Induced Disjoint Paths in Circular-Arc Graphs in Linear Time ⋆

Petr A. Golovach[1], Daniël Paulusma[2], and Erik Jan van Leeuwen[3]

[1] Department of informatics, University of Bergen, Norway,
petr.golovach@ii.uib.no
[2] School of Engineering and Computer Science, Durham University, UK,
daniel.paulusma@durham.ac.uk
[3] Max-Planck Institut für Informatik, Saarbrücken, Germany,
erikjan@mpi-inf.mpg.de

Abstract. The Induced Disjoint Paths problem is to test whether an graph G on n vertices with k distinct pairs of vertices (s_i, t_i) contains paths $P_1, \ldots, P_k$ such that P_i connects s_i and t_i for $i = 1, \ldots, k$, and P_i and P_j have neither common vertices nor adjacent vertices (except perhaps their ends) for $1 \leq i < j \leq k$. We present a linear-time algorithm that solves Induced Disjoint Paths and finds the corresponding paths (if they exist) on circular-arc graphs. For interval graphs, we exhibit a linear-time algorithm for the generalization of Induced Disjoint Paths where the pairs (s_i, t_i) are not necessarily distinct. In both cases, if a representation of the graph is given, then the algorithms run in $O(n+k)$ time.

1 Introduction

A classic algorithmic problem on a graph G with k distinct pairs of vertices (s_i, t_i) is to find vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i and t_i for $i = 1, \ldots k$. Known as the Disjoint Paths problem, it is NP-complete on general graphs [16], but can be solved in $O(n^3)$ time for any fixed integer k [25] (that is, it is fixed-parameter tractable). The Induced Disjoint Paths problem also takes as input a graph G with k distinct pairs of vertices (s_i, t_i) and also asks whether there are paths $P_1, \ldots, P_k$ such that P_i connects s_i and t_i for $i = 1, \ldots, k$, but with the extra condition that $P_1, \ldots, P_k$ must be *mutually induced*, that is, no two paths P_i, P_j have common or adjacent vertices (except perhaps their end-vertices). Notice that the Disjoint Paths problem can be reduced to Induced Disjoint Paths by subdividing every edge of the graph. The Induced Disjoint Paths problem is NP-complete even for instances with $k = 2$ [2, 5], and thus in particular is not fixed-parameter tractable unless P=NP.

⋆ This work is supported by EPSRC (EP/K025090/1) and Royal Society (JP100692). The research leading to these results has also received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 267959. A preliminary version of this paper appeared as an extended abstract in the proceedings of WG 2014 [10].

The hardness of both Disjoint Paths and Induced Disjoint Paths on general graphs inspired research on their complexity on structured graph classes. On the negative side, Disjoint Paths remains NP-complete on line graphs [20] and split graphs [14], and Induced Disjoint Paths remains NP-complete on claw-free graphs [6] (in fact, even on line graphs). Both problems remain NP-complete on planar graphs [19, 8]. In these cases, however, fixed-parameter algorithms are known [9, 14, 17, 24, 25]. