rest of our argument is connected to the one in [7] but has its own features and is more than just a variation.

Assume that Algorithm 1 stops at Line 10 for $t = t^*$. Note that from the range of variation of t in the main loop (Line 3), we have $t^* < q$. Denote by P_1^* and P_2^* the paths constructed by the algorithm before it quits. Before we can proceed, we need some further notation from [7].

Algorithm 1: An algorithm for interval graphs that finds two internally vertex-disjoint (w_b, w_e)-paths P_1 and P_2 such that $T \subseteq V(P_1) \cup V(P_2)$.

```
begin
    let $P_1 = P_2 = w_b$;
    for $t = p$ to $q - 1$ do
        choose $P_i \in \{P_1, P_2\}$ such that the extremity of $P_i$ has the leftmost right bound;
        attach the vertices $x \in T \setminus (V(P_1) \cup V(P_2))$ s.t. $r_x = t$ to $P_i$;
        for $i = 1, 2$ do
            if the extremity of $P_i$ has right bound at most $t$ then
                if the subset of vertices $y \in (C_t \cap C_{t+1}) \setminus (V(P_1) \cup V(P_2))$ is not empty
                    then extend $P_i$ by attaching such a $y$ having the leftmost right bound;
                    else report that $T$ is not cyclable and quit;
            end
        end
    end
    attach the vertices $x \in T \setminus \{w_e\}$ s.t. $l_x = q$ to $P_1$, then attach $w_e$ to both $P_1$ and $P_2$;
    return $P_1$ and $P_2$;
end
```

For real numbers $a \leq b$, $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$, $[a, b) = \{x \in \mathbb{R} \mid a \leq x < b\}$, and $(a, b) = \{x \in \mathbb{R} \mid a < x < b\}$. If vertex u has been processed by the algorithm and attached to a path at some step t of the for loop at Lines 3–13, we say that u has been *activated at time* $a_u = t$. We define $a_{w_b} = p$. If u is activated and a vertex v has been attached to u at some step $t' \geq t$ of the for loop, we say that u has been *deactivated at time* $d_u = t'$. Thus, $\ell_u \leq a_u \leq d_u \leq r_u$ and u is said to be *free*, *active* or *depleted* on, respectively, the intervals $[\ell_u, a_u)$, $[a_u, d_u)$ and $[d_u, r_u]$. Note that some of these intervals may be empty. Whenever we say that u is free (respectively, active or depleted) on an interval I of the real line, this means that $I \subseteq [\ell_u, a_u)$ (respectively, $I \subseteq [a_u, d_u)$ or $I \subseteq [d_u, r_u]$). We also say that $v \in V(P_i^*)$ for $i \in \{1, 2\}$ is a *descendant* of $u \in V(P_i^*)$ if v was attached to P_i^* after u and that v is the *last descendant* on an interval I if v is the last vertex attached to P_i^* at steps $t \in I$ of the for loop at Lines 3–13. A vertex v is said to be *renounced* if it is missed by the algorithm, that is, $\ell_v \leq t^*$ and $v \notin V(P_1^*) \cup V(P_2^*)$. The set of renounced vertices is denoted by R.

Let $G^* = G - R$, and let $T^* = V(G) \setminus R$. For $i \in \{1, \dots, s\}$, denote $C_i^* = C_i \setminus R$. Clearly, $C_1^*, \dots, C_s^*$ is a clique path of G^*. Recall that for $1 \leq i \leq j \leq s$, $C_{i,j}^* = \bigcup_{h=i}^{j} C_h^*$.

The description of Algorithm 1, with tie-breaking order 1 π, implies the following property.

Lemma 14. *Algorithm 1 for the instance (G^*, T^*) of* Terminal Cyclability, *with tie-breaking order π, quits at Line 10 and constructs the paths P_1^* and P_2^*.*

As mentioned above, the fact that our Algorithm 1 for (G^*, T^*) works along the same lines as Algorithm 1 of [7] will allow us to use the following Lemma 2.2 of [7]. We provide the proof for completeness.

Lemma 15 ([7])**.** *Let $t \in \{p, \dots, q - 1\}$ such that Algorithm 1 with input (G^*, T^*) either finishes iteration t of the for loop at Lines 3–13 or terminates at Line 10 within iteration t. If there is at least one depleted vertex on the interval $(t, t + 1)$, then there exists an integer t' with $p \leq t' < t$ and with the following properties:*

(i) $C_{t'+1,t}^* \setminus (C_{t'}^* \cup C_{t+1}^*) \neq \emptyset$,
(ii) there exists a unique vertex $u \in C_{t'}^* \cap C_{t+1}^*$ such that u is active on $(t', t' + 1)$ and u is depleted on $(t, t + 1)$,
(iii) all vertices that are active on $(t, t+1)$ are also active on $(t', t'+1)$, with the only possible exception of the last descendant v of u on $(t', t + 1)$ which may be free on $(t', t' + 1)$,
(iv) all vertices that are depleted on $(t, t + 1)$ are also depleted on $(t', t' + 1)$, except u which is active on $(t', t' + 1)$,
(v) all vertices that are active on $(t', t' + 1)$ are also active on $(t, t + 1)$, except u which is depleted on $(t, t + 1)$, and
(vi) all vertices that are free on $(t', t' + 1)$ are also free on $(t, t + 1)$, with the only possible exception of v if it is active on $(t, t + 1)$.

Proof. The proof is almost identical to the proof of Lemma 2.2 of [7].

Assume that there is at least one depleted vertex during the interval $(t, t + 1)$. Notice that $t > p$, because there is no vertex with the right bound greater than p which is deactivated within the first iteration. Let u be a vertex with the latest deactivation time among those that are depleted during $(t, t + 1)$. This vertex is included in one of the paths constructed by the algorithm. Without loss of generality, for the rest of the proof, assume that u is in P_1.

We will show that this vertex u is unique. Notice that all but at most one of the vertices deactivated during a given iteration of the loop on lines 3–13 (say, at time t) have the right bound equal to t and hence cannot be depleted during a nonempty interval. The only possible exception is the extremity of the path P_1 chosen at line 4 (and only if it is deactivated due to attaching a vertex to P_1 at the next line).

We define Q to be the subpath of P_1 constructed within iteration t formed by all descendants of u, except that if the last descendant v of u is active during $(t, t+1)$, we do not include v in Q. Observe that the successor of u has the same deactivation time as u, hence it is distinct from v, and therefore Q is nonempty. Let ℓ_Q be the leftmost left bound among the left bounds of vertices of Q, and let r_Q be the rightmost right bound.

If P_1 has a vertex that is active during $(t, t+1)$, this vertex is v and it is not a vertex of Q. Thus, all vertices of Q are either depleted during $(t, t+1)$ or their right bounds are at most t. By the choice of u, none of them belongs to C^*_{t+1}, and hence $r_Q \leq t$. We choose $t' = \min\{\ell_Q - 1, p\}$. Notice that for $w \in V(Q)$, $r_w \geq d_u$. Thus, if we let w be the vertex of Q such that $\ell_w = \ell_Q$, then w is free during $(t'+1, d_u)$.

Observe that all vertices of Q are in $C^*_{t'+1,t} \setminus (C^*_{t'} \cup C^*_{t+1})$. Hence, this set is not empty and property (i) is proved.

To show (ii), observe that since the deactivation of u happened when its successor x in P_1 was free, we have $d_u \geq \ell_x$. Hence, x cannot be depleted during $(t', t'+1)$. Observe that $u \neq w_b$, as w_b is not depleted during $(t-1, t)$. Therefore, u has a predecessor in P_1. Denote it by u'. If u' was adjacent to the vertex w of Q, then the algorithm would choose w as the successor of u', since $r_{u'} > r_Q \geq r_w$. Consequently, the left bound of u' is at most t', so u is active during $(t', t'+1)$. The uniqueness of u will follow easily once we establish property (iv).

To show (iii), assume that y is a vertex different from v that is active during $(t, t+1)$ but has been activated after t'. Note that y is in P_2. Since w_b is not active during $(t, t+1)$, $y \neq w_b$ and y has a predecessor y'. We first suppose that y is active during $(d_u - 1, d_u)$. The vertex y' is deactivated at some time t'' such that $t'+1 \leq t'' \leq d_u - 1$. Hence, it is adjacent to the previously defined vertex w of Q that is free during $(t'+1, d_u)$. Since $r_w \leq r_Q < t+1 \leq r_y$, the successor of y' should be w rather than y, a contradiction.

It follows that y is not active during $(d_u - 1, d_u)$. The path P_2 contains a vertex y'' that is active during $(d_u - 1, d_u)$, where y is a descendant of y''. Observe that y'' is not active during $(t, t+1)$ because y is. Suppose that the right bound of y'' is at least $t+1$. Then y'' is depleted during $(t, t+1)$, so by the choice of u, y'' is deactivated before time d_u and cannot be active during $(d_u - 1, d_u)$, a contradiction.

Thus, the right bound of y'' is not larger than t. But then $P_2 \ni y''$ should have been chosen at line 4 of the algorithm instead of $P_1 \ni u$.

For (iv), assume that some $z \neq u$ is depleted during $(t, t+1)$, but $d_z \geq t'+1$. By the choice of u, we have $d_z < d_u$. Without loss of generality, assume that z was chosen such that d_z is maximum. Note that z is a vertex of P_2. If P_2 contains a vertex z' that is active during $(t, t+1)$, then by (iii), z' is active during $(t', t'+1)$ and we conclude that z cannot be included in P_2; a contradiction.

It follows that no vertex of P_2 is active during $(t, t+1)$, that is, $t = t^*$. Moreover, by the choice of z, the right bounds of all its descendants are at most t, because if there is a descendant z' of z with $r_{z'} \geq t+1$, then z' is depleted during $(t, t+1)$ and $d_{z'} > d_z$, a contradiction. The path P_2 must contain a vertex that is active during $(d_u - 1, d_u)$. However, this vertex has a right bound smaller than the one of u, contradicting the correct execution of the algorithm at line 4.

To obtain (v), assume that $a \neq u$ is active during $(t', t'+1)$ but not active during $(t, t+1)$. The vertex a is included in P_2. If one of the descendants of a is active during $(t, t+1)$, then by (iii), this vertex is active during $(t', t'+1)$ contradicting the activeness of a at the same time. Similarly, if a or one of its descendants is depleted during $(t, t+1)$, then by (iv), this vertex is depleted during $(t', t'+1)$ and a cannot be active. It follows that the right bounds of a and its descendants are less than or equal to t. Note that P_2 has a vertex that is active during $(d_u - 1, d_u)$. Therefore, P_2 should be selected by the algorithm at line 4 instead of P_1 (whose extremity is u); a contradiction.

It remains to prove (vi). Let b be a vertex that is free during $(t', t'+1)$ and not free during $(t, t+1)$. Moreover, we assume that $b \neq v$ if v is active during $(t, t+1)$. Our algorithm does not terminate until time t. Therefore, b is included in P_2 which has a vertex that is active during $(t', t'+1)$. By (v), this vertex remains active until $t+1$, but it means that b is not included in P_2. □

It is useful to have one additional property (vii).

Lemma 16. *Let $t \in \{p, \ldots, q-1\}$ such that Algorithm 1 with input (G^*, T^*) either finishes iteration t of the for loop at Lines 3–13 or terminates at Line 10 within iteration t. If there is at least one depleted vertex on the interval $(t, t+1)$, then there exists an integer t' with $p \leq t' < t$ that satisfies the conditions (i)–(vi) of Lemma 15 and the following additional property:*

(vii) there is $x \in V(G^*)$ such that $a_x = t'$ and x is active during $(t', t'+1)$.

Proof. By Lemma 15, there is $t' < t$ such that the conditions (i)–(vi) are satisfied. We choose the minimum t' that satisfies (i)-(vi) and show that (vii) holds for t'.

Let us first show that there exists a vertex $x \in V(G^*)$ such that $a_x = t'$. Assume, towards a contradiction, that there is no x with $a_x = t'$. We prove that (i)–(vi) are satisfied for $t'' = t' - 1$, which will contradict the minimality of t'.

For condition (i), observe that $C^*_{t'+1,t} \setminus (C^*_{t'} \cup C^*_{t+1})$ is exactly the subset of vertices that are involved in some of the cliques between $C^*_{t'+1}$ and C^*_t and in no clique out of this range. Therefore, since $t'' < t'$, we have $C^*_{t'+1,t} \setminus (C^*_{t'} \cup C^*_{t+1}) \subseteq C^*_{t''+1,t} \setminus (C^*_{t''} \cup C^*_{t+1})$, implying that condition (i) holds for t'' as well.

If $y \in V(G^*)$ is active on $(t', t'+1)$, then y is active on $(t'', t''+1)$ given that $a_y \neq t'$. This immediately implies that (ii) and (iii) are satisfied. If $y \in V(G^*)$ is depleted on $(t', t'+1)$, then y is depleted on $(t'', t''+1)$ because no vertex was activated at time t'. Therefore, (iv) holds. If $y \in V(G^*)$ is active on $(t'', t''+1)$, then y is still active on $(t', t'+1)$ as no vertex was activated at time t' by the assumption we made towards a contradiction. Hence, (v) holds. Finally, suppose that $y \in V(G^*)$ is free on $(t'', t''+1)$. Then it is either free on $(t', t'+1)$ or $r_y = t'$ as it was not activated at time t'. Since every vertex of G^* that has a right bound before t^* should be activated at some time, then we have $r_y > t'$ and y is free on $(t', t'+1)$. This implies that (vi) holds. Therefore, (i)-(vi) are satisfied for $t'' = t' - 1$, giving the desired contradiction.

Since there exists some vertex activated at time t', consider one such vertex x that is the last to be attached to one of the two paths at time $a_x = t'$. If x is not active on $(t', t'+1)$ (that is, $d_x = t'$) then the fact that $t' < t \leq t^*$ implies that the algorithm must have attached some vertex y to the path containing x, at time t' and after x was attached. But this implies that $a_y = t'$, contradicting the choice of x. $\square$

We now use Lemma 16 to construct the following decreasing sequence $t_1, t_2, \dots$ of positive integers. We set $t_1 = t^*$. Then we construct t_{i+1} from the already constructed t_i as follows. If, for $t = t_i > p$, there is at least one depleted vertex on $(t, t+1)$, then find $t' < t$ such that the conditions (i)–(vii) of Lemmas 15 and 16 are satisfied and set $t_{i+1} = t'$. We stop the construction if there is no depleted vertex on $(t, t+1)$ for $t = t_i$.

Clearly, the constructed sequence is finite and we denote by k the number of its elements, that is, the sequence is $t_1, \dots, t_k$.

For $i \in \{1, \dots, k\}$, we define $S_i = C^*_{t_i} \cap C^*_{t_i+1}$ and let $S^* = \bigcup_{i=1}^{k} S_i$. We use the following crucial property of S^* that was shown in the proof of Theorem 2.1 of [7]. We provide the proof for completeness.

Lemma 17 ([7]). *The set S^* is a separator of G^* and*

$$c(G^* - S^*) \geq k + 1 > |S^*|.$$

Proof. The subgraphs $G^*[C^*_{1,t_k}] - S^*$ and $G[C^*_{t_1+1,s}] - S^*$ contain w_b and w_e, respectively; in particular, they have at least one component each. By Lemma 15(i), $G^*[C^*_{t_{i+1}+1,t_i}] - S^*$ has at least one component for each $i \in \{1, \dots, k-1\}$. Since all these components are distinct components of $G^* - S^*$, the graph $G^* - S^*$ has at least $k+1$ components.

By Lemma 15 (ii), (v) and (vi), $(C^*_{t_{i+1}} \cap C^*_{t_{i+1}+1}) \setminus (C^*_{t_i} \cap C^*_{t_i+1})$ contains only vertices that are depleted during $(t_{i+1}, t_{i+1}+1)$ for each $i \in \{1, \dots, k-1\}$. Further, $C^*_{t_1} \cap C^*_{t_1+1}$ has no vertices that are free during $(t, t+1)$, because at least one path is not extendable at time t_1. Also this set has at most one vertex that is active during $(t, t+1)$. Hence, the remaining vertices are depleted. By Lemma 15 (ii) and (iv), for each $i \in \{1, \dots, k-1\}$, exactly one vertex that is depleted during (t_i, t_{i+1}) has a different status during $(t_{i+1}, t_{i+1}+1)$ and is active. It follows that $|S^*| \leq 1 + (k-1) = k < c(G^* - S^*)$. $\square$

From Lemma 17, we can establish an essential result for the proof of Theorem 12.

Lemma 18. *The set S^* is a T-separator in G and*

$$c_T(G - S^*) > |S^*|.$$

Mindful of Lemma 17, Lemma 18 intuitively states that the set R of renounced vertices of G does not play an important role in finding a T-separator of G whose removal "maximizes" the number of resulting components containing some member of T.

Proof. Let $X_k = C^*_{1,t_k} \setminus C^*_{t_k+1}$, $X_i = C^*_{t_{i+1}+1,t_i} \setminus (C^*_{t_{i+1}} \cup C^*_{t_i+1})$ for $i \in \{1, \dots, k-1\}$ and $X_0 = C^*_{t_1+1,s} \setminus C^*_{t_1}$. We have two claims.

Claim 19. $c_T(G^* - S^*) \geq k + 1$.

It suffices to prove that each X_i has nonempty intersection with T. Let us first argue that $w_b \in X_k$ and $w_e \in X_0$.

As the main loop of Algorithm 1 (Lines 3–13) starts iterating with $t = p$, there is no depleted vertex on $(t, t+1)$ for $t < p$. Hence $t_k \geq p$ and given that $w_b \in C_p \setminus C_{p+1}$ it follows that $w_b \in C^*_p \setminus C^*_{p+1}$. In other words, $w_b \in C^*_{1,p} \setminus C^*_{p+1}$ and so $w_b \in X_k$. Similarly, $w_e \in C_q \setminus C_{q-1} = C^*_q \setminus C^*_{q-1}$ since $t^* < q$, which implies that $w_e \in C^*_{q,s} \setminus C^*_{q-1}$ and so $w_e \in X_0$.

Now fix some $i \in \{1, \dots, k-1\}$. By construction of the sequence $t_1, \dots, t_k$ the conditions (i)–(vii) of Lemmas 16 are satisfied with $t = t_i$ and $t' = t_{i+1}$. By (ii), there is a vertex $u \in C^*_{t_{i+1}} \cap C^*_{t_i+1}$ that is active on $(t_{i+1}, t_{i+1}+1)$ and depleted on (t_i, t_i+1). This means that $t_{i+1} + 1 \leq d_u \leq t_i$ and $r_u \geq t_i + 1$. From these bounds, some vertex x must have been attached to the path with extremity u at time $t = d_u$ in Line 5 of Algorithm 1 and so, again by the algorithm, must be a member of T with $r_x = d_u < t_i + 1$.

If we can show that $x \in X_i = C^*_{t_{i+1}+1,t_i} \setminus (C^*_{t_{i+1}} \cup C^*_{t_i+1})$, then the claim follows. Since $r_x < t_i + 1$, then x in neither free nor active on (t_i, t_i+1). Hence, by (vi), x is also not free on $(t_{i+1}, t_{i+1}+1)$. Therefore, $t_{i+1} < \ell_x$. Since $r_x < t_i + 1$, then $x \notin C^*_{t_{i+1}} \cup C^*_{t_i+1}$. This means that $x \in X_i$ and the claim is proved.

Claim 20. *For all distinct $i, j \in \{0, \dots, k\}$ and every $x \in X_i$ and $y \in X_j$, x and y are in distinct components of $G - S^*$.*

From the definition of S^* and by Observation 10, S^* separates X_i and X_j in $G^* = G[V \setminus R]$ and we now want to prove that S^* also separates X_i and X_j in G. In other words, we want to prove that the vertices of R do not connect vertices from distinct X_i's. To this purpose, it suffices to show that for every $z \in R$ there is $i \in \{0, \dots, k\}$ such that $t_{i+1}+1 \leq \ell_z \leq r_z \leq t_i$, where we assume that $t_0 = s$ and $t_{k+1} = 0$.

Suppose, towards a contradiction, that there is some $z \in R$ and some $i \in \{1, \dots, k\}$ with the property that $\ell_z \leq t_i$ and $r_z > t_i$, and assume without loss of generality that i is minimum with respect to these conditions.

We first show that $i > 1$. Indeed, if $i = 1$ then $\ell_z \leq t_1 = t^*$ and $r_z > t^*$. But as z is a member of R, it follows that $z \in (C_{t^*} \cap C_{t^*+1}) \setminus (V(P_1) \cup V(P_2))$. But $(C_{t^*} \cap C_{t^*+1}) \setminus (V(P_1) \cup V(P_2)) = \emptyset$ from the condition at Line 8, given that Algorithm 1 quits at Line 10 at time $t = t^*$, a contradiction.

Therefore $i > 1$. Recall in this case that t_i was constructed from $t = t_{i-1}$ by choosing $t_i < t$ such that for $t' = t_i$ the conditions (i)–(vii) of Lemmas 15 and 16 hold. To finish off the proof of the claim, we will show that $\ell_z \leq t_i \leq t_{i-1} < r_z$, giving the final contradiction since, by the minimality of i, there is no $z \in R$ with $\ell_z \leq t_{i-1}$ and $r_z > t_{i-1}$.

We already know that $\ell_z \leq t_i \leq t_{i-1}$. By (vii), there is $x \in V(G^*)$ such that $a_x = t_i$ and x is active on $(t_i, t_i + 1)$. By (ii) and (v), x is either active or depleted on $(t_{i-1}, t_{i-i} + 1)$. In either case, $r_x \geq t_{i-1} + 1 > t_i$. Given that $a_x = t_i$ (that is, x was attached to some path at the t_i-th iteration of the for loop of Algorithm 1 at Lines 3–13) and $r_x > t_i$, it follows that x was attached to some path at Line 9 of Algorithm 1. Hence, the right bound of x is less than or equal to that of z, which implies $r_z \geq t_{i-1} + 1$ and the claim is proved.

It follows that

$$c_T(G - S^*) \geq k + 1 > |S^*|,$$

where the first inequality follows from Claims 19 and 20 and the second from Lemma 17. □

We are now ready to complete the proof of Theorem 12.

Proof of Theorem 12. We summarize the main steps of our algorithm and analyze its running time.

As argued earlier, we can safely assume that G is connected. As mentioned above, we start by applying the algorithm of Booth and Lueker [4] to construct a clique path $C_1, \dots, C_s$ of G in time $\mathcal{O}(n + m)$. Note that the algorithm outputs a clique path where each clique is inclusion maximal. In particular, $s \leq n$. The algorithm also computes ℓ_v and r_v for each vertex $v \in V(G)$. This allows us to find the vertices w_b and w_e in time $\mathcal{O}(n)$. Also in time $\mathcal{O}(n)$, we construct the list L that contains the set of right bounds of the elements of $T \setminus \{w_b, w_e\}$ in increasing order.

Next, we run Algorithm 1. Notice that the only computations of the algorithm involve, at each iteration t of the for loop, deciding

(a) whether the path under consideration should be extended, and
(b) if (a) holds, then which vertex of G is to be attached to its extremity.

Given that, by the algorithm, a path is extended only if the right bound of its extremity or of some vertex of T is precisely t, computation (a) takes constant time with the list L at hand at each iteration t of the for loop. Hence, computation (a) takes $\mathcal{O}(n)$ time by the end of the algorithm.

Whenever a path is to be extended, we must consider the neighborhood of its extremity. As we only extend one path at a time, by the end of the algorithm the number of vertices that have been considered to be attached is at most $\sum_{v \in V} d_G(v) = 2m$. Hence computation (b) takes $\mathcal{O}(m)$ time by the end of the algorithm. Thus Algorithm 1 takes $\mathcal{O}(n+m)$ time.

If Algorithm 1 finishes at Line 15 and outputs two paths P_1 and P_2, then we are done by Lemma 13. Otherwise, Algorithm 1 finishes at Line 10 so we work backwards through the algorithm and need to describe how to construct the sequence $t_1, \dots, t_k$, the set $S^* = \bigcup_{i=1}^{k} S_i$ and the paths P_1^* and P_2^* in $\mathcal{O}(n + m)$ time.

As we run the algorithm, we record

(a) for each vertex in $P_1^* \cup P_2^*$, its activation and depletion time and
(b) for each $t \in \{p, \dots, q\}$, the set A_t of vertices that are active on $(t, t + 1)$.

Clearly, all these auxiliary computations can be done in time $\mathcal{O}(n)$.

Let $Q = V(P_1^*) \cup V(P_2^*)$ be ordered with respect to their left bounds (break ties arbitrarily). Denote by δ_i the number of vertices that are depleted on (t_i, t_i+1) for $i \in \{1, \dots, k\}$. Scanning Q, starting from its last vertex and working backwards towards its first vertex, we find S_i, t_i and δ_i for each $i \in \{1, \dots, k\}$. The set S_1 together with δ_1 is computed directly in time $\mathcal{O}(n)$. It remains to show how to compute t_{i+1}, S_{i+1} and δ_{i+1} if t_i, S_i and δ_i are given.

If $\delta_i = 0$, the construction stops. Otherwise, for $t = t_i$, we successively consider values of $t' = t - 1, \dots$ and stop as soon as t' satisfies conditions (i)–(vii) of Lemma 16 which, by the lemma, is guaranteed to happen since $\delta_i > 0$.

To verify (vii), we only need to check whether there is $x \in A_{t'}$ such that $a_x = t'$.

To verify (i), we work backwards through Q, starting from its last element towards its first, until we find some vertex $y \in Q$ such that $\ell_y \geq t' + 1$. If such a vertex exists, then (i) holds. Otherwise, (i) fails.

Conditions (ii), (iii) and (v) can be easily verified by considering $A_{t'}$ and A_t.

For (iv), we count the number of vertices that are depleted on $(t', t'+1)$ whose right bounds are at least $t+1$. Then comparing this number with δ_t, we can decide whether (iv) holds. Finally, to check (vi), we consider the free vertices of $C^*_{t'} \cap C^*_{t'+1}$.

To evaluate the total running time, notice that we can trace Q once only, because we can simply consider, for each values of t' and t, the subset of Q from the place we stopped in the preceding step. Hence, the total number of operations involving Q is $\mathcal{O}(n)$. Moreover, for each $x \in A_{t'}$ with $a_x = t'$, we have $C^*_{t'} \cap C^*_{t'+1} \subseteq N_{G^*}[x]$. Hence, considering $C^*_{t'} \cap C^*_{t'+1}$ can be done in time that is proportional to $|N_{G^*}(x)|$. Since each such x is unique for distinct values of t', it follows that the total number of operations is $\mathcal{O}(n+m)$. Hence, all the sets S_i for $i \in \{1,\dots,k\}$ can be computed in $\mathcal{O}(n+m)$ time.

It follows that $S^* = \cup_{i=1}^{k} S_i$ can be constructed in time $\mathcal{O}(n+m)$, as needed. By Lemma 18, S^* is a T-separator with $c_T(G - S^*) - |S^*| > 0$ and hence certifies a no-answer. □

To avoid possible misunderstandings, note that the assumption that the cliques of the clique path of G in the proof of Theorem 12 are maximal is crucial for the running time analysis. But it is also necessary to prove Lemmas 13–18 without this maximality assumption, since the cliques $C^*_1, \dots, C^*_s$ of the graph G^* (obtained from G by the removal of the set of renounced vertices) are not necessarily maximal. In other words, it is essential to start off with an input graph whose clique path consists of maximal cliques but to also prove statements that concern interval graphs whose clique path may contain nonmaximal cliques.

Using Theorem 12, we can now describe, in Corollary 22, how to obtain an algorithm for Cycle Segment Cover and prove Theorem 1 for interval graphs. First, we state an auxiliary folklore observation (see, e.g., [13]).

Observation 21. *Let G be a graph, $T \subseteq V(G)$ and k be a positive integer. Let G' be the graph obtained from G by adding k vertices that are universal in G'. Then $\mathrm{sc}_T(G) \le k$ if and only if $\mathrm{sc}_T(G') \le 0$.*

Proof. Observe that the separators of G' are exactly the subsets S' of vertices containing both some separator S of G and the k introduced universal vertices, which gives $|S'| = |S| + k$. On the other hand, observe that the connected components of $G' \setminus S'$ are the same as the connected components of $G \setminus S$ and that the terminals are also the same in G' and in G. □

Corollary 22. *There is an algorithm that, given an instance (G, T, r) of* Cycle Segment Cover *where G is an interval graph and T is not a clique, in time $\mathcal{O}(n+m)$ returns either a T-cycle-segment cover of size at most r or a T-separator S^* with $c_T(G - S^*) - |S^*| > r$ that certifies a no-answer.*

Proof. If $r = 0$, then an instance $(G, T, 0)$ of Cycle Segment Cover is precisely an instance (G, T) of Terminal Cyclability and the corollary follows by Theorem 12. So we can assume, from now on, that $r \ge 1$.

Let G' be the interval graph obtained[2] from G by adding a set X of r universal vertices to G and consider (G', T) as an instance of Terminal Cyclability. If G' contains a cycle C that covers T and does not intersect X, then C also covers T in G. So we can assume that $C \cap X \neq \emptyset$ which, in turn, implies that $C - X$ consists of a family of at most r mutually disjoint paths that covers T, as required.

Let us now consider the case where G' is not T-cyclable. By Theorem 12, G' contains a T-separator S such that $c_T(G' - S) > |S|$ and, as each vertex of X is universal and S is a separator, $X \subseteq S$. Let $S^* = S \setminus X$. Then S^* is a T-separator in G such that

$$c_T(G - S^*) = c_T(G' - S) > |S| = |S^*| + r,$$

so that rearranging gives us $c_T(G - S^*) - |S^*| > r$ as needed.

To complete the proof, it remains to argue that the algorithm from Theorem 12 with input (G', T) can be adapted to run in time $\mathcal{O}(n+m)$, where $n = |V(G)|$ and $m = |E(G)|$. (The reader might suspect that replicating the proof of Theorem 12 is sufficient for this purpose. It is, however, possible that $|E(G')|$ is not $\mathcal{O}(m)$ so that the size of G' is not linear in the size of G so we need to more carefully analyze Algorithm and the procedure that is used to construct S^*.)

Instead of running Algorithm 1 directly with input (G', T), we rather run it with input (G, T) as long as the if condition of Line 8 in Algorithm is satisfied. If the algorithm returns P_1 and P_2, then G is T-cyclable and there is nothing to prove. So assume that at some iteration of the for loop, it was impossible to attach a new vertex from G to some extremity of the path under consideration, i.e. the if condition of Line 8 is not satisfied. In this case, we augment our algorithm by attaching an arbitrary vertex x of X (not included in either paths so far) to this path. It is clear that this is possible as long as there is a vertex of X not included in either paths. Given the list L at hand from the proof of Theorem 12, the repetition of this procedure is an algorithm that runs in time $\mathcal{O}(n+m)$ that either returns a yes-instance of Cycle Segment Cover or terminates when every vertex of X has been included in one of the paths at the time the if condition of Line 8 in Algorithm 1 is not satisfied.

In the latter case, we must describe how to construct the set S^* in $\mathcal{O}(n+m)$ time. As we run the above procedure, we store in a list L', for each interval $(t, t+1)$, the (size of the) subsets of vertices of X that are active, depleted and free during $(t, t+1)$. This can be done in $\mathcal{O}(n)$ time as we can simply determine the activation and depletion times of

[2] Note that the class of interval graphs is stable under the addition of a universal vertex.

each vertex of X as these are universal. We then proceed just as in the proof of Theorem 12 by constructing the sequence $t_1, \dots, t_k$ and the sets $S_1, \dots, S_k$ and let $S^* = \cup_{i=1}^{k} S_i$. Since, using the list L', we can easily accommodate the set X when verifying conditions (i)–(vii) as we construct the sequence $t_1, \dots, t_k$, this completes the proof. □

3.2. *k*-Cyclability *for interval graphs*

In this subsection we prove Theorem 2 for interval graphs. For this, given an interval graph G which is not complete, we shall determine, in polynomial time, the largest integer k such that G is k-cyclable. Corollary 22 immediately gives the following corollary.

Corollary 23. *The class of interval graphs is cycle-scattering dual (see Definition 3).*

Proof. Let $G = (V, E)$ be an interval graph and let $T \subseteq V$ be a set of terminals. We want to prove that $\mathsf{seg}_T(G) = \max\{\mathsf{sc}_T(G), 0\}$ (see Definition 3). We denote $\mathcal{A}$ the algorithm whose existence is stated in Corollary 22.

From Observation 3, we have $\mathsf{sc}_T(G) \leq \mathsf{seg}_T(G)$. Then, if $\mathsf{seg}_T(G) = 0$, we indeed have $\mathsf{seg}_T(G) = \max\{\mathsf{sc}_T(G), 0\}$. On the other hand, if $\mathsf{seg}_T(G) > 0$, thanks to Observation 3 again, we just need to prove that $\mathsf{sc}_T(G) \geq \mathsf{seg}_T(G)$. To this purpose, simply run Algorithm $\mathcal{A}$ on the instance $(G, T, \mathsf{seg}_T(G) - 1)$. Since, by definition, it is impossible to cover the vertices of T using only $\mathsf{seg}_T(G) - 1$ paths in G, then, from Corollary 22, Algorithm $\mathcal{A}$ provides a set S^* such that $c_T(G - S^*) - |S^*| \geq \mathsf{seg}_T(G)$. This implies that $\mathsf{sc}_T(G) \geq \mathsf{seg}_T(G)$. □

By Lemma 9 and Corollary 23, to solve k-Cyclability on interval graphs, it is sufficient to construct a polynomial algorithm that computes the k-scattering number of G for any $k \leq n - 1$. Afterwards, the only task remaining will consist in finding the largest integer k such that $\mathsf{sc}^k(G) \leq 0$.

We start with the following lemma.

Lemma 24. *Let G be an interval graph, let $C_1, \dots, C_s$ be a clique path of G, where $C_1, \dots, C_s$ are pairwise distinct maximal cliques of G, and let S be a separator of G. Then there exist $t_1, \dots, t_r$ with $1 \leq t_1 < \dots < t_r < s$ such that*

$$S' = \bigcup_{i=1}^{r} (C_{t_i} \cap C_{t_i+1}) \subseteq S \tag{4}$$

is a separator of G and $c(G - S') \geq c(G - S)$.

Proof. Note that since G has a separator S, G is not a complete graph, that is, $s \geq 2$. By Proposition 11, the minimal separators of G are the sets $C_t \cap C_{t+1}$ for $t \in \{1, \dots, s-1\}$. This means that there is $t \in \{1, \dots, s-1\}$ such that $C_t \cap C_{t+1} \subseteq S$. Consider the set of indices $\mathcal{I} = \{t \in \{1, \dots, s-1\} \mid C_t \cap C_{t+1} \subseteq S\}$. We then denote $\mathcal{I} = \{t_1, \dots, t_r\}$ with $t_1 < \dots < t_r$. Let $S' = \bigcup_{i=1}^{r} (C_{t_i} \cap C_{t_i+1}) \subseteq S$.

We show that for any two distinct vertices u and v of $G - S$ that are in distinct components of this graph, u and v are in distinct components of $G - S'$ as well. By Proposition 11, if u and v are in distinct components of $G - S$, then there is $t \in \{1, \dots, s-1\}$ such that $C_t \cap C_{t+1} \subseteq S$ is a minimal $\{u, v\}$-separator in G. By the construction of $\mathcal{I}$, we have $t \in \mathcal{I}$ and, therefore, $C_t \cap C_{t+1} \subseteq S'$. Hence, u and v are in distinct components of $G - S'$.

We obtain that if any two vertices are in distinct components of $G - S$, they are also in distinct components of $G - S'$. We conclude that $c(G - S') \geq c(G - S)$. □

Informally, the above lemma states that, in computing the k-scattering number, one can restrict one's attention to separators of a special form, which we call *canonical separators*.

Definition 4. A *canonical separator* of an interval graph G is a separator of G that is the union of some minimal separators of G. By convention, when G is not connected, we also say that the empty set is a canonical separator.

A direct consequence of Lemma 24 is that, for an interval graph G, $\mathsf{sc}^k(G)$ can be equivalently defined as

$$\mathsf{sc}^k(G) = \max\{c(G - S) - |S| \mid S \text{ is a canonical separator of } G \text{ s.t. } |S| \leq k - 1\}. \tag{5}$$

Our algorithm follows a dynamic programming scheme on the given clique path $C_1, \dots, C_s$ of a noncomplete interval graph G composed by pairwise distinct maximal cliques, which we artificially extend with empty cliques $C_0 = C_{s+1} = \emptyset$, for convenience of notations.

For each integer $i \in \{1, \dots, s+1\}$, we denote $G_i = G[C_{1,i}]$ and $n_i = |C_{1,i}|$. Along our algorithm, we dynamically compute, for each $i \in \{1, \dots, s\}$, a table that we denote D_i, indexed by integers from 0 to $n_i - 1$. For each $j \in \{0, \dots, n_i - 1\}$, D_i is determined by

$$\begin{aligned} D_i(j) = \max\{c(G_i - S) - |S| \mid\ & S \text{ is a canonical separator of } G_{i+1} \text{ s.t.} \\ & |S| \leq j \text{ and } C_i \cap C_{i+1} \subseteq S\} \end{aligned}$$

whenever G_{i+1} has a canonical separator S of size at most j such that $C_i \cap C_{i+1} \subseteq S$, and $D_i(j) = -\infty$ otherwise. Remember that, from Proposition 11, the minimal separators of G_s are the sets $C_t \cap C_{t+1}$ for $t \in \{1, \dots, s-1\}$. Since $C_s \cap C_{s+1} = \emptyset$, it follows that the minimal separators of G_s and G_{s+1} are the same and so are their canonical separators (see Definition 4). Moreover, note that $G_s = G[C_{1,s}] = G$. Therefore, by taking $i = s$ in the definition of $D_i(j)$ above, we obtain

$$D_s(j) = \max\{c(G - S) - |S| \mid S \text{ is a canonical separator of } G \text{ s.t. } |S| \leq j\}$$

so that, from Eq. (5), $D_s(j) = \mathrm{sc}^{j+1}(G)$, that is, D_s is the table of values of the k-scattering numbers for $k \in \{1, \dots, n\}$.

The dynamic programming scheme we use is initialized by filling the table D_1. Since G is not a complete interval graph, we have that $s \geq 2$ and G_2 has a unique canonical separator $C_1 \cap C_2$. Hence, we set $D_1(j) = -\infty$ if $0 \leq j < |C_1 \cap C_2|$ and $D_1(j) = 1 - |C_1 \cap C_2|$ if $|C_1 \cap C_2| \leq j \leq n_1 - 1$. We shall then use the recursion formula given by Lemma 25 in order to compute D_s. For every pair of indices $\alpha, \beta \in \{1, \dots, s\}$ such that $\alpha \leq \beta$ define

$$c_{\alpha,\beta} = c(G[C_{\alpha,\beta} \setminus (C_{\alpha-1} \cup C_{\beta+1})]),$$

and set $\delta_i = |C_i \cap C_{i+1}|$ for $i \in \{1, \dots, s-1\}$ and $\delta_s = +\infty$.

Lemma 25. *Let G be a noncomplete interval graph and $C_1, \dots, C_s$ a clique path of G composed by pairwise distinct maximal cliques together with empty cliques C_0 and C_{s+1}. For every $i \geq 2$ and every $k \in \{0, \dots, n_i - 1\}$,*

(i) if $k < |C_i \cap C_{i+1}|$ then $D_i(k) = -\infty$, and
(ii) if $k \geq |C_i \cap C_{i+1}|$, then

$$D_i(k) = \max_{1 \leq j \leq i-1} \{c_{1,i} - \delta_i,\ c_{j+1,i} - |(C_i \cap C_{i+1}) \setminus C_j| + D_j(k - |(C_i \cap C_{i+1}) \setminus C_j|)\}.$$

Proof. Item (i) is given by the definition of $D_i(k)$ so it only remains to prove item (ii). If we can show that both

$$D_i(k) \geq \max_{1 \leq j \leq i-1} \{c_{1,i} - \delta_i, c_{j+1,i} - |(C_i \cap C_{i+1}) \setminus C_j| + D_j(k - |(C_i \cap C_{i+1}) \setminus C_j|)\} \tag{6}$$

$$D_i(k) \leq \max_{1 \leq j \leq i-1} \{c_{1,i} - \delta_i, c_{j+1,i} - |(C_i \cap C_{i+1}) \setminus C_j| + D_j(k - |(C_i \cap C_{i+1}) \setminus C_j|)\} \tag{7}$$

hold, then (ii) follows.

We first prove (6). Note that $D_i(k) \geq c_{1,i} - \delta_i$. The inequality is trivial for $i = s$. Otherwise, for $i \leq s-1$, $S = C_i \cap C_{i+1}$ is a canonical separator of G_{i+1} and $c(G_i - S) = c_{1,i}$. It remains to show that for all j such that $1 \leq j \leq i-1$, we have

$$D_i(k) \geq c_{j+1,i} - |(C_i \cap C_{i+1}) \setminus C_j| + D_j(k - |(C_i \cap C_{i+1}) \setminus C_j|). \tag{8}$$

Let $j \in \{1, \dots, i-1\}$. If $D_j(k - |(C_i \cap C_{i+1}) \setminus C_j|) = -\infty$, then (8) is trivial. Assume that $D_j(k - |(C_i \cap C_{i+1}) \setminus C_j|) \neq -\infty$. From the definition of D_j, it follows that there exists a canonical separator S of G_{j+1} such that

(a) $C_j \cap C_{j+1} \subseteq S$
(b) $|S| \leq k - |(C_i \cap C_{i+1}) \setminus C_j|$ and
(c) $c(G_j - S) - |S| = D_j(k - |(C_i \cap C_{i+1}) \setminus C_j|)$.

Take $S' = S \cup ((C_i \cap C_{i+1}) \setminus C_j)$. Since $j \leq i-1$, we have $(C_i \cap C_{i+1}) \cap S \subseteq C_j$, which implies that $|S'| = |S| + |(C_i \cap C_{i+1}) \setminus C_j|$. For the same reason, $(C_i \cap C_{i+1}) \cap C_j = (C_i \cap C_{i+1}) \cap (C_j \cap C_{j+1})$ which, together with (a), implies that $C_i \cap C_{i+1} \subseteq S'$. Therefore, since S is canonical and, from (b), $|S'| = |S| + |(C_i \cap C_{i+1}) \setminus C_j| \leq k$, it follows that S' is canonical and so $D_i(k) \geq c(G_i - S') - |S'|$. On the other hand, by construction of S', $c(G_i - S') = c_{j+1,i} + c(G_j - S)$. Combining these observations with (b) and (c) gives us

$$\begin{aligned} D_i(k) &\geq c(G_i - S') - |S'| \\ &= c_{j+1,i} + c(G_j - S) - |S'| \\ &= c_{j+1,i} - |(C_i \cap C_{i+1}) \setminus C_j| + c(G_j - S) - |S| \\ &= c_{j+1,i} - |(C_i \cap C_{i+1}) \setminus C_j| + D_j(k - |(C_i \cap C_{i+1}) \setminus C_j|) \end{aligned}$$

and (6) is proved.

We now prove (7). By definition of D_i, we may choose a canonical separator S of size at most k such that $C_i \cap C_{i+1} \subseteq S$ and $c(G_i - S) - |S| = D_i(k)$. To prove (7), we assume that $c(G_i - S) - |S| > c_{1,i} - \delta_i$ and shall show that there exists j such that $1 \leq j \leq i-1$ and

$$D_i(k) \leq c_{j+1,i} - |(C_i \cap C_{i+1}) \setminus C_j| + D_j(k - |(C_i \cap C_{i+1}) \setminus C_j|).$$

We want to prove that there exists j with $j < i$ such that $C_j \cap C_{j+1} \subseteq S$. Suppose that $i = s$. Note that $C_s \cap C_{s+1} = \emptyset$ is not a separator of $G_{s+1} = G$ unless G is disconnected and $\emptyset$ is its minimal separator. In this case, there exists j with $1 \leq j < s$ such that $C_j \cap C_{j+1} = \emptyset \subseteq S$. If $i < s$, then $\delta_i = |C_i \cap C_{i+1}|$. Given that $c(G_i - S) - |S| > c_{1,i} - |C_i \cap C_{i+1}|$, since S

is a canonical separator, we obtain that there is some $j < i$ such that $C_j \cap C_{j+1} \subseteq S$. We conclude that in both cases, there exists j with $j < i$ such that $C_j \cap C_{j+1} \subseteq S$ and we chose the maximum value of j with this property.

Now define $S^* = S \setminus ((C_i \cap C_{i+1}) \setminus C_j)$ so that $|S| = |S^*| + |(C_i \cap C_{i+1}) \setminus C_j|$. By our choice of S and j, we have that

$$D_i(k) = c(G_i - S) - |S| = c_{j+1,i} + c(G_j - S) - |S|,$$

and, by the choice of j, $c(G_j - S) = c(G_j - S^*)$ so that

$$D_i(k) = c_{j+1,i} - |(C_i \cap C_{i+1}) \setminus C_j| + c(G_j - S^*) - |S^*| \tag{9}$$

On the other hand, since S is a canonical separator and $C_j \cap C_{j+1} \subseteq S^*$, it follows that S^* is a canonical separator of size $|S| - |(C_i \cap C_{i+1}) \setminus C_j| \le k - |(C_i \cap C_{i+1}) \setminus C_j|$. Hence,

$$c(G_j - S^*) - |S^*| \le D_j(k - |(C_i \cap C_{i+1}) \setminus C_j|),$$

which, combined with (9), gives us

$$D_i(k) \le c_{j+1,i} - |(C_i \cap C_{i+1}) \setminus C_j| + D_j(k - |(C_i \cap C_{i+1}) \setminus C_j|)$$

and (7) is proved. This completes the proof of the lemma. □

We are now ready to state the main result of this subsection that implies Theorem 2 for interval graphs.

Theorem 26. *For a noncomplete interval graph G, one can solve k-*Cyclability *and compute the k-scattering number $\mathrm{sc}^k(G)$ for all $k \in \{1, \dots, n\}$ in time $\mathcal{O}(n^3)$.*

Proof. Given a noncomplete interval graph G, we compute a clique path $C_1, \dots, C_s$ using the algorithm of Booth and Lueker [4]. Then we apply our dynamic programming algorithm to compute the tables D_i for $i \in \{1, \dots, s\}$. The correctness of the construction of D_1 is straightforward and Lemma 25 ensures that the computation of the subsequent tables is correct as well.

From D_s, we compute the k-scattering numbers $\mathrm{sc}^k(G)$ for all $k \in \{1, \dots, n\}$. Then we find the largest $k \in \{1, \dots, n\}$ such that $\mathrm{sc}^k(G) \le 0$, and by Lemma 9 and Corollary 23, this gives us the maximum k such that G is k-cyclable.

To evaluate the running time, recall that the algorithm of Booth and Lueker [4] takes $\mathcal{O}(n + m)$ time. The clique path of G can be stored as a list in which each cell C_j contains the list L_j of vertices whose left bound is C_j, sorted by increasing right bound. This sorting can be done in $O(n)$ time for the whole clique path.

Prior to the dynamic programming algorithm itself, we precompute all the coefficients $c_{\alpha,\beta}$, which are needed for the recursion formula of Lemma 25. This takes $O(n^3)$ time as there are $O(n^2)$ of them and the number of connected components in an interval graph can be computed in $O(n)$ time.

For the dynamic programming part, it is easy to see that as i increases one can maintain the list of vertices in $C_i \cap C_{i+1}$. For each fixed i, it is possible to determine the quantities $|(C_i \cap C_{i+1}) \setminus C_j|$ for all the values of j in $\mathcal{O}(n)$ total time. To see this, first sort the vertices of $C_i \cap C_{i+1}$ by decreasing left bound as primary key and increasing right bound as secondary key, which takes $O(n)$ time. Then, scan the clique path of G, using scanning index j, starting from clique C_i in backward direction (that is, by decreasing j). During this scan, when clique C_j is considered, go through the list L_j of vertices whose left bound is C_j and for each vertex in $C_i \cap C_{i+1}$ encountered in L_j, increment the counter for the vertices in $|(C_i \cap C_{i+1}) \setminus C_{j-1}|$. This takes time proportional to $|L_j|$, as both L_j and the vertices of $C_i \cap C_{i+1}$ having left bound C_j are sorted by increasing right bound. Therefore, the whole scan takes $O(n)$ time. Then, the maximum in the recursion formula takes $O(n)$ time to be computed for each $D_i(k)$ and as there are $O(n^2)$ couples i, k to be considered, this gives a running time of $O(n^3)$ for the whole algorithm. □

4. Bipartite permutation graphs

In this section we prove Theorems 1 and 2 for bipartite permutation graphs.

Definition 5. Let $G = (V_1, V_2, E)$ a bipartite graph, and let $\sigma_1 = \langle u_1, \dots, u_p \rangle$ and $\sigma_2 = \langle v_1, \dots, v_q \rangle$ be orderings of V_1 and V_2 respectively. It is said that (σ_1, σ_2) is a *strong ordering* of G if for every $1 \le i < i' \le p$ and $1 \le j' < j \le q$, if $u_i v_j, u_{i'} v_{j'} \in E(G)$, then $u_i v_{j'}, u_{i'} v_j \in E(G)$.

We use the following crucial property proved by Spinrad, Brandstädt and Stewart [35].

Proposition 27 ([35]). *A bipartite graph G is a bipartite permutation graph if and only if it has a strong ordering.*

Spinrad, Brandstädt and Stewart also proved in [35] that it can be decided in linear time whether a bipartite graph is a permutation graph and that, in this case, a strong ordering of the graph can be obtained in linear time as well. Throughout this section we assume that all considered bipartite permutation graphs are given together with their strong orderings. We also assume that $V_1 = \{u_1, \dots, u_p\}$ and $V_2 = \{v_1, \dots, v_q\}$ and use the default notation $\sigma_1 = \langle u_1, \dots, u_p \rangle$ and $\sigma_2 = \langle v_1, \dots, v_q \rangle$ where (σ_1, σ_2) is a strong ordering.

A strong ordering of a bipartite permutation graph has the following useful property.

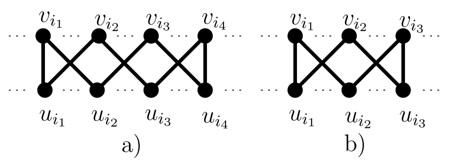

Fig. 2. Regular cycles.

Lemma 28 ([35]). *Let (σ_1, σ_2) be a strong ordering of a connected bipartite permutation graph $G = (V_1, V_2, E)$. Then for every $i \in \{1, 2\}$ and every $v \in V_i$, the vertices of $N_G(v)$ occur consecutively in σ_{3-i}.*

Proposition 27 implies that Hamiltonian paths and cycles have a simple structure in a bipartite permutation graph (if they exist).

Proposition 29 ([35]). *Let $G = (V_1, V_2, E)$ be a connected bipartite permutation graph with a strong ordering (σ_1, σ_2).*

- *If G has a Hamiltonian cycle, then $p = q$ and*
  - *if p is even, then $C = u_1v_2u_3v_4 \cdots u_{p-1}v_pu_pv_{p-1} \cdots u_2v_1u_1$ is a Hamiltonian cycle,*
  - *if p is odd, then $C = u_1v_2u_3 \cdots v_{p-1}u_pv_pu_{p-1}v_{p-2} \cdots u_2v_1u_1$ is a Hamiltonian cycle.*
- *If G has a Hamiltonian path, then there is a Hamiltonian path P such that the path ordering with respect to P of V_1 and V_2 is σ_1 and σ_2 respectively.*

4.1. Certifying algorithms for Terminal Cyclability *and* Cycle Segment Cover

In this subsection, we prove Theorem 1 for bipartite permutation graphs. Let us sketch the main steps of our argument. Firstly, we shall consider Terminal Cyclability and Cycle Segment Cover for $r \geq 1$ separately. The main reason for this is that the auxiliary structural results used in our algorithms are slightly different when we aim to cover the terminals by a cycle and when we use a family of paths.

Recall that in Terminal Cyclability, we are given a bipartite permutation graph $G = (V_1, V_2, E)$ and a subset $T \subseteq V(G)$ and must decide whether there is a cycle that covers T. To be able to decide in $O(n+m)$ time whether such a cycle exists and, if the answer is no, to produce a separator S of G such that $c_T(G - S) > |S|$ that certifies a no-answer, we introduce the notion of *safe pairs*. Essentially, a pair (s, t) of integers is safe if there exists a cycle for the instance $(G_{s,t}, T_{s,t})$, where $G_{s,t} = G[\{u_1, \dots, u_s\} \cup \{v_1, \dots, v_t\}]$ and $T_{s,t} = T \cap V(G_{s,t})$, and (s, t) will be called a *maximal* safe pair if there exists no safe pair (s', t') for which either $s' > s$ and $t' \geq t$ or $t' > t$ and $s' \geq s$. We show that to decide whether T is cyclable, it suffices to restrict one's attention to maximal safe pairs. Our algorithm tries to find a maximal safe pair (s, t) such that $T = T_{s,t}$ and if it fails to discover such a pair, then a maximal safe pair allows us to find a separator S mentioned above.

For Cycle Segment Cover for $r \geq 1$, we will use the same approach but some technical difficulties which are not present in the case of Terminal Cyclability must, however, be overcome.

It would have been ultimately convenient to find a solution to Cycle Segment Cover that uses our solution to Terminal Cyclability as a blackbox, thereby avoiding any kind of repetitive argumentation. However, one cannot hope to adopt the same approach as in the case of interval graphs, because the class of bipartite permutation graphs is not closed under the addition of a universal vertex[3] and so an analogue of Observation 21 for bipartite permutation graphs cannot be applied.

4.1.1. Terminal Cyclability *for bipartite permutation graphs*

Throughout this subsection, let $G = (V_1, V_2, E)$ be a bipartite permutation graph with a strong ordering (σ_1, σ_2), and let $T \subseteq V(G)$ be distinct from a clique. In this subsection, we also assume that G is connected unless it is explicitly stated to be otherwise.

Definition 6. A cycle C of G that covers T is said to be *regular* if there are indices $1 \leq i_1 < \cdots < i_k \leq p$ and $1 \leq j_1 < \cdots < j_k \leq q$ such that

(i) if k is even, then $C = u_{i_1}v_{j_2}u_{i_3}v_{j_4} \cdots u_{i_{k-1}}v_{j_k}u_{i_k}v_{j_{k-1}} \cdots u_{i_2}v_{j_1}u_{i_1}$ (see Fig. 2(a)) and
(ii) if k is odd, then $C = u_{i_1}v_{j_2}u_{i_3} \cdots v_{j_{k-1}}u_{i_k}v_{j_k}u_{i_{k-1}}v_{j_{k-2}} \cdots u_{i_2}v_{j_1}u_{i_1}$ (see Fig. 2(b)).

Similarly, a pair $(\pi_1 = \langle i_1, \dots, i_k\rangle, \pi_2 = \langle j_1, \dots, j_k\rangle)$ of sequences of integers such that $1 \leq i_1 < \cdots < i_k \leq p$ and $1 \leq j_1 < \cdots < j_k \leq q$ is said to form a *regular T-cover* if C defined in (i) or (ii) (depending on the parity of k) is a cycle that contains all the vertices of T.

[3] In fact, the class of bipartite permutation graphs is not closed under the addition of a vertex universal with respect to one side of the bipartition, i.e., for V_1 or V_2.

Lemma 30. *If G has a cycle containing the vertices of T, then there is a regular cycle C that covers T or, equivalently, there is a pair of sequences (π_1, π_2) that is a regular T-cover.*

Proof. Let C be a cycle containing the vertices of T and let $U = V(C)$. Clearly, C is a Hamiltonian cycle of $G[U]$. Let $\{u_{i_1}, \ldots, u_{i_k}\} = U \cap V_1$, where $1 \le i_1 < \cdots < i_k \le p$. Let also $\{v_{j_1}, \ldots, v_{j_k}\} = U \cap V_2$ for $1 \le j_1 < \cdots < j_k \le q$. By Proposition 29, $G[U]$ has the Hamiltonian cycle C' such that either $C' = u_{i_1}v_{j_2}u_{i_3}v_{j_4}\cdots u_{i_{k-1}}v_{j_k}u_{i_k}v_{j_{k-1}}\cdots u_{i_2}v_{j_1}u_{i_1}$ or $C' = u_{i_1}v_{j_2}u_{i_3}\cdots v_{j_{k-1}}u_{i_k}v_{j_k}u_{i_{k-1}}v_{j_{k-2}}\cdots u_{i_2}v_{j_1}u_{i_1}$ if k is either even or odd, respectively. By Definition 6, C' is a regular cycle covering T. $\square$

Lemma 31. *Let $k \ge 2$. Then $(\pi_1 = \langle i_1, \ldots, i_k\rangle, \pi_2 = \langle j_1, \ldots, j_k\rangle)$ with $1 \le i_1 < \cdots < i_k \le p$ and $1 \le j_1 < \cdots < j_k \le q$ forms a regular T-cover if and only if $T \subseteq \{u_{i_1}, \ldots, u_{i_k}\} \cup \{v_{j_1}, \ldots, v_{j_k}\}$ and, for each $h \in \{2, \ldots, k\}$, $u_{i_{h-1}}v_{j_{h-1}}, u_{i_{h-1}}v_{j_h}, u_{i_h}v_{j_{h-1}}, u_{i_h}v_{j_h} \in E(G)$.*

Proof. Suppose that $(\pi_1 = \langle i_1, \ldots, i_k\rangle, \pi_2 = \langle j_1, \ldots, j_k\rangle)$ forms a regular T-cover. By definition, $T \subseteq \{u_{i_1}, \ldots, u_{i_k}\} \cup \{v_{j_1}, \ldots, v_{j_k}\}$. Suppose that k is even. Then $C = u_{i_1}v_{j_2}u_{i_3}v_{j_4}\cdots u_{i_{k-1}}v_{j_k}u_{i_k}v_{j_{k-1}}\cdots u_{i_2}v_{j_1}u_{i_1}$ is a cycle. Therefore, for each $h \in \{2, \ldots, k\}$, $u_{i_{h-1}}v_{j_h}, u_{i_h}v_{j_{h-1}} \in E(G)$. Similarly, if k is odd, then $C = u_{i_1}v_{j_2}u_{i_3}\cdots v_{j_{k-1}}u_{i_k}v_{j_k}u_{i_{k-1}}v_{j_{k-2}}\cdots u_{i_2}v_{j_1}u_{i_1}$ is a cycle and we again have that $u_{i_{h-1}}v_{j_h}, u_{i_h}v_{j_{h-1}} \in E(G)$ for $h \in \{2, \ldots, k\}$. By Definition 5, $u_{i_{h-1}}v_{j_{h-1}}, u_{i_h}v_{j_h} \in E(G)$ for every $h \in \{2, \ldots, k\}$, and we obtain that $u_{i_{h-1}}v_{j_{h-1}}, u_{i_{h-1}}v_{j_h}, u_{i_h}v_{j_{h-1}}, u_{i_h}v_{j_h} \in E(G)$.

For the opposite direction, assume that $T \subseteq \{u_{i_1}, \ldots, u_{i_k}\} \cup \{v_{j_1}, \ldots, v_{j_k}\}$ and for each $h \in \{2, \ldots, k\}$, it holds that $u_{i_{h-1}}v_{j_{h-1}}, u_{i_{h-1}}v_{j_h}, u_{i_h}v_{j_{h-1}}, u_{i_h}v_{j_h} \in E(G)$. We have that if k is even, then $C = u_{i_1}v_{j_2}u_{i_3}v_{j_4}\cdots u_{i_{k-1}}v_{j_k}u_{i_k}v_{j_{k-1}}\cdots u_{i_2}v_{j_1}u_{i_1}$ is a cycle, and if k is odd, then $C = u_{i_1}v_{j_2}u_{i_3}\cdots v_{j_{k-1}}u_{i_k}v_{j_k}u_{i_{k-1}}v_{j_{k-2}}\cdots u_{i_2}v_{j_1}u_{i_1}$ is a cycle. By Definition 6, this means that (π_1, π_2) is a regular T-cover. $\square$

Definition 7. Let $s \in \{1, \ldots, p\}$ and $t \in \{1, \ldots, q\}$, and let $T_{s,t} = T \cap \big(\{u_1, \ldots, u_s\} \cup \{v_1, \ldots, v_t\}\big)$. The pair (s, t) is called *safe* if $u_sv_t \in E(G)$ and there is a pair $\pi_{s,t} = (\pi_1 = \langle i_1, \ldots, i_k\rangle, \pi_2 = \langle j_1, \ldots, j_k\rangle)$ with $1 \le i_1 < \cdots < i_k = s$ and $1 \le j_1 < \cdots < j_k = t$ such that either $k = 1$ and $T_{s,t} \subseteq \{u_s, v_t\} = \{u_{i_1}, v_{j_1}\}$ or $k \ge 2$ and $\pi_{s,t}$ is a regular $T_{s,t}$-cover. A safe pair (s, t) is *maximal* if there exists no safe pair (s', t') for which either $s' > s$ and $t' \ge t$ or $t' > t$ and $s' \ge s$.

If the algorithm fails to find a safe pair (s, t) such that $T_{s,t} = T$, then the algorithm should produce a T-separator S such that $c_T(G - S) > |S|$. We show that there is such a separator of a special structure in the next lemma.

Lemma 32. *Let (s, t) be a maximal safe pair such that $T_{s,t} \neq T$. Then there is a nonnegative integer r such that either $S = \{u_s, u_{s-1}, \ldots, u_{s-r}\}$ or $S = \{v_t, v_{t-1}, \ldots, v_{t-r}\}$ is a T-separator such that $c_T(G - S) > |S|$. Moreover, given $\pi_{s,t}$, the set S can be found in time $\mathcal{O}(n)$.*

Proof. Since (s, t) is safe, by definition $u_sv_t \in E(G)$ and there is a pair $\pi_{s,t} = (\pi_1 = \langle i_1, \ldots, i_k\rangle, \pi_2 = \langle j_1, \ldots, j_k\rangle)$ with $1 \le i_1 < \cdots < i_k = s$ and $1 \le j_1 < \cdots < j_k = t$ such that either $k = 1$ and $T_{s,t} \subseteq \{u_s, v_t\} = \{u_{i_1}, v_{j_1}\}$ or $k \ge 2$ and $\pi_{s,t}$ is a regular $T_{s,t}$-cover.

If u_s has a neighbor in $\{v_{t+1}, \ldots, v_q\}$ and v_t has a neighbor in $\{u_{s+1}, \ldots, u_p\}$, then, from Definition 5 and Lemma 28, we have that $u_sv_{t+1}, u_{s+1}v_t, u_{s+1}v_{t+1} \in E(G)$. But, by Lemma 31, this implies $(s + 1, t + 1)$ is a safe pair, contradicting the maximality of (s, t). Hence either u_s has no neighbor in $\{v_{t+1}, \ldots, v_q\}$ or v_t has no neighbor in $\{u_{s+1}, \ldots, u_p\}$. By symmetry, we can assume without loss of generality that v_t is adjacent to neither of $u_{s+1}, \ldots, u_p$. By the strong ordering property, this implies that no vertex of $\{v_1, \ldots, v_t\}$ is adjacent to some vertex of $\{u_{s+1}, \ldots, u_p\}$. But, since G is connected, there must exists some vertex $a \in \{u_1, \ldots, u_s\}$ and some vertex $b \in \{v_{t+1}, \ldots, v_q\}$ such that $ab \in E(G)$. Since $u_sv_t \in E(G)$, Definition 5 and Lemma 28 imply that u_s is adjacent to b and hence to v_{t+1}.

Our aim is to show that there is $0 \le r \le s - 1$ such that $S = \{u_s, u_{s-1}, \ldots, u_{s-r}\}$ is a T-separator and $c_T(G - S) > |S|$. Towards this aim, we formally define $j_{k+1} = t + 1$ and show the following claim.

Claim 33. *There must exist an integer $\ell \ge 0$ such that*

(i) $u_{s-\ell} = u_{i_{k-\ell}}$ and,
(ii) either $u_{s-\ell} = u_1$ or $v_{j_{k-\ell+1}}$ is not adjacent to $u_{s-\ell-1}$.

Proof of Claim 33. Let $\ell \in \{0, \ldots, k - 1\}$ be maximum such that $u_{s-\ell} = u_{i_{k-\ell}}$. Observe that ℓ is well-defined (since $s = i_k$) and that $u_{i_{k-\ell}}, \ldots, u_{i_k}$ appear consecutively in σ_1. Suppose, towards a contradiction, that (ii) fails for every $h \in \{0, \ldots, \ell\}$, that is, $s - \ell \ge 2$ and $v_{j_{k-h+1}}$ is adjacent to u_{s-h-1} (see Fig. 3). If $\ell = k - 1$ then since $v_{j_{k-\ell+1}}u_{s-\ell-1} \in E(G)$, $u_{s-\ell-1}v_{j_{k-\ell}} \in E(G)$ by Definition 5. Hence $u_{i_{h-1}}v_{j_h}, u_{i_{h-1}}v_{j_{h+1}}, u_{i_h}v_{j_h}, u_{i_h}v_{j_{h+1}} \in E(G)$ for every $h \in \{1, \ldots, k\}$ and hence, by Lemma 31, $(\langle i_0 = s - \ell - 1, \ldots, i_k\rangle, \langle j_1, \ldots, j_{k+1}\rangle)$ is a regular $T_{s,t+1}$-cover, which contradicts the maximality of (s, t).

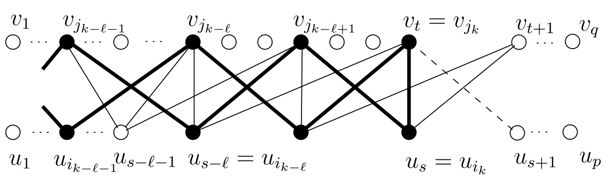

Fig. 3. Configuration leading to a contradiction. Here $\ell = 2$, the edges of $T_{s,t}$-cover are shown by thick lines, and crucial adjacencies are shown by thin lines.

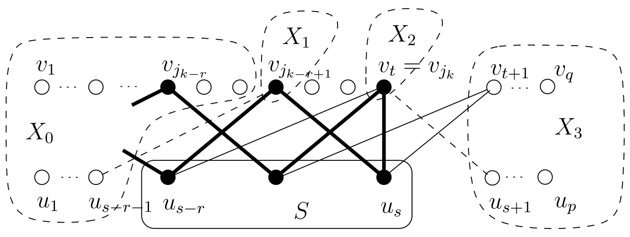

Fig. 4. Construction of S. Here $r = 2$, the edges of $T_{s,t}$-cover are shown by thick lines, and crucial nonadjacencies are shown by straight dashed lines.

Now suppose $\ell < k - 1$. This means that $i_{k-\ell-1} < s - \ell - 1 < s - \ell = i_{k-\ell}$. Since $v_{j_{k-\ell-1}}$ is adjacent to both $u_{s-l} = u_{i_{k-l}}$ and $u_{i_{k-l-1}}$, then, by Lemma 28, $v_{j_{k-\ell-1}}$ is adjacent to $u_{s-\ell-1}$. Let

$$i'_h = \begin{cases} i_{h+1} & \text{if } 0 \le h \le k-\ell-1, \\ s-\ell-1 & \text{if } h = k-\ell-1, \\ i_h & \text{if } k-\ell \le h \le k. \end{cases}$$

for $h \in \{0, \dots, k\}$. By Definition 5, it holds that $u_{i'_{h-1}}v_{j_h}, u_{i'_{h-1}}v_{j_{h+1}}, u_{i'_h}v_{j_h}, u_{i'_h}v_{j_{h+1}} \in E(G)$ for every $h \in \{1, \dots, k\}$. Hence, by Lemma 31, $(\langle i'_0, \dots, i'_k\rangle, \langle j_1, \dots, j_{k+1}\rangle)$ is a regular $T_{s,t+1}$-cover, again contradicting the maximality of (s, t). Hence, (ii) holds for the chosen value of ℓ. Therefore, ℓ exists and the claim is proved. □ □

Let r be the minimum value of $\ell \geq 0$ satisfying the conditions (i) and (ii) of Claim 33. We prove that $S = \{u_s, u_{s-1}, \dots, u_{s-r}\}$ is a T-separator of G such that $c_T(G - S) > |S|$.

Define (see Fig. 4)

$$X_0 = \{u_1, \dots, u_{s-r-1}\} \cup \{v_1, \dots, v_{j_{k-r+1}-1}\},$$
$$X_h = \{v_{j_{k-r+h}}\} \text{ for } 1 \le h \le r,$$
$$X_{r+1} = \{u_{s+1}, \dots, u_p\} \cup \{v_{t+1}, \dots, v_q\}.$$

Observe that $X_0, X_1, \dots, X_{r+1}$ are disjoint subsets of $V(G) \setminus S$. By Claim 33(ii), Definition 5 and Lemma 28, there are no edges between X_0 and X_h for $1 \le h \le r + 1$. Of course, there are also no edges between X_i and X_j for $1 \le i < j \le r$. Since, by assumption, there are no edges between v_t and $\{u_{s+1}, \dots, u_p\}$, there are also no edges between X_h and X_{r+1} for $h \in \{1, \dots, r\}$. The vertices $u_1, \dots, u_{s-r-1}$ have no neighbors among the vertices $v_{j_{k-r+1}}, \dots, v_q$, and the vertices $v_1, \dots, v_{j_{k-r+1}-1}$ are not adjacent to any of the vertices $u_{s+1}, \dots, u_p$. Thus, $N_G(X_0) \subseteq S$. Furthermore, $N_G(X_i) \subseteq S$ for $i \in \{1, \dots, r\}$. Finally, because $u_{s+1}, \dots, u_p$ are not adjacent to either of $v_1, \dots, v_t$, $N_G(X_{r+1}) \subseteq S$. We have that $N_G(X_i) \subseteq S$ for $i \in \{0, \dots, r + 1\}$ and, therefore, every two vertices that are in distinct sets X_j and X_j for $i, j \in \{0, \dots, r + 1\}$ are in distinct components of $G - S$. Thus, if we can show that $T \cap X_\ell \neq \emptyset$ for every $\ell \in \{0, \dots, r + 1\}$, then

$$c_T(G - S) \ge r + 2 > |S| = r + 1$$

and we obtain that $c_T(G - S) > |S|$.

Given that $T_{s,t} \neq T$, we immediately have that $T \cap X_{r+1} \neq \emptyset$.

Now, suppose towards a contradiction that $X_0 \cap T = \emptyset$. Thus,

$$T_{s,t+1} \subseteq T_{s,t} \cup \{v_{t+1}\} \subseteq \{u_{i_{k-r}}, \dots, u_{i_k}\} \cup \{v_{j_{k-r+1}}, \dots, v_{j_{k+1}}\}.$$

If $r = 0$, then $T_{s,t+1} \subseteq \{u_s, v_{t+1}\}$ (given that $j_{k+1} = t+1$ and $i_k = s$) so that $(s, t+1)$ is a safe pair, which contradicts the maximality of (s, t). Let $r \ge 1$. Then for every $h \in \{0, \dots, r-1\}$, we have that $u_{i_{k-h-1}}v_{j_{k-h}}, u_{i_{k-h-1}}v_{j_{k-h+1}}, u_{i_{k-h}}v_{j_{k-h}}, u_{i_{k-h}}v_{j_{k-h+1}} \in E(G)$. Thus, by Lemma 31, the pair $\big(\langle i_{k-r}, \dots, i_k\rangle, \langle j_{k-r+1}, \dots, j_{k+1}\rangle\big)$ is a $T_{s,t+1}$-cover, which again contradicts the maximality of (s, t).

Finally, to prove that $T \cap X_h \neq \emptyset$ for each $h \in \{1, \ldots, r\}$, we suppose for a contradiction that there exists $\ell \in \{1, \ldots, r\}$ such that $v_{j_{k-\ell+1}} \notin T$. By the choice of r, $u_{i_{k-h}} v_{k-h+2} \in E(G)$ for each $h \in \{0, \ldots, \ell-1\}$. Thus, if we define for $h' \in \{1, \ldots, k\}$,

$$j'_{h'} = \begin{cases} j_{h'} & \text{if } 1 \leq h \leq k-\ell, \\ j_{h'+1} & \text{if } k-\ell+1 \leq h \leq k. \end{cases}$$

then the pair $(\pi_1 = \langle i_1, \ldots, i_k \rangle, \langle j'_1, \ldots, j'_k \rangle)$ is a $T_{s,t+1}$-cover by Lemma 31. This final contradiction to the maximality of (s, t) proves the first part of the lemma.

Note that to construct our set S, we only have to find r, that is, the minimum value of ℓ satisfying (i) and (ii) of Claim 33, and this can be easily done in time $\mathcal{O}(n)$ provided $\pi_{s,t}$ is given (the details are left to the reader) and so the "moreover" part of the lemma is immediate. □

Let us emphasize once again that one way of understanding Lemma 32 is that if there is no cycle that covers T in G, then one can find a separator S of G such that $c_T(G - S) > |S|$ by "simply" finding a maximal safe pair of G. That is, to be able to compute S efficiently it suffices to compute a maximal safe pair efficiently. In our next lemma, we provide a characterization of safe pairs that will allow us to identify them efficiently. To be able to give the details of the lemma, we define for a safe pair (s, t), the value $pred(s, t) = \emptyset$ if $T_{s,t} \subseteq \{u_s, v_t\}$, and set $pred(s, t) = (s', t')$, otherwise, where (s', t') is a safe pair such that $(s', t') \neq (s, t)$, $s' \leq s$, $t' \leq t$, $u_{s'} v_t, u_s v_{t'} \in E(G)$ and $T \cap (\{u_{s'+1}, \ldots, u_{s-1}\} \cup \{v_{t'+1}, \ldots, v_{t-1}\}) = \emptyset$. Notice that the choice of (s', t') may be not unique. We show that $pred(s, t)$ can be chosen in a special way.

Lemma 34. *Suppose $s \in \{1, \ldots, p\}$, $t \in \{1, \ldots, q\}$, $u_s v_t \in E(G)$ and $T_{s,t} \setminus \{u_s, v_t\} \neq \emptyset$. Then (s, t) is a safe pair if and only if $s, t \geq 2$, $u_{s-1} v_t, u_s v_{t-1} \in E(G)$ and at least one of the following holds:*

(i) $(s-1, t-1)$ is a safe pair;
(ii) $(s-1, t-1)$ is not safe, $(s, t-1)$ is safe and $v_{t-1} \notin T$;
(iii) $(s-1, t-1)$ is not safe, $(s-1, t)$ is safe and $u_{s-1} \notin T$.

Moreover, if (s, t) is safe, then we can set $pred(s, t) = (s-1, t-1)$, $pred(s, t) = pred(s, t-1)$ and $pred(s, t) = pred(s-1, t)$ in the cases (i), (ii) and (iii) respectively.

Proof. To prove the "only if" part, suppose that (s, t) is a safe pair such that $s \in \{1, \ldots, p\}$, $t \in \{1, \ldots, q\}$, $u_s v_t \in E(G)$ and $T_{s,t} \setminus \{u_s, v_t\} \neq \emptyset$. Since $T_{s,t} \setminus \{u_s, v_t\} \neq \emptyset$, there is a pair $\pi_{s,t} = (\pi_1 = \langle i_1, \ldots, i_k \rangle, \pi_2 = \langle j_1, \ldots, j_k \rangle)$ with $1 \leq i_1 < \cdots < i_k = s$ and $1 \leq j_1 < \cdots < j_k = t$ such that $k \geq 2$ and $\pi_{s,t}$ is a regular $T_{s,t}$-cover. Over all such pairs, let $\pi_{s,t}$ be such that $(i_k - i_{k-1}) + (j_k - j_{k-1})$ is minimum. Clearly, $s, t \geq 2$ and, by Lemma 28, $u_{s-1} v_t, u_s v_{t-1} \in E(G)$. If $i_{k-1} = s-1$ and $j_{k-1} = t-1$, then (i) holds and we can set $pred(s, t) = (s-1, t-1)$.

Suppose that $i_{k-1} < s-1$ or $j_{k-1} < t-1$. Observe that $(s-1, t-1)$ is not a safe pair in this case by the selection of $\pi(s, t)$. Assume that $i_{k-1} < s-1$ and $j_{k-1} < t-1$. By Lemma 28 and Definition 5, $u_{i_{k-1}} v_{t-1}, u_{s-1} v_{j_{k-1}}, u_{s-1} v_{t-1} \in E(G)$. Since (s, t) is a safe pair, $T \cap (\{u_{i_{k-1}+1}, \ldots, u_{i_k-1}\} \cup \{v_{j_{k-1}+1}, \ldots, v_{j_k-1}\}) = \emptyset$. This means that $(s-1, t-1)$ is a safe pair; a contradiction. Hence, either $i_{k-1} = s-1$ or $j_{k-1} = t-1$. Assume that $i_{k-1} = s-1$ and $j_{k-1} < t-1$. Note that $v_{t-1} \notin T$ by Definition 6 for $T_{s,t}$-cover. By Lemma 28, $u_{s-1} v_{t-1}, u_s v_{t-1} \in E(G)$. It follows that $(s, t-1)$ is a safe pair by Lemma 31. Let $(s', t') = pred(s, t-1)$. Suppose that $s' < s-1$. Observe that $u_{t'} v_{s-1} \in E(G)$ by Lemma 28. Then by the definition of a safe pair and (s', t'), we obtain that $(s-1, t-1)$ is a safe pair; a contradiction. We conclude that $s' = s-1$ and we can set $pred(s, t) = (s', t')$, because $v_{t-1} \notin T$. Thus, (ii) is fulfilled and we can set $pred(s, t) = pred(s, t-1)$. The case $i_{k-1} < s-1$ and $j_{k-1} = t-1$ is symmetric and by the same arguments we conclude that (iii) holds and we can set $pred(s, t) = pred(s-1, t)$.

To prove the "if" part, suppose that $s, t \geq 2$, $u_{s-1} v_t, u_s v_{t-1} \in E(G)$ and at least one of (i), (ii) and (iii) holds. If (i) holds, then (s, t) is a safe pair, because $u_{s-1} v_t, u_s v_{t-1} \in E(G)$. Suppose that (i) does not hold, that is, $(s-1, t-1)$ is not safe. Since (ii) and (iii) are symmetric, we can assume without loss of generality that (ii) holds. If $T_{s,t-1} \subseteq \{u_s, v_{t-1}\}$, then $T_{s,t-1} \subseteq \{u_s\}$ because $v_{t-1} \notin T$. Therefore $T_{s,t} \subseteq \{u_s, u_t\}$, but then $T_{s,t} \setminus \{u_s, v_t\} = \emptyset$, a contradiction. Hence, by definition of a safe pair, there is a pair $\pi_{s,t-1} = (\pi_1 = \langle i_1, \ldots, i_k \rangle, \pi_2 = \langle j_1, \ldots, j_k \rangle)$ with $1 \leq i_1 < \cdots < i_k = s$ and $1 \leq j_1 < \cdots < j_k = t-1$ such that $k \geq 2$ and $\pi_{s,t-1}$ is a regular $T_{s,t-1}$-cover.

Suppose that $i_{k-1} < s-1$. By Lemma 28, $u_{s-1} v_{j_{k-1}}, u_{s-1} v_{t-1} \in E(G)$. Since $T \cap \{u_{i_{k-1}+1}, \ldots, u_{s-1}\} = \emptyset$, we obtain that $(s-1, t-1)$ is a safe pair, a contradiction. Hence, $i_{k-1} = s-1$. Since $u_{s-1} v_t, u_s v_t \in E(G)$ and $v_{t-1} \notin T$, we immediately obtain that (s, t) is a safe pair. This completes the proof. □

We are now ready to solve Terminal Cyclability for bipartite permutation graphs.

Theorem 35. *There is an algorithm that, given an instance (G, T) of* Terminal Cyclability *where G is a bipartite permutation graph and T is not a clique, in time $\mathcal{O}(n+m)$ returns either a cycle of G covering T or a T-separator S^* with $c_T(G - S^*) - |S^*| > 0$ that certifies a no-answer.*

Proof. Let $G = (V_1, V_2, E)$ be a bipartite permutation graph. If G has two distinct components containing vertices of T, then G has no cycle covering T and we return $S^* = \emptyset$. Otherwise, the vertices of T are in the same component and we can discard the other components if they exist. Clearly, all this can be done in linear time.

So we can henceforth assume that G is connected. By Proposition 27, G has a strong ordering $(\sigma_1 = \langle u_1,\dots,u_p\rangle, \sigma_2 = \langle v_1,\dots,v_q\rangle)$ that can be constructed in time $\mathcal{O}(n+m)$ [35]. By Lemma 28, the neighborhood of each vertex is a sequence of the consecutive vertices of V_1 or V_2 with respect to σ_1 or σ_2 respectively.

Next, we try to decide whether, for each $s = 1,\dots,p$ and each neighbor v_t of u_s, the pair (s,t) is safe and, if this is the case, compute $pred(s,t)$. To do so, first check whether $T_{s,t} \subseteq \{u_s, v_t\}$. If this is the case, (s,t) is safe and we set $pred(s,t) = \emptyset$. Otherwise, check whether $s,t \geq 2$, $u_{s-1}v_t, u_sv_{t-1} \in E(G)$ and whether one of the conditions (i)–(iii) of Lemma 34 holds. If this is the case, set $pred(s,t)$ as defined in Lemma 34. Thanks to Lemma 34, this determines whether (s,t) is safe or not. Obviously, all this can be done in $\mathcal{O}(m)$ time. For each fixed s, we also compute the maximum t for which (s,t) is a safe pair. Clearly, all this can be done in time $\mathcal{O}(m)$.

Suppose we find a safe pair (s,t) such that $T = T_{s,t}$ (to decide whether $T = T_{s,t}$ in linear time, we use (σ_1,σ_2)). We know, by Lemma 30, that there is a regular cycle C that covers T. Hence, by using the labels $pred(s',t')$ assigned to the safe pairs (s',t'), we can construct the pair $\pi_{s,t} = (\pi_1 = \langle i_1,\dots,i_k\rangle, \pi_2 = \langle j_1,\dots,j_k\rangle)$ with $1 \leq i_1 < \cdots < i_k = s$ and $1 \leq j_1 < \cdots < j_k = t$ that is a regular T-cover. Since T is not a clique, $k \geq 2$ and we therefore obtain a regular T-cover. Clearly, this can be done in time $\mathcal{O}(n)$.

Suppose we do not find a safe pair (s,t) such that $T = T_{s,t}$. We consider the last $s \in \{1,\dots,p\}$ such that there is a safe pair (s,t). Let t be the maximum t such that (s,t) is a safe pair using the stored precomputed value. Then we have that (s,t) is a maximal safe pair such that $T_{s,t} \neq T$. Thanks to Lemma 32, there is a T-separator S^* such that $c_T(G-S^*) > |S^*|$ which we can output in time $\mathcal{O}(n)$.

To complete the proof, observe that the total running time is $\mathcal{O}(n+m)$. □

4.1.2. Cycle Segment Cover *with $r \geq 1$ for bipartite permutation graphs*

In this subsection we construct a certifying algorithm for Cycle Segment Cover on bipartite permutation graphs for the case $r \geq 1$. Throughout this subsection, $G = (V_1, V_2, E)$ is a bipartite permutation graph with a strong ordering (σ_1,σ_2) and $T \subseteq V(G)$. Recall that we assume that $\sigma_1 = \langle u_1,\dots,u_p\rangle$ and $\sigma_2 = \langle v_1,\dots,v_q\rangle$. Notice that G may be disconnected and we do not demand T be distinct from a clique.

Let P be a path in G. We say that P is *straight* if the path order of $V_1 \cap V(P)$ coincides with the order induced by σ_1 and the path order of $V_2 \cap V(P)$ coincides with the order induced by σ_2.

Lemma 36. *For every $X \subseteq V(G)$, it holds that if G has a path P with $V(P) = X$, then there is a straight path P' with $V(P') = X$.*

Proof. Immediate from Proposition 29. □

Using Lemma 36, we assume that all paths that are considered in this subsection are straight.

We now make a few definitions that are analogues of similar definitions for cycles given in the previous subsection. For a path P, denote by $\pi(P) = (\pi_1(P), \pi_2(P))$ a pair of increasing sequences of integers $\pi_1(P) = \langle i_1,\dots,i_k\rangle$ and $\pi_2(P) = \langle j_1,\dots,j_r\rangle$ such that $V_1 \cap V(P_1) = \{u_{i_1},\dots,u_{i_k}\}$ and $V_2 \cap V(P) = \{v_{j_1},\dots,v_{j_r}\}$. Note that one of the sequences is empty if P is trivial, that is, if P only has a single vertex. Fix $s \in \{1,\dots,p\}$ and $t \in \{1,\dots,q\}$, and write $T_{s,t} = T \cap (\{u_1,\dots,u_s\} \cup \{v_1,\dots,v_t\})$. Then a pair (s,t) is said to be 1-*safe* if there is a path P with $V(P) \subseteq \{u_1,\dots,u_s\} \cup \{v_1,\dots,v_t\}$ such that u_s is an end-vertex of P, $u_sv_t \in E(P)$ and $T_{s,t} \subseteq V(P)$. Similarly, (s,t) is said to be 2-*safe* if there is a path P with $V(P) \subseteq \{u_1,\dots,u_s\} \cup \{v_1,\dots,v_t\}$ such that v_t is an end-vertex of P, $u_sv_t \in E(P)$ and $T_{s,t} \subseteq V(P)$. For the sake of brevity, call (s,t) simply *safe* if (s,t) is either 1-safe or 2-safe and call its corresponding path P (s,t)-*safe*. A safe pair (s,t) is maximal if there is no safe pair (s',t') such that either $s < s'$ and $t \leq t'$ or $s \leq s'$ and $t < t'$. Clearly, if there is a safe pair (s,t) such that $T = T_{s,t}$, then we have a path that covers T.

Lemma 37. *Let G be connected and let (s,t) be a maximal safe pair with the corresponding (s,t)-safe path P such that $T_{s,t} \neq T$. Then there is a nonnegative integer ℓ such that either $S = \{u_s, u_{s-1},\dots,u_{s-\ell}\}$ or $S = \{v_t, v_{t-1},\dots,v_{t-\ell}\}$ is a $T_{s,t}$-separator such that*

(i) $c_{T_{s,t}}(G-S) > |S|$,

(ii) $N_G(\{u_{s+1},\dots,u_p\} \cup \{v_{t+1},\dots,v_q\}) \subseteq S$ *and for each component H of $G-S$ containing a vertex of $T \setminus T_{s,t}$,* $V(H) \subseteq \{u_{s+1},\dots,u_p\} \cup \{v_{t+1},\dots,v_q\}$.

Moreover, such a separator S satisfying (i) and (ii) can be found in $\mathcal{O}(s+t)$ time.

Proof. By symmetry, we assume without loss of generality that (s,t) is a 2-safe pair, that is, v_t is an end-vertex of P and u_s is the previous vertex. Let $\pi(P) = (\pi_1(P), \pi_2(P))$ for $\pi_1(P) = \langle i_1,\dots,i_k\rangle$ and $\pi_2(P) = \langle j_1,\dots,j_r\rangle$.

If $N_G(v_t) \cap \{u_{s+1},\dots,u_p\} \neq \emptyset$, then $v_tu_{s+1} \in E(G)$ by the definition of a strong ordering and Lemma 28 and $(s+1,t)$ is a 1-safe pair contradicting the maximality of (s,t). Hence, v_t has no neighbor in $\{u_{s+1},\dots,u_p\}$. Because G is connected and $T_{s,t} \neq T$, we have that $t < q$ and u_s is adjacent to v_{t+1}.

Claim 38. *There is an integer $h \geq 0$ such that*

(a) $u_{s-h} = u_{i_{k-h}}$ *and,*

(b) either $u_{s-h} = u_1$ *or* $v_{j_{r-h}}$ *is not adjacent to* u_{s-h-1}.

Proof of Claim 38. Let $h \in \{0, \dots, k-1\}$ be maximum such that $u_{s-h} = u_{i_{k-h}}$; thus $u_{i_{k-h}}, \dots, u_{i_k}$ are consecutive vertices in σ_1. Suppose the claim is false for all $h' \in \{0, \dots, h\}$, that is, $s-h \geq 2$ and $v_{j_{r-h'}}$ is adjacent to $u_{s-h'-1}$.

Suppose that $h = k-1$. If v_{j_1} is an end-vertex of P, then $r = k+1$ and $v_{j_1} u_{s-k} v_{j_2} u_{s-k+1} \cdots v_{j_r} u_s v_{t+1}$ is a path and $T_{s+1,t} \subseteq T_{s,t} \cup \{v_{t+1}\} \subseteq V(P) \cup \{v_{t+1}\} \subseteq V(P')$. Hence, $(s, t+1)$ is a safe pair, which contradicts the maximality of (s,t). Similarly, if u_{i_1} is an end-vertex of P, then $r = k$ and $u_{s-k} v_{j_1} u_{s-k-1} \cdots v_{j_r} u_s v_{t+1}$ is a path, which again contradicts the maximality of (s,t).

Finally, suppose that $k-1 < h$. This means that $i_{s-h-1} < s-h-1 < s-h = i_{s-h}$. By Lemma 28, $v_{j_{r-h-1}}$ is adjacent to u_{s-h-1}. Let $P' = P[\{u_{i_1}, \dots, u_{i_{k-h-1}}\} \cup \{v_{j_1}, \dots, v_{j_{r-h-1}}\}]$. Then $P' u_{s-h-1} v_{j_{r-h}} \cdots v_{j_r} u_s v_{t+1}$ is a path, contradicting the maximality of (s,t). This completes the proof of the claim. □

We define $\ell \geq 0$ be the minimum value of h for which Claim 38 is fulfilled. We prove that $S = \{u_s, u_{s-1}, \dots, u_{s-\ell}\}$ is a $T_{s,t}$-separator satisfying (i) and (ii).

Let $X_0 = \{u_1, \dots, u_{s-\ell-1}\} \cup \{v_1, \dots, v_{j_{r-\ell-1}}\}$, and $X_h = \{v_{j_{r-\ell+h-1}}\}$ for $h \in \{1, \dots, \ell+1\}$. Clearly, $X_0, \dots, X_{\ell+1}$ are disjoint subsets of $V(G) \setminus S$. Since v_t has no neighbor in $\{u_{s+1}, \dots, u_p\}$, it follows by Definition 5 and Claim 38, that for every two vertices that are in distinct sets X_i and X_j for $i, j \in \{0, \dots, \ell+1\}$, it holds that these vertices are in distinct components of $G-S$. Thus, if we can show that $T_{s,t} \cap X_h \neq \emptyset$ for $h \in \{0, \dots, \ell+1\}$, then it will follow that $c_{T_{s,t}}(G-S) \geq \ell+2 > |S| = \ell+1$ and (i) holds. Notice that $T_{s,t} \cap X_h = T \cap X_h$ for $h \in \{0, \dots, \ell+1\}$. Thus, we can consider T instead of $T_{s,t}$.

To show that $T \cap X_0 \neq \emptyset$, assume that $T \cap X_0 = \emptyset$. Then we can assume that $\ell = k-1$, $r = k$ and $P = u_{s-k-1} v_{j_1} \cdots u_s v_{j_k}$. By the definition of ℓ, there is a path $P' = v_{j_1} u_{s-k-1} \cdots v_{j_k} u_s v_{t+1}$ but this contradicts the maximality of (s,t). Hence, $T \cap X_0 \neq \emptyset$.

To prove that $T \cap X_h \neq \emptyset$ for $h \in \{1, \dots, \ell+1\}$, we have to show that $v_{j_{r-\ell}}, \dots, v_{j_r} \in T$. To obtain a contradiction, assume that there is $h \in \{0, \dots, \ell\}$, such that $v_{j_{r-h}} \notin T$. Let $P' = P[\{u_{i_1}, \dots, u_{i_{k-h}}\} \cup \{v_{j_1}, \dots, v_{j_{r-h-1}}\}]$, that is, we truncate P at $u_{i_{k-h}}$. Let $j_{r+1} = t+1$. Because u_s is adjacent to v_{t+1} and by the definition of ℓ, $u_{s-h'} v_{s-h'+1}$ for $h' \in \{0, \dots, h\}$. Then there is the path $P'' = P v_{j_{r-h-1}} u_{s-h-1} \dots v_{j_r} u_s v_{t+1}$ that contradicts the maximality of (s,t). This completes the proof that $T \cap X_h \neq \emptyset$ for $h \in \{0, \dots, \ell+1\}$.

To show (ii), recall that $N_G(\{u_{s+1}, \dots, u_p\} \cup \{v_{t+1}, \dots, v_q\}) \subseteq V_1$ and by the definition of ℓ, $v_{s-\ell-1}$ is not adjacent to v_{t+1}. Then $N_G(\{u_{s+1}, \dots, u_p\} \cup \{v_{t+1}, \dots, v_q\}) \subseteq S$. Since, $T \setminus T_{s,t} \subseteq \{u_{s+1}, \dots, u_p\} \cup \{v_{t+1}, \dots, v_q\}$, we have that (ii) holds.

To see that S can be found in $\mathcal{O}(s+t)$ time, it is sufficient to observe that to find minimum ℓ satisfying (a) and (b), we traverse P starting from v_t and this can be done in time $\mathcal{O}(s+t)$. □

As in Section 4.1.1, we now state lemmas for efficiently computing safe pairs and safe paths. To be able to state the lemmas concisely, we need some definitions. Let (s,t) be an h-safe pair for some $h \in \{1,2\}$, and let P be its corresponding (s,t)-safe path so that $u_s v_t$ is an end-edge of P. If P has exactly two vertices, then set $pred_h(s,t) = \emptyset$. Otherwise, set $pred_h(s,t) = (s',t')$ where $s' \leq s$, $t' \leq t$ and $u_{s'} v_{t'}$ is the unique edge of P adjacent to $u_s u_t$. These labels allow us to compute P. Note that P is not necessarily unique for (s,t).

Lemma 39. *Let G be connected and let $s \in \{1, \dots, p\}$ and $t \in \{1, \dots, q\}$. Then a pair (s,t) is a 1-safe (respectively, 2-safe) pair if and only if $u_s v_t \in E(G)$ and one of the following is fulfilled:*

(i) $T_{s,t} \subseteq \{u_s, v_t\}$,
(ii) $s \geq 2$ and $(s-1,t)$ is a 2-safe pair (respectively, $t \geq 2$ and $(s,t-1)$ is a 1-safe pair),
(iii) $s \geq 2$, $u_{s-1} \notin T$ and $(s-1,t)$ is a 1-safe pair (respectively, $t \geq 2$, $v_{t-1} \notin T$ and $(s,t-1)$ is a 2-safe pair).

Moreover, if (s,t) is safe, then we can assign $pred_1(s,t) = \emptyset$ in case (i), $pred_1(s,t) = pred_2(s-1,t)$ and $pred_2(s,t) = pred_1(s,t-1)$ in case (ii), and $pred_1(s,t) = pred_1(s-1,t)$ and $pred_2(s,t) = pred_2(s,t-1)$ case (iii).

Proof. By symmetry, it is sufficient to prove the lemma for 1-safe pairs.

For the "only if" part, suppose that (s,t) is 1-safe and let P be its corresponding (s,t)-safe path. Assume that $T_{s,t} \setminus \{u_s, v_t\} \neq \emptyset$. Then $s \geq 2$. Let $(s',t) = pred_1(s,t)$. If $s' = s-1$, then $(s-1,t)$ is a safe pair and (ii) holds. Assume that $s' < s-1$. Since $T_{s,t} \subseteq V(P)$, $u_{s-1} \notin T$. By Lemma 28, $u_{s-1} v_t \in E(G)$. Thus, $(s-1,t)$ is a 1-safe pair (set $P := (P \setminus \{u_s\}) \cup \{u_{s-1}\})$ and (iii) follows.

For the "if" part, suppose $u_s v_t \in E(G)$. If (i) holds and $T_{s,t} \subseteq \{u_s, v_t\}$, then (s,t) is 1-safe and we can let $pred(s,t) = \emptyset$. If (ii) holds, then since $(s-1,t)$ is 2-safe, there is an $(s-1,t)$-safe path P' with end-vertex v_t. Then (s,t) is a 1-safe pair such that Pu_s is its corresponding (s,t)-safe path and we let $pred_1(s,t) = pred_2(s-1,t)$. Finally, suppose that (iii) holds. Let P' be an $(s-1,t)$-safe path with end-vertex u_{s-1} and let v_t be its unique vertex adjacent to u_{s-1}. Let $P = (P' \setminus \{u_{s-1}\}) \cup \{u_s\}$. Then P is a path and since $u_{s-1} \notin T$, $T_{s,t} \subseteq V(P)$. Hence, (s,t) is a 1-safe pair and we can let $pred_1(s,t) = pred_1(s-1,t)$. □

Lemma 40. *Let G be connected, and let $s \in \{1, \dots, p\}$ and $t \in \{1, \dots, q\}$ be such that $(s,t) \neq (p,q)$. Then a safe pair (s,t) is maximal if and only if there is no safe pair (s',t') such that $s' \geq s$, $t' \geq t$, $(s,t) \neq (s',t')$ and either $s' = s$ or $t' = t$.*

Proof. Suppose that (s,t) is maximal. By definition, there is no safe pair (s',t') such that $(s,t) \neq (s',t')$ and either $s' > s$ and $t' \geq t$ or $t' > t$ and $s' \geq s$.

Conversely, suppose that there is no safe pair (s', t') such that $(s, t) \neq (s', t')$ and either $s' = s$ or $t' = t$. To obtain a contradiction assume that (s, t) is not maximal. Then there are $s' \in \{s, \dots, p\}$ and $t' \in \{t, \dots, q\}$ such that $(s', t') \neq (s, t)$ and (s', t') is safe. Assume that this pair is chosen in such a way that $\min\{s' - s, t' - t\}$ is minimum. We claim that either $s = s'$ or $t = t'$. Assume that $s' > s$ and $t' > t$. Suppose that $T_{s',t'} \subseteq \{u_{s'}, v_{t'}\}$. Since G is connected, either $t < q$ and $u_s v_{t+1} \in E(G)$ or $s < p$ and $u_{s+1} v_t \in E(G)$. Hence either $(s, t + 1)$ or $(s + 1, t)$ is safe, a contradiction. Otherwise, by Lemma 39 (ii) and (iii), we have that either $(s' - 1, t')$ or $(s', t' - 1)$ is safe. This contradicts the choice of (s', t'). □

Now we are ready to show the main result of this subsection.

Theorem 41. *There is an algorithm that, given a bipartite permutation graph G and $T \subseteq V(G)$, in time $\mathcal{O}(n + m)$ finds a minimum size family of vertex-disjoint paths $\mathcal{P}$ covering T and, if T cannot be covered by a single path, a T-separator S^* such that $c_T(G - S^*) - |S^*| \geq |\mathcal{P}|$ certifying the optimality of $\mathcal{P}$.*

Proof. We show an algorithm for connected graphs and then explain how to deal with the disconnected case. In a first stage, we describe the algorithm and prove its correctness. In a second and final stage, we analyze its running time. Our algorithm will return a minimum size family $\mathcal{P}$ of vertex-disjoint paths that cover T and a T-separator S^* such that $c_T(G - S^*) - |S^*| \geq |\mathcal{P}|$ if $|\mathcal{P}| \geq 2$ and the algorithm will return $S^* = \emptyset$ if $T = \emptyset$ or if T can be covered by one path. Note that in the latter case S^* is not a T-separator.

By Proposition 27, G has a strong ordering $(\sigma_1 = \langle u_1, \dots, u_p\rangle, \sigma_2 = \langle v_1, \dots, v_q\rangle)$. Of course, if $T = \emptyset$ then $\mathcal{P} = \emptyset$, and if $|T| = 1$ then $\mathcal{P} = \{T\}$ and $S^* = \emptyset$. So we can assume from now on that $|T| \geq 2$. We begin by computing a maximal safe pair (s, t) of G together with its corresponding (s, t)-safe path P. In case $T_{s,t} = T$, we set $\mathcal{P} = \{P\}$ and $S^* = \emptyset$. In case $T_{s,t} \neq T$, we find a $T_{s,t}$-separator $S \subseteq \{u_1, \dots, u_s\} \cup \{v_1, \dots, v_t\}$ such that

(i) $c_{T_{s,t}}(G - S) > |S|$, and
(ii) $N_G(\{u_{s+1}, \dots, u_p\} \cup \{v_{t+1}, \dots, v_q\}) \subseteq S$ and for each component H of $G - S$ containing a vertex of $T \setminus T_{s,t}$, $V(H) \subseteq \{u_{s+1}, \dots, u_p\} \cup \{v_{t+1}, \dots, v_q\}$.

Thanks to Lemma 37, we know that S exists.

We next recursively apply our algorithm to each of the $k \geq 1$ components $H_1, \dots, H_k$ of the graph $H = G \setminus \big(\{u_1, \dots, u_s\} \cup \{v_1, \dots, v_t\}\big)$. That is, for each H_i we take $T_i = T \cap V(H_i) \subsetneq T$ and return a minimum size family $\mathcal{P}_i$ of paths that cover T_i and a certificate S_i such that

$$c_{T_i}(H_i - S_i) - |S_i| \geq |\mathcal{P}_i|. \tag{10}$$

Here we assume inductively the correctness of the algorithm for $H_1, \dots, H_k$. Taking $\mathcal{P} = \{P\} \cup \mathcal{P}_1 \cup \dots \cup \mathcal{P}_k$, we obtain a family of paths that covers T. And taking $S^* = S \cup S_1 \cup \dots \cup S_k$, we infer by (10) and Lemma 37(i) that

$$c_T(G - S^*) - |S^*| \geq |\mathcal{P}|.$$

By Observation 3, this inequality implies that $\mathcal{P}$ has minimum size which completes the proof of correctness.

We now show that the algorithm can be implemented in time $\mathcal{O}(n + m)$.

The strong ordering of G can be constructed in time $\mathcal{O}(n + m)$ [35]. We compute it once and use it throughout the algorithm.

If $|T| \leq 1$, we need constant time to solve the problem. Otherwise, we have to find a maximal safe pair (s, t). For this, we compute 1 and 2-safe pairs together with their corresponding labels $pred_1$ and $pred_2$ by following the strong ordering. To be able to recognize that a pair is maximal, we alternate between V_1 and V_2 in the following way.

We introduce counters a_1, b_1 and a_2, b_2. Initially, $a_1 = b_1 = 1$ and $a_2 = b_2 = 0$. We find safe pairs and for each safe pair (i, j), we label it as 1 or 2-safe (or both) and compute $pred_1(i, j)$ or $pred_2(i, j)$ (or both). Assume that for each $i \in \{1, \dots, a_1 - 1\}$, we already computed the safe pairs (i, j) for all j such that $u_i v_j \in E(G)$. Symmetrically, we assume that for each $j \in \{1, \dots, a_2 - 1\}$, we already computed the safe pairs (i, j) for all i such that $u_i v_j \in E(G)$. We iterate as follows. For each $i = a_1, \dots, b_1$, we find the safe pairs (i, j) and compute the labels for all j such that $u_i v_j \in E(G)$ using Lemma 39. For each i, we also find the maximum index c_i such that (i, c_i) is safe and set $c_i = 0$ otherwise. Next, we compute $c^* = \max\{c_i \mid a_1 \leq i \leq b_1\}$ and define $a_2 = a_2 + 1$ and $b_2 = c^*$. We then pass to V_2 and proceed analogously: For each $j = a_2, \dots, b_2$, we find the safe pairs (i, j) and compute the labels for all i such that $u_i v_j \in E(G)$. We also find the maximum value d_j such that (d_j, j) is safe for $j \in \{a_2, \dots, b_2\}$ and set $d_j = 0$ if there is no such a pair. We compute $d^* = \max\{d_j \mid a_2 \leq j \leq b_2\}$. Then we define $a_1 = a_1 + 1$, $b_1 = d^*$ and again pass to V_1 etc. We stop when we find a safe pair (s, t) such that either $T_{s,t} = T$ or recognize that (s, t) is maximal using Lemma 40. Observe that the values c_i and d_i allow us to verify whether each considered safe pair (i, j) is maximal using the already computed information whenever we finish considering the edges incident to u_i and v_j.

So far, the algorithm runs in time $\mathcal{O}(\sum_{i=1}^{s} d_G(u_i) + \sum_{j=1}^{t} d_G(v_j))$, where (s, t) is some maximal safe pair. By the last claim of Lemma 37, we compute S in time $\mathcal{O}(s + t)$. By Lemma 37, either $S = \{u_s, u_{s-1}, \dots, u_{s-\ell}\}$ or $S = \{v_t, v_{t-1}, \dots, v_{t-\ell}\}$ for some nonnegative integer ℓ. Assume without loss of generality that $S = \{u_s, u_{s-1}, \dots, u_{s-\ell}\}$. By Lemma 37 (ii), $N_G(\{u_{s+1}, \dots, u_p\} \cup \{v_{t+1}, \dots, v_q\}) \subseteq S$. Let $W = \{v_j \mid t + 1 \leq j \leq j \text{ and } v_j u_{s+1} \notin E(G)\}$. By Definition 5 and Lemma 28, the vertices of W are isolated vertices of H. Moreover, $H - W$ is either empty or connected, because G is connected.

As the vertices of W form trivial components of H, the computations take $\mathcal{O}(|W|)$ time. Finally, recursively applying the algorithm for the graph $H - W$, one can verify that the total running time is $\mathcal{O}(n+m)$.

It remains to extend the algorithm for disconnected graphs. If G has a unique component containing the vertices of T, we simply run the algorithm for this component. Assume that G has at least two connected components $G_1, \dots, G_r$ containing the vertices of T. We compute a minimum family $\mathcal{P}_i$ that cover $T_i = T \cap V(G_i)$ and a certificate S_i for $i \in \{1, \dots, r\}$. Then we define $\mathcal{P} = \mathcal{P}_1 \cup \dots \cup \mathcal{P}_r$ and $S^* = S_1 \cup \dots \cup S_r$. Notice that even if $S^* = \emptyset$, then S^* is a T-separator, because G has at least two components containing the vertices of T. Then it is straightforward to verify that $\mathcal{P}$ covers T and S^* is a T-separator certifying optimality. □

4.1.3. *Cycle Segment Cover for bipartite permutation graphs*

Combining Theorems 35 and 41 we obtain the following corollary that proves the claim of Theorem 1 for bipartite permutation graphs.

Corollary 42. *There is an algorithm that, given an instance (G, T, r) of* Cycle Segment Cover *where G is a bipartite permutation graph and T is not a clique, in time $\mathcal{O}(n+m)$ returns either a T-cycle-segment cover of size at most r or a T-separator S^* with $c_T(G - S^*) - |S^*| > r$ that certifies a no-answer.*

Proof. If $r = 0$, then we simply use Theorem 35. Assume that $r \geq 1$. We use the algorithm from Theorem 41. The algorithm in $\mathcal{O}(n+m)$ time finds a minimum size family of vertex-disjoint paths $\mathcal{P}$ covering T. If $r \geq |\mathcal{P}|$, then we return $\mathcal{P}$. Otherwise, $r < |\mathcal{P}|$ and, in particular, this means that T cannot be covered by a single path. Then the algorithm from Theorem 41 returns a T-separator S^* such that $c_T(G - S^*) - |S^*| \geq |\mathcal{P}| > r$ and we return S^*. □

4.2. *k*-Cyclability *for bipartite permutation graphs*

Theorem 35 allows us to solve k-Cyclability for bipartite permutation graphs. We just need one auxiliary lemma.

Lemma 43. *Let $G = (V_1, V_2, E)$ be a connected bipartite permutation graph with at least three vertices and let k be a positive integer. Then G is k-cyclable if and only if for every $i = 1, 2$ and every set $T \subseteq V_i$ of $\min\{|V_i|, k\}$ consecutive vertices with respect to σ_i, T is cyclable.*

Proof. By definition, if G is k-cyclable, then for every $i = 1, 2$ and every set $T \subseteq V_i$ of $\min\{|V_i|, k\}$ consecutive vertices with respect to σ_i, T is cyclable.

To show the opposite implication, assume that G is not k-cyclable. We must show that for some $i = 1, 2$ there exists a set $T \subseteq V_i$ of $\min\{|V_i|, k\}$ consecutive vertices with respect to σ_i such that T is not cyclable.

Since G is not k-cyclable, we can find a subset $T \subseteq V(G)$ of size at most k that is not cyclable. Let us argue that T can be assumed to not be a clique. If T has size one, then T is cyclable by definition. The one outstanding case is $k = 2$ as G is bipartite. Suppose that $T = \{u, v\}$ for some adjacent vertices $u \in V_1$ and $v \in V_2$. Since G has at least three vertices and T is not cyclable, either u or v, say u, is a cut-vertex of G. Then u has two neighbors x and y that are in distinct components of $G - u$. It follows that $\{x, y\}$ is not cyclable, as needed.

Since T is not a clique and not cyclable, there is a maximal safe pair (s, t), where $s \in \{1, \dots, p\}$ and $t \in \{1, \dots, q\}$, such that $T_{s,t} \neq T$. By Lemma 32, either there is $1 \leq s' \leq s$ and $S = \{u_{s'}, \dots, u_s\}$ or there is $1 \leq t' \leq t$ and $S = \{v_{t'}, \dots, v_t\}$ such that S is a T-separator and $c_T(G - S) > |S|$. By symmetry, assume without loss of generality that $S = \{u_{s'}, \dots, u_s\}$.

Let $v_{h'} \in V_2$ be the last vertex with respect to σ_2 that is adjacent to $u_{s'-1}$ if $s' \geq 2$ and $v_{h'} = v_1$ if $s' = 1$. Let also v_h be the first vertex with respect to σ_2 that is adjacent to u_{s+1} if $s < p$ and $v_h = v_q$ if $s = p$. Since S is a T-separator, we have that S is a separator of G. By the definition of a strong ordering and Lemma 28, we have that $h' < h$, the vertices $v_{h'}, \dots, v_h$ are in distinct components of $G - S$ and each component of $G - S$ contains exactly one vertex of $T' = \{v_{h'}, \dots, v_h\}$. Therefore,

$$|T'| = c_{T'}(G - S) = c(G - S) \geq c_T(G - S) > |S|.$$

So if $|T'| \leq k$ then we are done. If $|T'| > k$, then we select an arbitrary subset $T'' \subseteq T'$ such that T'' contains k consecutive vertices with respect to σ_2. We have that

$$c_{T''}(G - S) = |T''| = k \geq |T| \geq c_T(G - S) > |S|$$

and, therefore, T'' is not cyclable. This completes the proof. □

Theorem 44. *k-Cyclability can be solved in time $\mathcal{O}(nm)$ on bipartite permutation graphs.*

Proof. Let (G, k) be an instance of k-Cyclability where $G = (V_1, V_2, E)$ is a bipartite permutation graph. If $|V(G)| \leq 3$, then the problem is trivial, and if $k \leq 1$, then it can be solved in linear time by Proposition 6. So we can assume that $|V(G)| \geq 3$ and $k \geq 2$. If G is disconnected, then G is not k-cyclable. Suppose that G is connected. By Proposition 27, G has a strong ordering (σ_1, σ_2) that can be constructed in time $\mathcal{O}(n+m)$ [35]. Our algorithm then checks for every $i = 1, 2$ whether T is cyclable for every set $T \subseteq V_i$ of $\min\{|V_i|, k\}$ consecutive vertices with respect to σ_i. Each computation can be done, by Theorem 35, in time $\mathcal{O}(n+m)$. Applying Lemma 43, G is k-cyclable if and only if every such T is cyclable. The total running time is $\mathcal{O}(nm)$. □

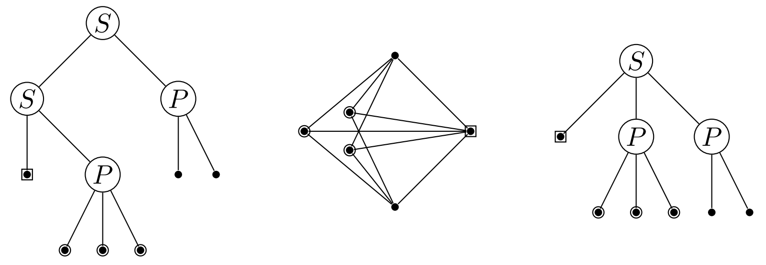

Fig. 5. Example of a composition tree using parallel and series operations (left), the cograph G it represents (center) and the cotree of G (right), which is unique. Some vertices are decorated in order to ease the reading.

Our algorithm for Terminal Cyclability on bipartite permutation graphs can be seen as a certifying algorithm for k-Cyclability as well: If (G, k) is a yes-instance for $G = (V_1, V_2, E)$, then we can produce the certificate that consists of a strong ordering (σ_1, σ_2) and a family of cycles containing T for each set $T \subseteq V_i$ of $\min\{|V_i|, k\}$ consecutive vertices with respect to σ_i $(i = 1, 2)$. If (G, k) is a no-instance, the certificate is a separator S of G such that $|S| \leq k-1$ and $c(G-S) > |S|$.

5. Cographs

In this section, we show Theorems 1 and 2 for cographs.

There are several characterizations of the class of *cographs*. They are often defined as the graphs that do not admit the P_4 (path on 4 vertices) as induced subgraph. Equivalently, they are the graphs obtained from a single vertex under the closure of the parallel composition and the series composition. The parallel composition of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the disjoint union of G_1 and G_2, *i.e.*, the graph $G_{par} = \big(V_1 \cup V_2, E_1 \cup E_2\big)$. The series composition of two graphs G_1 and G_2 is the disjoint union of G_1 and G_2 plus all possible edges from a vertex of G_1 to one of G_2, *i.e.*, the graph $G_{ser} = \big(V_1 \cup V_2, E_1 \cup E_2 \cup \{xy \mid x \in V_1, y \in V_2\}\big)$. These operations can naturally be extended to a finite number of graphs.

This gives a nice representation of a cograph G by a tree whose leaves are the vertices of the graph and whose internal nodes (nonleaf nodes) are labeled P, for parallel, or S, for series, corresponding to the operations used in the construction of G. It is always possible to find such a labeled tree $\mathcal{T}$ representing G such that every internal node has at least two children, no two parallel nodes are adjacent in $\mathcal{T}$ and no two series nodes are adjacent. This tree $\mathcal{T}$ is unique [11] and is called the *cotree* of G. See the example on Fig. 5.

We denote $\mathcal{T}_u$ the subtree of $\mathcal{T}$ rooted at u and $V(u)$ the subset of vertices of G that are the leaves of $\mathcal{T}_u$. Cographs form a hereditary family, i.e. every induced subgraph of a cograph is also a cograph. The cotree of the induced subgraph $G[V(u)]$ of cograph G is precisely $\mathcal{T}_u$. The set of children of node u of $\mathcal{T}$ is denoted $\mathcal{C}(u)$ and the subset of its children u_i such that $V(u_i) \cap T \neq \emptyset$ is denoted $\mathcal{C}_T(u)$, where as usual, T is the set of terminals given in the Terminal Cyclability problem.

The lists we use in the algorithm contain no duplicates. They are denoted with parentheses and their elements are separated by commas, e.g. (a, b, c, d). We use three operations on a list: `first` returns the first element of the list, `last` returns its last element and operation `Insert` inserts a given element in the list after a specified element. For example, if $L = (a, b, c, d)$, the operation `Insert` x after c in L updates L as $L = (a, b, c, x, d)$.

5.1. The algorithm for Cycle Segment Cover

The aim of this subsection is to prove the following theorem.

Theorem 45. *There is an algorithm that, given an instance (G, T, r) of* Cycle Segment Cover *where G is a cograph, given by its cotree $\mathcal{T}$, and T is not a clique, outputs in $\mathcal{O}(n)$ time either a T-cycle-segment cover of size at most r or a T-separator S^* with $c_T(G - S^*) - |S^*| > r$ that certifies a no-answer.*

The general idea of the algorithm we present is to recursively compute, for each node u of the cotree $\mathcal{T}$ of G, a T-cycle-segment cover of $T \cap V(u)$ in $G[V(u)]$ of minimum size. More precisely, we compute a quadruple $(\sigma(u), \mathcal{P}(u), Q(u), S(u))$ that satisfies the following invariant.

Invariant 1. *The quadruple $(\sigma(u), \mathcal{P}(u), Q(u), S(u))$ computed for each node $u \in \mathcal{T}$ is such that:*

1. *$\sigma(u) = \mathrm{seg}_{T\cap V(u)}(G[V(u)])$ and*
2. *$\mathcal{P}(u)$ is a minimum $(T \cap V(u))$-cycle-segment cover in $G[V(u)]$ and*
3. *$Q(u) = V(u) \setminus \bigcup_{P\in\mathcal{P}(u)} V(P)$.*
4. *if $\sigma(u) > 0$ then $S(u)$ is a separator of $G[V(u)]$ such that $c_T(G[V(u)] - S(u)) - |S(u)| = \sigma(u)$, and if $\sigma(u) = 0$ then $S(u) = \emptyset$.*

The algorithm terminates when the quadruple $(\sigma(r), \mathcal{P}(r), Q(r), S(r))$ has been computed for the root r of $\mathcal{T}$. As $V(r) = V(G)$, we then have that $\sigma(r) = \mathrm{seg}_T(G)$ and $\mathcal{P}(r)$ is a T-cycle-segment cover of G of size $\sigma(r)$. In addition, if $\sigma(r) > 0$, the algorithm produces a separator $S(r)$ such that $c_T(G[V(r)] - S(r)) - |S(r)| = \sigma(r)$ that is a certificate that $\mathcal{P}(r)$ is a minimum T-cycle-segment cover. The purpose of $Q(u)$ is that it contains all the vertices that are not used in $\mathcal{P}(u)$ and that are available to connect some paths later in the algorithm, when treating the ancestors of u. The paths P in $\mathcal{P}(u)$ are represented by their sequence of vertices $P = x_1, x_2, \ldots, x_l$. When $\mathcal{P}(u)$ contains a unique element which is a cycle, this cycle is also represented as a path, which satisfies the additional condition that x_1 and x_l are adjacent.

The algorithm uses two recursive subroutines that call each other: one for series nodes, named `CoverSeries`(u), and one for parallel nodes, named `CoverParallel`(u). The first call to one of these subroutines is made on the root of the cotree and therefore the whole algorithm terminates when the call made on the root is over. Routine `CoverSeries`(u) calls `CoverParallel`(u_1) on a unique specific child u_1 of u, while `CoverParallel`(u) calls `CoverSeries`(u_i) on all the children u_i of u such that $V(u_i)$ contains some terminal. We now describe the two routines `CoverSeries` and `CoverParallel`.

5.1.1. Algorithm for parallel nodes

Algorithm 2: `CoverParallel`(u)

```
1  Pre-conditions: $u$ is a parallel node.
2  begin
3      sort the children $\{u_1, \ldots, u_k\}$ of $u$ such that the children $u_i \in \mathcal{C}_T(u)$ appear first;
4      for $i = 1$ to $|\mathcal{C}_T(u)|$ do
5          $(\sigma_i, \mathcal{P}_i, Q_i, S_i) \leftarrow$ CoverSeries($u_i$);
6      end
7      if $|\mathcal{C}_T(u)| = 1$ then
8          $(\sigma, \mathcal{P}, Q, S) \leftarrow (\sigma_1, \mathcal{P}_1, Q_1 \cup (V(u) \setminus V(u_1)), S_1)$;
9      else
10         $\sigma \leftarrow \sum_{u_i \in \mathcal{C}_T(u)} |\mathcal{P}(u_i)|$;
11         $\mathcal{P} \leftarrow \bigcup_{u_i \in \mathcal{C}_T(u)} \mathcal{P}(u_i)$;
12         $Q \leftarrow \bigcup_{u_i \in \mathcal{C}_T(u)} Q(u_i) \cup \bigcup_{u_i \in \mathcal{C}(u) \setminus \mathcal{C}_T(u)} V(u_i)$;
13         $S \leftarrow \bigcup_{u_i \in \mathcal{C}_T(u)} S(u_i)$;
14     end
15     return $(\sigma, \mathcal{P}, Q, S)$;
16 end
```

The routine for parallel nodes is called `CoverParallel`(u) (Algorithm 2). It is very simple. Given the quadruples $(\sigma(u_i), \mathcal{P}(u_i), Q(u_i), S(u_i))$ computed for each child $u_i \in \mathcal{C}_T(u)$, it outputs the quadruple $(\sigma(u), \mathcal{P}(u), Q(u), S(u))$ for u which is obtained as described in Algorithm 2.

The validity of Routine `CoverParallel` is stated by the following lemma.

Lemma 46. *When u is a parallel node, the quadruple $(\sigma, \mathcal{P}, Q, S)$ returned by Routine `CoverParallel`(u) satisfies Invariant 1.*

Proof. We have to check that the four properties of Invariant 1 are satisfied. Let us start with the general case where $\mathcal{C}_T(u) > 1$. From Invariant 1, for each child $u_i \in \mathcal{C}_T(u)$ we have $Q(u_i) = V(u_i) \setminus \bigcup_{P \in \mathcal{P}(u_i)} V(P)$. Therefore, since $\mathcal{P}$ is set to $\bigcup_{u_i \in \mathcal{C}_T(u)} \mathcal{P}(u_i)$ (Line 11) and $Q(u)$ is set to $\bigcup_{u_i \in \mathcal{C}_T(u)} Q(u_i) \cup \bigcup_{u_i \in \mathcal{C}(u) \setminus \mathcal{C}_T(u)} V(u_i)$ (Line 12), then we also have $Q(u) = V(u) \setminus \bigcup_{P \in \mathcal{P}(u)} V(P)$ and 3. of Invariant 1 is satisfied.

Observe that since u is a parallel node, for each child u_i of u, $V(u_i)$ is a connected component of $G[V(u)]$. Then, since $|\mathcal{C}_T(u)| > 1$, there is no cycle in $G[V(u)]$ containing all the vertices of $T \cap V(u)$. In addition, $\mathcal{P}(u) = \bigcup_{u_i \in \mathcal{C}_T(u)} \mathcal{P}(u_i)$ is clearly a $T \cap V(u)$-cycle-segment cover in $G[V(u)]$ (recall that cycles are represented as paths) and we have $\sigma(u) = |\mathcal{P}(u)|$. Then, in order to prove that Properties 1. and 2. hold, it is enough to exhibit a separator S of $G[V(u)]$ such that $c_T(G[V(u)] - S) - |S| = \sigma(u)$. This is obviously given by $S(u)$ if Property 4. holds, which we prove now

Note that, because for every child $u_i \in \mathcal{C}_T(u)$ the quadruple $(\sigma(u_i), \mathcal{P}(u_i), Q(u_i), S(u_i))$ satisfies Property 4., if $\sigma(u_i) > 0$, then $S(u_i)$ is such that $c_T(G[V(u_i)] - S(u_i)) - |S(u_i)| = \sigma(u_i)$. Let us denote C the subset of children u_i of u such that $V(u_i)$ contains some vertex of T (*i.e.* $u_i \in \mathcal{C}_T(u)$ with our notations) and $\sigma(u_i) > 0$. Since, by definition, $S(u_i) = \emptyset$ when $\sigma(u_i) = 0$, then we have $S(u) = \bigcup_{u_i \in C} S(u_i)$. Clearly, $S(u)$ is a separator of $G[V(u)]$. Moreover, observe that for any $u_i \in C$ the connected components of $G[V(u)] - S(u)$ that contain some vertex of $T \cap V(u_i)$ are exactly the connected components of $G[V(u_i)] - S(u_i)$ that contain some vertex of T, because $S(u)$ is such that $S(u) \cap V(u_i) = S(u_i)$. Furthermore, observe that for any $u_i \in \mathcal{C}_T(u) \setminus C$, $V(u_i)$ is a connected component of $G[V(u)] - S(u)$ and contains some vertex of T. Therefore, $c_T(G[V(u)] - S(u)) - |S(u)| = |\mathcal{C}_T(u) \setminus C| + \sum_{u_i \in C}(c_T(G[V(u_i)] - S(u_i)) - |S(u_i)|)$. For $u_i \in \mathcal{C}_T(u) \setminus C$, we have $|\mathcal{P}(u_i)| = 1$ and so $|\mathcal{C}_T(u) \setminus C| = \sum_{u_i \in \mathcal{C}_T(u) \setminus C} |\mathcal{P}(u_i)|$. On the other hand, for $u_i \in C$, we have

$c_T(G[V(u_i)] - S(u_i)) - |S(u_i)| = \sigma(u_i) = |\mathcal{P}(u_i)|$, from 4.. Then, by injecting these equalities in the identity above we get $c_T(G[V(u)] - S(u)) - |S(u)| = \sum_{u_i \in \mathcal{C}_T(u) \setminus C} |\mathcal{P}(u_i)| + \sum_{u_i \in C} |\mathcal{P}(u_i)| = \sum_{u_i \in \mathcal{C}_T(u)} |\mathcal{P}(u_i)| = |\mathcal{P}(u)|$, from the definition of $\mathcal{P}(u)$ at Line 11. Finally, we obtain $c_T(G[V(u)] - S(u)) - |S(u)| = |\mathcal{P}(u)|$, which proves that 4. holds, and then, as noted before, Properties 1. and 2. also hold.

Now, consider the case where $|\mathcal{C}_T(u)| = 1$ and denote u_1 the unique element of $\mathcal{C}_T(u)$. Since $V(u_1)$ is a connected component of $G[V(u)]$, we have $\sigma(u) = \sigma(u_1)$. In this case (see Lines 7 and 8), the quadruple returned by `CoverParallel` is clearly correct: it is the same as the quadruple of u_1, except for $Q(u)$, which is augmented with the vertices of $V(u) \setminus V(u_1)$. □

Time complexity of `CoverParallel`. If we exclude the time spent for the calls to Routine `CoverSeries` (Lines 4 to 6) and provided that the children of u are already sorted as stated at Line 3 (we will do this for all nodes of the tree at the same time as a preprocessing step before the first call to Routines `CoverParallel` and `CoverSeries`), then the time needed for Routine `CoverParallel` to compute the quadruple $(\sigma(u), \mathcal{P}(u), Q(u), S(u))$ is $O(|\mathcal{C}(u)|)$. Indeed, it is enough to scan the children of u while computing the desired sum for $\sigma(u)$ and performing the desired union for $\mathcal{P}(u)$, $Q(u)$ and $S(u)$. Representing all these sets by lists, the union of two sets is performed in constant time. Then the whole time taken by Routine `CoverParallel` is the time needed to scan the children of u, that is $O(|\mathcal{C}(u)|)$.

5.1.2. Algorithm for series nodes

The algorithm for series node (Algorithm 4) is called `CoverSeries`(u). It uses the quadruple $(\sigma(u_1), \mathcal{P}(u_1), Q(u_1), S(u_1))$ already computed for its child u_1 such that $|V(u_1)|$ is maximum and calls Routine `ExtendCov`$(X, \sigma, \mathcal{P}, Q, S)$ (Algorithm 3) once for each of its children u_i, $i \geq 2$, distinct from u_1. We first describe Routine `ExtendCov`$(X, \sigma, \mathcal{P}, Q, S)$.

Routine `ExtendCov`. Given two disjoint subsets of vertices X and Y that are complete to each other (all possible edges between X and Y) and given a minimum $(T \cap Y)$-cycle-segment cover in $G[Y]$ (encoded as a quadruple $(\sigma, \mathcal{P}, Q, S)$ that satisfies Invariant 1), Routine `ExtendCov` computes a minimum $(T \cap (X \cup Y))$-cycle-segment cover in $G[X \cup Y]$. It uses a list *Seq* of paths and a `merge` operation which consists in concatenating all the paths in list *Seq*.

The main idea of Routine `ExtendCov` is as follows. First, we use the vertices of X (and prioritarily those of $T \cap X$) to connect the paths in $\mathcal{P}$ until they eventually form one cycle (Lines 9 to 16). We can do this as X is complete to Y. If there are not enough vertices in X to obtain a cycle in this way, the Routine stops and return the reduced set of paths obtained. Otherwise, if we obtain a cycle before using all the vertices of X, it may happen that the terminals in $T \cap X$ are not all contained in this cycle (even if we use the terminals of $T \cap X$ before the other vertices in X). Then, the routine includes them in the cycle. In order to ensure that we can include all of them in the cycle, we proceed in two steps. As long as there remain some unused vertices in Q, that is some vertices in Y that are not in the cycle, we insert at the same time one vertex of Q and one vertex of $T \cap X$ (Lines 18 to 22). This ensures, together with the pre-condition $|Y| \geq |X|$ of Routine `ExtendCov`, that if there are no more vertices of Q we can use before we inserted all the vertices of $T \cap X$, then the number of vertices remaining in $T \cap X$ is less than the number of edges in the paths of $\mathcal{P}$. Then, each remaining vertex of $T \cap X$ can be inserted instead of one edge of some path in $\mathcal{P}$ (Lines 24 to 32), as the paths of $\mathcal{P}$ are made only of vertices in Y.

Validity of routine `ExtendCov`. It is stated by Lemma 50. In order to prove it, we first establish three preliminary technical lemmas.

Lemma 47. *At the end of Routine `ExtendCov`$(X, \sigma, \mathcal{P}, Q, S)$, $\mathcal{P}'$ consists of a collection of disjoint paths that contain all the vertices of $T \cap (X \cup Y)$.*

Proof. Let us first show that $\mathcal{P}'$ is a collection of disjoint paths. At the end of the algorithm (Lines 33 and 34), $\mathcal{P}'$ is built by merging into one path the sequence *Seq* built along the algorithm and by adding to $\mathcal{P}'$ the paths of $\mathcal{P}$ that are not in *Seq*. So we have to check that the sequence *Seq* of vertices built indeed defines a path and that the (rest of the) paths of $\mathcal{P}$, which are altered during the algorithm, at Line 28, are still paths at the end of the algorithm. To see that *Seq* indeed defines a path, observe that *Seq* is initialized with the path P_1 of $\mathcal{P}$ at Line 8 and that *Seq* is updated at Line 11 of the algorithm by placing a vertex of X between two paths of $\mathcal{P}$, whose vertices are in Y. Consequently, since X is complete to Y (see Pre-condition *(ii)*), *Seq* remains a path. At Line 16, before *Seq* is updated by adding a vertex of X at the end, the last element in *Seq* was a path of $\mathcal{P}$ which is made only of vertices of Y, so *Seq* again remains a path, which now finishes by a vertex in X. Then, at Line 19, when *Seq* is updated again by appending to it the sequence (y_l, x_j), *Seq* continues to define a path as well. Finally, the paths in $\mathcal{P}$ that are in *Seq* are altered at Line 28 (if there exists some path in $\mathcal{P}$ that is not in *Seq*, then at Line 16, we have $\sigma' > 1$ and so $j = n + 1$, and Lines 18 to 32 are not executed). At that time a vertex $x_j \in X$ is inserted after the vertex α of some path $P_l \subseteq Y$. Again, because X is complete to Y, then x_j is adjacent to both α and the next vertex on P_l and P_l remains a path after the insertion of x_j. Thus, at the end of Routine `ExtendCov`, $\mathcal{P}'$ is a collection of disjoint paths.

We now show that $\mathcal{P}'$ contains all the vertices of $T \cap (X \cup Y)$. Recall that from Pre-condition *(ii)*, $n \leq |Y|$ and so $t \leq |Y|$. After the first loop (Lines 10 to 14) and the conditional following it (Line 16), we have inserted exactly $j - 1$ vertices

Algorithm 3: ExtendCov($X, \sigma, \mathcal{P}, Q, S$)

```
1  Pre-conditions: (i) $X \subseteq V(G)$ and (ii) $\sigma = \mathrm{seg}_{T\cap Y}(G[Y])$ for some $Y \subseteq V(G) \setminus X$ such that $Y$ is complete to $X$ and
   $|Y| \geq |X|$ and (iii) $\mathcal{P}$ is a minimum $(T \cap Y)$-cycle-segment cover in $G[Y]$ and (iv) $Q = Y \setminus \bigcup_{P\in\mathcal{P}} V(P)$ and (v) if
   $\sigma > 0$ then $S$ is such that $c_{T\cap Y}(G[Y] - S) - |S| = \sigma$ and if $\sigma = 0$ then $S = \emptyset$.
2  begin
3      let $n = |X|$, $t = |T \cap X|$;
4      we denote $X = \{x_1, \dots, x_n\}$ the vertices of $X$ sorted so that
5      the vertices of $T \cap X = \{x_1, \dots, x_t\}$ form an interval in the beginning, when $t > 0$;
6      we denote $\mathcal{P} = \{P_1, \dots, P_{|\mathcal{P}|}\}$, where $|\mathcal{P}| \geq 1$, and $Q = \{y_1, \dots, y_{|Q|}\}$ when $|Q| \geq 1$;
7      $\sigma' \leftarrow \sigma$; $Q' \leftarrow Q \cup X$;
8      $j \leftarrow 1$; $Seq \leftarrow (P_1)$;
9      if $\sigma' > 0$ then
10         while $j \leq n$ and $\sigma' > 1$ do
11             $Seq \leftarrow Seq.(x_j, P_{j+1})$;
12             $Q' \leftarrow Q' \setminus \{x_j\}$;
13             $j \leftarrow j + 1$; $\sigma' \leftarrow \sigma' - 1$;
14         end
15     end
16     if $j \leq n$ then $Seq \leftarrow Seq.(x_j)$; $Q' \leftarrow Q' \setminus \{x_j\}$; $j \leftarrow j + 1$; $\sigma' \leftarrow \sigma' - 1$;
17     $l \leftarrow 1$;
18     while $j \leq t$ and $l \leq |Q|$ do
19         $Seq \leftarrow Seq.(y_l, x_j)$;
20         $Q' \leftarrow Q' \setminus \{y_l, x_j\}$;
21         $j \leftarrow j + 1$; $l \leftarrow l + 1$;
22     end
23     $h \leftarrow 0$;
24     while $j \leq t$ and $h < |\mathcal{P}|$ do
25         $h \leftarrow h + 1$; $\alpha \leftarrow$ first($P_h$);
26         while $j \leq t$ and $\alpha \neq$ last($P_h$) do
27             $\alpha_{next} \leftarrow$ next($\alpha$);
28             Insert $x_j$ between $\alpha$ and $\alpha_{next}$ in $P_h$;
29             $Q' \leftarrow Q' \setminus \{x_j\}$;
30             $\alpha \leftarrow \alpha_{next}$; $j \leftarrow j + 1$;
31         end
32     end
33     if $\sigma' \geq 2$ then $\mathcal{P}' \leftarrow \{$merge($Seq$)$, P_{|\mathcal{P}|-\sigma'+2}, \dots, P_{|\mathcal{P}|}\}$;
34                else $\mathcal{P}' \leftarrow \{$merge($Seq$)$\}$;
35     if $\sigma' > 0$ then $S' \leftarrow S \cup X$;
36                else $S' \leftarrow \emptyset$;
37     return $(\sigma', \mathcal{P}', Q', S')$;
38 end
```

of X in $\mathcal{P}$. Then, either we have $j - 1 \geq t$ and all the vertices of $T \cap X$ have been inserted in $\mathcal{P}$, or we have $j \leq t$ and the second loop (Lines 18 to 22) will iterate at least once. Note that before the second loop starts, we have $j \leq t \leq n$, which implies that the first loop stopped because $\sigma' = 1$ and that the conditional of Line 16 was positive, so we now have $\sigma' = 0$. Since σ' was initialized at σ and decremented always at the same time as j is incremented, then, at Line 17 of the algorithm, just before the second loop starts, we have inserted $j - 1 = \sigma$ vertices of X in $\mathcal{P}$. After the second loop stops, if we have $j > t$, since again, at that point, $j - 1 = t$ vertices of X have been inserted in $\mathcal{P}$, then all those in $T \cap X$ have been inserted. Otherwise, if $j \leq t$, the second loop stopped because $l = |Q| + 1$ and then exactly $|Q|$ new vertices of X have been inserted in $\mathcal{P}$ during the execution of the second loop. This gives a total of $|\mathcal{P}| + |Q|$ vertices of X inserted in $\mathcal{P}$ so far and the third loop (Lines 24 to 32) is ready to iterate at least once, because $j \leq t$. Now, we show that when the third loop stops, we always[4] have $j > t$. If $h < |\mathcal{P}|$, this is clear. On the other hand, if $h \geq |\mathcal{P}|$, then the third loop was executed once for each of the paths $P_l \in \mathcal{P}$. And for each path P_l, the inner loop (Lines 26 to 31) executes exactly $|E(P_l)|$ times and insert one vertex of X in $\mathcal{P}$ at each iteration (Line 28). Then, during the execution of the third

[4] In other words, the loop condition $l < |Q|$ is not necessary, it is used only to explicitly guarantee to the reader that the third loop will never iterate with a value of l for which P_l is undefined. But actually, this is already guaranteed by the sole condition $j \leq t$ and the pre-conditions of Routine ExtendCov, as we prove it now.

loop, $\sum_{P\in\mathcal{P}}|E(P)|$ new vertices of X are inserted in $\mathcal{P}$, which makes the total number of such vertices reach the value $|\mathcal{P}|+|Q|+\sum_{P\in\mathcal{P}}|E(P)|$, and as usual the incrementation of j during the algorithm guarantees that this value is also $j-1$. So we have $j-1=|\mathcal{P}|+|Q|+\sum_{P\in\mathcal{P}}|E(P)|$. Moreover, from Pre-condition *(iv)* we have $|Q|=n-\sum_{P\in\mathcal{P}}|V(P)|$ and $\sum_{P\in\mathcal{P}}|E(P)|$ also writes $(\sum_{P\in\mathcal{P}}|V(P)|)-|\mathcal{P}|$. This gives $j-1=n\geq t$, which means that all the vertices in $T\cap X$ have been inserted in $\mathcal{P}'$, which ends the proof of the lemma. □

Lemma 48. *At the end of Routine ExtendCov($X,\sigma,\mathcal{P},Q,S$), if $|\mathcal{P}'|>1$ or $|\mathcal{P}'|=1$ and the unique element in $\mathcal{P}'$ is not a cycle, then $\sigma'=|\mathcal{P}'|$. Otherwise, i.e. if $|\mathcal{P}'|=1$ and the unique element in $\mathcal{P}'$ is a cycle, then $\sigma'=0$.*

Proof. Let us first consider the case where $\sigma>0$. In an execution of the algorithm, σ' is initialized at σ at Line 7 and from Pre-condition *(ii)* and *(iii)*, $\sigma=|\mathcal{P}|$. Then, in the first loop, σ' is decremented (Line 13) every time that two paths in $\mathcal{P}$ are joined together by a vertex of x (Line 11). If at the end of the execution of the first loop we have $j>n$, then the instructions at Lines 16 to 32 are not executed and it follows that at the end of the algorithm $\sigma'=|\mathcal{P}'|$. Moreover, since $\sigma>0$, by definition of $\mathcal{P}$, the last vertex of the last path in $\mathcal{P}$ and the first vertex of the first path in $\mathcal{P}$ are not adjacent. This implies that if $|\mathcal{P}'|=1$ then the unique element in $\mathcal{P}'$ is not a cycle and we are in the first case of the statement of the lemma. On the other hand, if $j\leq n$ at the end of the first loop, then $\mathcal{P}'$ contains only one path and this unique path is made a cycle by inserting a vertex of X at the end of it at Line 16, and σ' takes the value 0. Afterwards, in the rest of the execution of the algorithm, the value of σ' remains unchanged and the unique element in $\mathcal{P}'$ remains a cycle (see the proof of Lemma 47). Therefore, we are in the second case of the lemma.

Finally, if $\sigma=0$, then σ' is set to 0 at Line 7 and the first loop does not execute. Consequently, at Line 17, we have $\sigma'=0$ and the unique element in $\mathcal{P}'$, which is the unique element in $\mathcal{P}$, is a cycle. As above, it follows that the value of σ' remains unchanged and the unique element in $\mathcal{P}'$ remains a cycle until the end of the execution of the algorithm. We are again in the second case of the lemma. □

Lemma 49. *At the end of Routine ExtendCov($X,\sigma,\mathcal{P},Q,S$), if $\sigma'>0$, then we have $c_{T\cap(X\cup Y)}(G[X\cup Y]-S')-|S'|=|\mathcal{P}'|$.*

Proof. Since σ' only decreases during the routine, if $\sigma'>0$ at the end then $\sigma'>0$ at the beginning. Consequently, the condition $\sigma'>0$ at Line 9 is true and the first loop (Lines 10 to 14) executes (note that its number of iterations may be null though). Assume for contradiction that the condition $j\leq n$ at Line 16 is true. Since at the end of the first loop, we have $j>n$ or $\sigma'\leq 1$, it follows that $\sigma'\leq 1$. Moreover, since σ' decreases by exactly one at each iteration of the first loop, we must have $\sigma'=1$. Since the condition $j\leq n$ at Line 16 is true, σ' is then decremented (at Line 16 as well) and we have $\sigma'=0$ at the end of Routine ExtendCov, which is a contradiction. Therefore, when $\sigma'>0$ at the end of the routine, the condition at Line 16 is false, i.e. $j>n$. Consequently, Lines 18 to 32 are not executed. This implies that at the end of the algorithm $|\mathcal{P}'|=\sigma-n$, because each of the vertices of X that was added to *Seq* is between two paths of $\mathcal{P}$.

Beside this, observe that when $\sigma'>0$, at the end of the routine, we have $S'=S\cup X$ (see Line 35) and so $|S'|=|S|+n$. Moreover, note that $\sigma'>0$ at the end implies that $\sigma>0$. Then, from Pre-condition *(v)*, S is such that $c_{T\cap Y}(G[Y]-S)-|S|=\sigma$. Finally, observe that the connected components of $G[Y]-S$ and of $G[X\cup Y]-S'$ are exactly the same. As a consequence, we have $c_{T\cap(X\cup Y)}(G[X\cup Y]-S')-|S'|=c_{T\cap Y}(G[Y]-S)-|S'|=c_{T\cap Y}(G[Y]-S)-|S|-n=\sigma-n=|\mathcal{P}'|$, which ends the proof of the lemma. □

We can now prove the correctness of Routine ExtendCov.

Lemma 50. *When the pre-conditions of Routine ExtendCov($X,\sigma,\mathcal{P},Q,S$) are satisfied, the quadruple $(\sigma',\mathcal{P}',Q',S')$ returned by the routine satisfies the following four post-conditions:*

(1) $\sigma'=\mathrm{seg}_{T\cap(X\cup Y)}(G[X\cup Y])$ and
(2) $\mathcal{P}'$ is a minimum $(T\cap(X\cup Y))$-cycle-segment cover in $G[X\cup Y]$ and
(3) $Q'=(X\cup Y)\setminus\bigcup_{P\in\mathcal{P}'}V(P)$.
(4) if $\sigma'>0$ then S' is such that $c_{T\cap(X\cup Y)}(G[X\cup Y]-S')-|S'|=\sigma'$ and if $\sigma'=0$ then $S'=\emptyset$

Proof. First, observe that the four post-conditions above are the same as the conditions of Invariant 1, the difference being that Invariant 1 is stated for a node of the cotree, while here these conditions are applied more generally on an arbitrary subset of vertices. We prove that each of the conditions of the lemma holds at the end of the execution of the routine, starting with Condition (3). At the beginning of the algorithm (Line 7), Q' is initialized with $Q\cup X$, so at that time we have indeed $Q'=(X\cup Y)\setminus\bigcup_{P\in\mathcal{P}'}V(P)$. During the algorithm, every time some vertex of Q or X is incorporated in $\mathcal{P}$, it is withdrawn of Q' immediately after. So at the end of the algorithm, we still have $Q'=(X\cup Y)\setminus\bigcup_{P\in\mathcal{P}'}V(P)$ and Condition (3) is satisfied.

Condition (4) always holds at the end of the routine, because if $\sigma'=0$ then S' is set to $\emptyset$ at Line 36 and if $\sigma'>0$, then Lemmas 48 and 49 ensure Condition (4).

We now prove that Condition (2) holds. From Lemma 47, $\mathcal{P}'$ is a cycle-segment cover of $T\cap(X\cup Y)$. We need to prove in addition that it is a minimum $(T\cap(X\cup Y))$-cycle-segment cover. We distinguish two cases. First, if $|\mathcal{P}'|=1$ and the unique element in $\mathcal{P}'$ is a cycle, then $\mathcal{P}'$ is necessarily a minimum $(T \cap (X \cup Y))$-cycle-segment cover in $G[X \cup Y]$. Now, consider the case where $|\mathcal{P}'| > 1$ or $|\mathcal{P}'| = 1$ and the unique element in $\mathcal{P}'$ is not a cycle. From Lemma 48, in this case we have $\sigma' > 0$. Then, from Lemma 49, we have $c_{T\cap(X\cup Y)}(G[X \cup Y] - S') - |S'| = |\mathcal{P}'|$, which ensures that the segment cover $\mathcal{P}'$ is of minimum cardinality. Thus, in any case, Condition (2) holds at the end of the routine.

Finally, Condition (2) and Lemma 48 directly imply that Condition (1) holds as well, which ends the proof. □

Time complexity of routine `ExtendCov`. With a suitable implementation, Routine `ExtendCov` runs in $O(|X|)$ time. Sorting the vertices of X at Lines 4 and 5 so that the vertices of T appear as an interval at the beginning takes $O(|X|)$ time. In our implementation, X, $\mathcal{P}$, Q, Q', Seq and the paths in $\mathcal{P}$ are stored as lists. By managing appropriately pointers to these lists, all basic operations involved in the algorithm take constant time. Note that list Q' is stored in two parts, one for the vertices of Q and one for the vertices of X. At Lines 33 and 34, since all elements of Seq are lists, merging all of them takes $O(|Seq|)$ time. Moreover, since every second element in Seq is a path formed by a single vertex of X and since all these vertices are pairwise distinct, we have $O(|Seq|) = O(|X|)$, which is the time needed to perform `merge`(Seq). Then, in order to bound the complexity of `ExtendCov` it is sufficient to bound the total number of iterations of the four loops it contains. Let us start with the third loop (Lines 24 to 32) and observe that this loop iterates only when $j \le t$ just before Line 24, which implies that we had $j \le n$ (as $t \le n$) after the first loop stopped, at Line 14. Then, at that time, the first loop stopped because $\sigma' = 1$, which means that all the paths in $\mathcal{P}$ have been placed in Seq. Therefore, if the third loop iterates, we have $|\mathcal{P}| \le |Seq| = O(|X|)$, as explained above. As the third loop does not iterate more than $|\mathcal{P}|$ times (see condition $l < |\mathcal{P}|$) then its number of iterations is also $O(|X|)$. Finally, observe that in the first loop (Line 10 to 14), in the second loop (Line 18 to 22) and in the inner loop (Line 26 to 31) nested in the third one, at each iteration a distinct vertex of X leaves Q, so the total cumulated number of iterations of all these loops cannot exceed $|X|$ and the overall running time of Routine `ExtendCov` is $O(|X|)$.

Routine `CoverSeries`. The routine for treating a series node u is called `CoverSeries`. It is described in Algorithm 4 and mainly consists in iteratively calling Routine `ExtendCov` on all the children of u but one, on which Routine `CoverParallel` has been previously called.

Algorithm 4: `CoverSeries`(u)

```
1  Pre-conditions: $u$ is a series node.
2  begin
3      sort the children $\{u_1, \dots, u_k\}$ of $u$ such that
4      the children $u_i \in \mathcal{C}_T(u)$ appear first and $|V(u_1)|$ is maximum among children in $\mathcal{C}_T(u)$;
5      $(\sigma, \mathcal{P}, Q, S) \leftarrow$ CoverParallel($u_1$);
6      for $i = 2$ to $k$ do
7          $(\sigma', \mathcal{P}', Q', S') \leftarrow$ ExtendCov($V(u_i), \sigma, \mathcal{P}, Q, S$);
8          $(\sigma, \mathcal{P}, Q, S) \leftarrow (\sigma', \mathcal{P}', Q', S')$;
9      end
10     return $(\sigma, \mathcal{P}, Q, S)$;
11 end
```

Validity of routine `CoverSeries`. It is stated by the following lemma.

Lemma 51. *When u is a series node, the quadruple $(\sigma, \mathcal{P}, Q, S)$ returned by Routine* `CoverSeries`*(u) satisfies Invariant 1.*

Proof. The quadruple returned by the call to `CoverParallel`(u_1) at Line 5 satisfies the conditions of Invariant 1 for u_1 (see Section 5.1.1). Because these conditions are satisfied and because the $V(u_i)$'s are pairwise disjoint and complete to each other (since u is series), then, when the call to `ExtendCov`($V(u_i), \sigma, \mathcal{P}, Q, S$) occurs at Line 7 in one iteration of the loop, the pre-conditions of `ExtendCov` are satisfied, with $X = V(u_i)$, $i \ge 2$ and $Y = \bigcup_{j<i} V(u_j)$. Then, from Lemma 50, at the end of the iteration of the loop, we have $\sigma = \mathrm{seg}_{T\cap\bigcup_{j\le i} V(u_j)}(G[\bigcup_{j\le i} V(u_j)])$ and $\mathcal{P}$ is a minimum $(T \cap \bigcup_{j\le i} V(u_j))$-cycle-segment cover in $G[\bigcup_{j\le i} V(u_j)]$ and $Q = \bigcup_{j\le i} V(u_j) \setminus \bigcup_{P\in\mathcal{P}} V(P)$ and if $\sigma > 0$ then S is such that $c_{T\cap\bigcup_{j\le i} V(u_j)}(G[\bigcup_{j\le i} V(u_j)] - S) - |S| = \sigma$ and if $\sigma = 0$ then $S = \emptyset$. Consequently, the pre-conditions of `ExtendCov` are satisfied in the next iteration of the loop, with $X = V(u_{i+1})$, and $Y = \bigcup_{j\le i} V(u_j)$. When the loop stops to iterate, we have $i = k$ and $\bigcup_{j\le i} V(u_j) = V(u)$. Thus, the quadruple $(\sigma, \mathcal{P}, Q, S)$ returned by `CoverSeries` at Line 10 satisfies the four properties of Invariant 1. □

Time complexity of routine `CoverSeries`. If we exclude the time spent for the call to `CoverParallel`, which will be counted separately, in the complexity of `CoverParallel`, the time spent for a call to `CoverSeries`(u) is entirely due to the time spent for the calls to `ExtendCov` in the loop. From Section 5.1.2, each call takes time $O(|V(u_i)|)$, for $i \ge 2$. Consequently, the total time spent for all the calls to `ExtendCov` is $O(\sum_{i\ge 2} |V(u_i)|)$ and this is the overall time complexity of `CoverSeries`.

5.1.3. The whole algorithm for Cycle Segment Cover

The algorithm for Cycle Segment Cover of a cograph G mainly consists in calling one of the routines `CoverSeries` or `CoverParallel` on the root r of the cotree $\mathcal{T}$ of G: Routine `CoverSeries` if r is a series node and Routine `CoverParallel` if r is a parallel node. Then, these two routines call each other in a recursive way along the tree until the initial call made on the root terminates. When this call terminates, it returns (as any other call to routines `CoverSeries` and `CoverParallel`) a quadruple $(\sigma, \mathcal{P}, Q, S)$ which satisfies Invariant 1, as proven by Lemmas 46 and 51. Therefore, we have $\sigma = \mathsf{seg}_T(G)$, $\mathcal{P}$ is a minimum T-cycle-segment cover in G and S is certificate of this (when $\mathsf{seg}_T(G) > 0$, otherwise the cycle being the unique element of $\mathcal{P}$ is itself a certificate that $\mathsf{seg}_T(G) = 0$).

Let us now turn to the time complexity of this algorithm, which we will prove to be $O(n)$. Note that Routines `CoverSeries` and `CoverParallel` both need the children of the node u which is processed to be sorted in a special way: the children u_i whose subtree contains some terminal (i.e. $u_i \in \mathcal{C}_T(u)$, with the notations we adopted) must appear before the others and for u a series node, a child $u_i \in \mathcal{C}_T(u)$ such that $|V(u_i)|$ is largest must appear in first position. For complexity reasons, these sortings of the children lists of the nodes of the cotree $\mathcal{T}$ are made altogether in a preprocessing step that occur before the beginning of the algorithm in itself, that is before the first call to `CoverSeries` or `CoverParallel` on the root of $\mathcal{T}$. This preprocessing consists of a bottom-up traversal of the cotree $\mathcal{T}$ of G, starting from the leaves. In this process, each node v forwards to its parent u the number of leaves $|V(v)|$ in its subtree and the number $|V(v) \cap T|$ of these leaves that are terminals. Once a node has received the information from all its children, it determines its own and forward it to its parent. In addition, when $|V(v) \cap T| \neq 0$, v is placed in the front of the children list of u, and for u a series node, we keep track of its child v with greatest $|V(v)|$ and we place it again in the first position once all children of u have been processed.

Clearly, the complexity of the preprocessing step is $O(n)$ and this is also the complexity of the whole algorithm. Indeed, we already showed that, provided that the children of u are properly sorted (see the preprocessing step above), the complexity of one call to `CoverParallel`(u) is $O(|\mathcal{C}(u)|)$ and the complexity of one call to `CoverSeries`(u) is $O(\sum_{i\geq 2} |V(u_i)|)$, with u_1 the child of u in $\mathcal{C}_T(u)$ such that $|V(u_1)|$ is largest. Moreover, for a series node u, there is no call to `CoverParallel` that is made on the children of u different from u_1. Also note that obviously, computing the quadruple $(\sigma(x), \mathcal{P}(x), Q(x), S(x))$ for a terminal leaf $x \in T$ takes constant time. Then, a simple induction shows that the time needed to compute the quadruple $(\sigma(u), \mathcal{P}(u), Q(u), S(u))$ for a node u, either by a call to `CoverParallel`(u) or a call to `CoverSeries`(u), takes a time proportional to the size of $\mathcal{T}_u$, which is also $O(|V(u)|)$ since every node has at least two children. Thus, the overall complexity of the algorithm is $O(|V(r)|) = O(n)$ time.

5.2. The algorithm for k-Cyclability

We now consider the k-Cyclability problem on cographs. Theorem 45 gives the following corollary.

Corollary 52. *The class of cographs is cycle-scattering dual.*

Proof. The proof is the same as the proof of Corollary 23, replacing Corollary 22 with Theorem 45. □

Using Lemma 9 and Corollary 52, we construct a polynomial algorithm that computes the k-scattering number of G for any $k \leq n-1$. Afterwards, the only task remaining will consist in finding the largest integer k such that $\mathsf{sc}^k(G) \leq 0$. We prove the following theorem.

Theorem 53. *For a noncomplete cograph G, the scattering numbers $\mathsf{sc}^k(G)$ for all $k \in \{1, \dots, n\}$ can be computed and k-Cyclability can be solved in time $\mathcal{O}(n^3)$.*

Our algorithm is similar to, but simpler than, the one we presented for Cycle Segment Cover. It follows a bottom-up dynamic programming scheme along the cotree $\mathcal{T}$ of G. In this bottom-up process, for each node $u \in \mathcal{T}$, we determine a table D_u indexed by the integers from 0 to $|V(u)| - 1$ and that associates to each integer k in this range the number $D_u(k) = \mathsf{sc}^{k+1}(G[V(u)])$ (see Definition 2 for the definition of sc^{k+1}). When the tables D_{u_i} have been computed for all the children u_i of a node u, then the bottom-up process computes the table D_u of u as explained below, depending on whether u is a series node or a parallel node.

5.2.1. Algorithm for series nodes

To treat the case of u being a series node, it is crucial to note that in order to disconnect $G[V(u)]$ by withdrawing a subset $S \subseteq V(u)$ of vertices, it is necessary to include in S the vertices of $V(u_i)$ for all the children u_i of u except one, which we denote v. Moreover, note that for all $k \in \{0, \dots, |V(v)| - 1\}$, we straightforwardly have $D_u(k + \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|) \geq D_v(k) - \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|$.

The algorithm for building D_u uses these two facts as follows. It starts by setting $D_u(k) = -\infty$ for all the values of k in the range of the table, namely $\{0, \dots, |V(u)| - 1\}$. Then, for each child v of u and for each $k \in \{0, \dots, |V(v)| - 1\}$, if $D_u(k + \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|) < D_v(k) - \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|$, then the algorithm sets $D_u(k + \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|)$ to the value $D_v(k) - \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|$.

Let us now show that the values affected to $D_u(j)$ at the end of this algorithm are correct for all j. As we noted previously, we necessarily have $D_u(k + \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|) \geq D_v(k) - \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|$. So the only risk is that the value of $D_u(j)$ at the end of the algorithm is less than what it should be. More explicitly, there could exist a set $S \subseteq V(u)$ such that $c(G[V(u)]-S)-|S| > D_u(|S|)$ at the end of the algorithm. Assume for contradiction that such a set S exists. From our initial remark, S must include the vertices of $V(u_i)$ for all the children u_i of u except one, denoted v. Let $S_v = S \cap V(v)$, then we have $c(G[V(u)] - S) = c(G[V(v)] - S_v)$. This gives $c(G[V(u)] - S) - |S| = c(G[V(v)] - S_v) - |S_v| - \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|$ and since by definition $c(G[V(v)] - S_v) - |S_v| \leq D_v(|S_v|)$, we obtain

$$c(G[V(u)] - S) - |S| \leq D_v(|S_v|) - \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|$$

On the other hand, after the integer $|S_v|$ has been considered when treating child v of u, the algorithm ensures that $D_u(|S_v| + \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|) \geq D_v(|S_v|) - \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|$. As the value of $D_u(|S_v| + \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|) = D_u(|S|)$ only increases during the algorithm, at the end we have $D_u(|S|) \geq D_v(|S_v|) - \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)|$. Together with what precedes, this gives $c(G[V(u)] - S) - |S| \leq D_v(|S_v|) - \sum_{u_i \in \mathcal{C}(u)\setminus\{v\}} |V(u_i)| \leq D_u(|S|)$, which is a contradiction with our assumption. Therefore, the value of $D_u(j)$ at the end of the algorithm is not less than what it should be, it is correct for all $j \in \{0, \dots, |V(u)| - 1\}$.

For each child v of u the algorithm considers $|V(v)|$ values for k, each of them in constant time. Therefore, the total time spent by the algorithm for computing table D_u is $O(\sum_{v \in \mathcal{C}(u)} |V(v)|) = O(|V(u)|)$.

5.2.2. Algorithm for parallel nodes

The algorithm for a parallel node u makes use of a routine named $\texttt{ExtendScat}(D_1, D_2)$. Given the tables D_1 and D_2 of the scattering numbers of two graphs G_1 and G_2, ExtendScat computes the table D of the scattering numbers of the disjoint union G of graphs G_1 and G_2. More explicitly, for any $i \in \{0, \dots, |V(G_\alpha)|\}$, where $\alpha \in \{1, 2\}$, we have $D_\alpha(i) = \mathrm{sc}^{i+1}(G_\alpha)$. D is similarly defined by $D(i) = \mathrm{sc}^{i+1}(G)$ for all $i \in \{0, \dots, |V(G)|\}$.

The routine is very simple, it affects $D(i)$ with the value $\max_{k_1+k_2=i} D_1(k_1) + D_2(k_2)$ by parsing all the couples $(k_1, k_2) \in \{0, \dots, |V(G_1)|\} \times \{0, \dots, |V(G_2)|\}$. The correctness of this is straightforward and the running time of $\texttt{ExtendScat}(D_1, D_2)$ is $O(|V(G_1)||V(G_2)|)$ time.

The whole algorithm for a parallel node u takes as input the tables D_{u_i} of its l children u_i, $1 \leq i \leq l$ and computes

$$\texttt{ExtendScat}(D_{u_1}, \texttt{ExtendScat}(D_{u_2}, \dots, \texttt{ExtendScat}(D_{u_{l-1}}, \texttt{ExtendScat}(D_{u_l},))\dots)).$$

This gives D_u and takes time $O(\sum_{1\leq i \leq l-1}(|V(u_i)| . \sum_{i+1 \leq j \leq l} |V(u_j)|)) = O(|V(u)|^2)$.

5.2.3. The whole algorithm for k-Cyclability

As we already said, the algorithm for determining the largest k such that G is k-cyclable is a bottom-up process along the cotree $\mathcal{T}$ of G. In this process, for each node u of $\mathcal{T}$, we compute the table D_u that is indexed by integers i from 0 to $|V(u)| - 1$ and such that $D_u(i) = \mathrm{sc}^{i+1}(G[V(u)])$. Then, at the end of the algorithm, when the table D_r has been computed for the root r of $\mathcal{T}$, the largest k such that G is k-cyclable is simply the smallest k such that $D_r(k) > 0$, because $D_r(k) = \mathrm{sc}^{k+1}(G) > 0$ and $D_r(k-1) = \mathrm{sc}^k(G) \leq 0$.

The algorithm starts by computing the table D_l for all the leaves l of $\mathcal{T}$, which is the table having only one cell indexed 0 and containing the value $D_l(0) = 1$. Once the tables D_{u_i} have been computed for all the children u_i of u, then the table of u itself is determined by either using the algorithm for parallel nodes or for series nodes (both described above), depending of whether u is a series node or a parallel node. This process ends with the computation of the table of the root. As the time needed to process a series node u is $O(|V(u)|)$ and the time needed to process a parallel node is $O(|V(u)|^2)$, the execution of the whole algorithm takes time $O(\sum_{u \text{ series}} |V(u)| + \sum_{u \text{ parallel}} |V(u)|^2) = O(n^2) + O(n^3) = O(n^3)$.

Final remarks. In this algorithm, we simply output the largest k such that G is k-cyclable without providing any certificate for it. Nevertheless, note that we could augment the algorithm to provide a certificate that G is not $(k+1)$-cyclable. To this purpose, it is enough to exhibit a separator S of size k such that $c(G-S) - |S| \geq k+1$ and modifying the algorithm to do so is not difficult. However, providing a certificate that prove that all subsets T of terminals of size at most k are cyclable is a real challenge as, to the best of our knowledge, it is not even known whether there exists such a certificate of polynomial size in the case where input graphs are restricted to cographs.

Finally, we note that in the case where one is not interested in determining the largest k such that G is k cyclable but instead only wants to know whether G is k cyclable for some given k small compared to n, then one can adapt our algorithm by limiting the size of the tables D_u to be indexed from 0 to $k-1$. Doing this, the time complexity of processing a series node u becomes $O(k|\mathcal{C}(u)|)$ and the complexity for a parallel node becomes $O(k^2|\mathcal{C}(u)|)$. Consequently, the complexity of the whole algorithm expresses as $O(k^2 n)$ which is a significant improvement when k is much smaller than n.

6. Conclusions

We considered Terminal Cyclability together with its generalization Cycle Segment Cover and k-Cyclability on graph classes. We showed that Cycle Segment Cover on interval graphs, bipartite permutation graphs and cographs can be solved in linear time. Moreover, we obtained certifying algorithms for these problems that either produce a solution, that is, a cycle or a family of paths that cover the set of terminal vertices T or output a T-separator S^* that certifies a no-answer. We then used these results to show that k-Cyclability can be solved in polynomial time when restricted to these graph classes.

A natural open question is to consider the aforementioned problem for other graph classes. In particular, what can be said about the class of cocomparability graphs (see [6,22] for the formal definition and properties of this class)? For instance, it is proved by Deogun, Kratsch and Steiner [15] that a cocomparability graph G with at least three vertices has a Hamiltonian cycle if and only if $\mathsf{sc}(G) \leq 0$. They also proved that the set of vertices of G can be covered by at most k vertex-disjoint paths if and only if $\mathsf{sc}(G) \leq k$. This indicates that the class of cocomparability graphs is a natural candidate for Cycle Segment Cover and k-Cyclability. Still, we do not see how to extend the results of [15] to our settings.

Another interesting question is about the complexity of k-Cyclability. It is easy to see that the problem is in Π_2^{P}. Golovach et al. conjectured in [21] that k-Cyclability is Π_2^{P}-complete. The conjecture is still open.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] T. Akiyama, T. Nishizeki, N. Saito, NP-completeness of the Hamiltonian cycle problem for bipartite graphs, J. Inform. Process 3 (2) (1980/81) 73–76.

[2] E. Alkassar, S. Böhme, K. Mehlhorn, C. Rizkallah, A framework for the verification of certifying computations, J. Autom. Reason. 52 (3) (2014) 241–273, http://dx.doi.org/10.1007/s10817-013-9289-2.

[3] A. Björklund, T. Husfeldt, N. Taslaman, Shortest cycle through specified elements, in: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January (2012) 17-19, SIAM, 2012, pp. 1747–1753.

[4] K.S. Booth, G.S. Lueker, Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms, J. Comput. Syst. Sci. 13 (3) (1976) 335–379, http://dx.doi.org/10.1016/S0022-0000(76)80045-1.

[5] A. Brandstädt, D. Kratsch, On the restriction of some np-complete graph problems to permutation graphs, in: Fundamentals of Computation Theory, FCT '85, Cottbus, GDR, September (1985) 9-13, in: Lecture Notes in Computer Science, vol. 199, Springer, 1985, pp. 53–62, http://dx.doi.org/10.1007/BFb0028791.

[6] A. Brandstadt, V.B. Le, J.P. Spinrad, Graph Classes: A Survey, SIAM Monographs on Discrete Mathematics and Applications, Society for Industrial and Applied Mathematics (SIAM, Philadelphia, PA, 1999, http://dx.doi.org/10.1137/1.9780898719796.

[7] H. Broersma, J. Fiala, P.A. Golovach, T. Kaiser, D. Paulusma, A. Proskurowski, Linear-time algorithms for scattering number and hamilton-connectivity of interval graphs, J. Graph Theory 79 (4) (2015) 282–299, http://dx.doi.org/10.1002/jgt.21832.

[8] M. Chang, S. Peng, J. Liaw, Deferred-query: An efficient approach for some problems on interval graphs, Networks 34 (1) (1999) 1–10, http://dx.doi.org/10.1002/(SICI)1097-0037(199908)34:1<1::AID-NET1>3.0.CO;2-C.

[9] V. Chvátal, Tough graphs and hamiltonian circuits, Discrete Math. 5 (3) (1973) 215–228, http://dx.doi.org/10.1016/0012-365X(73)90138-6.

[10] D.G. Corneil, B. Dalton, M. Habib, Ldfs-based certifying algorithm for the minimum path cover problem on cocomparability graphs, SIAM J. Comput. 42 (3) (2013) 792–807, http://dx.doi.org/10.1137/11083856X.

[11] D.G. Corneil, H. Lerchs, L.S. Burlingham, Complement reducible graphs, Discrete Appl. Math. 3 (3) (1981) 163–174, http://dx.doi.org/10.1016/0166-218X(81)90013-5.

[12] M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, Parameterized Algorithms, Springer, 2015, http://dx.doi.org/10.1007/978-3-319-21275-3.

[13] P. Damaschke, Paths in interval graphs and circular arc graphs, Discrete Math. 112 (1–3) (1993) 49–64, http://dx.doi.org/10.1016/0012-365X(93)90223-G.

[14] P. Damaschke, J.S. Deogun, D. Kratsch, G. Steiner, Finding hamiltonian paths in cocomparability graphs using the bump number algorithm, Order 8 (4) (1991) 383–391.

[15] J.S. Deogun, D. Kratsch, G. Steiner, 1-tough cocomparability graphs are hamiltonian, Discrete Math. 170 (1–3) (1997) 99–106, http://dx.doi.org/10.1016/0012-365X(95)00359-5.

[16] J.S. Deogun, G. Steiner, Polynomial algorithms for hamiltonian cycle in cocomparability graphs, SIAM J. Comput. 23 (3) (1994) 520–552, http://dx.doi.org/10.1137/S0097539791200375.

[17] R. Diestel, Graph Theory, fourth ed., in: Graduate texts in mathematics, vol. 173, Springer, 2012.

[18] G.A. Dirac, In abstrakten graphen vorhandene vollständige 4-graphen und ihre unterteilungen, Math. Nachr. 22 (1960) 61–85, http://dx.doi.org/10.1002/mana.19600220107.

[19] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[20] M.R. Garey, D.S. Johnson, R.E. Tarjan, The planar hamiltonian circuit problem is np-complete, SIAM J. Comput. 5 (4) (1976) 704–714, http://dx.doi.org/10.1137/0205049.

[21] P.A. Golovach, M. Kaminski, S. Maniatis, D.M. Thilikos, The parameterized complexity of graph cyclability, SIAM J. Discrete Math. 31 (1) (2017) 511–541, http://dx.doi.org/10.1137/141000014.

[22] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Vol. 57, Elsevier, 2004.

[23] R.J. Gould, A look at cycles containing specified elements of a graph, Discrete Math. 309 (21) (2009) 6299–6311, http://dx.doi.org/10.1016/j.disc.2008.04.017.

[24] J.E. Hopcroft, R.E. Tarjan, Efficient algorithms for graph manipulation [H] (algorithm 447), Commun. ACM 16 (6) (1973) 372–378, http://dx.doi.org/10.1145/362248.362272.

[25] S. Hsieh, C. Ho, T. Hsu, M. Ko, The hamiltonian problem on distance-hereditary graphs, Discrete Appl. Math. 154 (3) (2006) 508–524, http://dx.doi.org/10.1016/j.dam.2005.07.012.

[26] R. Hung, M. Chang, Linear-time algorithms for the hamiltonian problems on distance-hereditary graphs,, Theor. Comput. Sci. 341 (1–3) (2005) 411–440, http://dx.doi.org/10.1016/j.tcs.2005.04.009.

[27] R. Hung, M. Chang, Linear-time certifying algorithms for the path cover and hamiltonian cycle problems on interval graphs, Appl. Math. Lett. 24 (5) (2011) 648–652, http://dx.doi.org/10.1016/j.aml.2010.11.030.

[28] H.A. Jung, On a class of posets and the corresponding comparability graphs, J. Comb. Theory, Ser. B 24 (2) (1978) 125–133, http://dx.doi.org/10.1016/0095-8956(78)90013-8.

[29] J.M. Keil, Finding hamiltonian circuits in interval graphs, Inf. Process. Lett. 20 (4) (1985) 201–206, http://dx.doi.org/10.1016/0020-0190(85)90050-X.

[30] Y.D. Liang, N. Blum, Circular convex bipartite graphs: Maximum matching and Hamiltonian circuits, Inf. Process. Lett. 56 (4) (1995) 215–219, http://dx.doi.org/10.1016/0020-0190(95)00145-3.

[31] G.K. Manacher, T.A. Mankus, C.J. Smith, An optimum $\Theta(n \log n)$ algorithm for finding a canonical hamiltonian path and a canonical hamiltonian circuit in a set of intervals, Inf. Process. Lett. 35 (4) (1990) 205–211, http://dx.doi.org/10.1016/0020-0190(90)90025-S.

[32] R.M. McConnell, K. Mehlhorn, S. Näher, P. Schweitzer, Certifying algorithms, Comput. Sci. Rev. 5 (2) (2011) 119–161, http://dx.doi.org/10.1016/j.cosrev.2010.09.009.

[33] H. Müller, Hamiltonian circuits in chordal bipartite graphs, Discrete Math. 156 (1–3) (1996) 291–298, http://dx.doi.org/10.1016/0012-365X(95)00057-4.

[34] N. Robertson, P.D. Seymour, Graph minors. XIII. The disjoint paths problem, J. Comb. Theory, Ser. B 63 (1) (1995) 65–110, http://dx.doi.org/10.1006/jctb.1995.1006.

[35] J.P. Spinrad, A. Brandstädt, L. Stewart, Bipartite permutation graphs, Discrete Appl. Math. 18 (3) (1987) 279–292, http://dx.doi.org/10.1016/S0166-218X(87)80003-3.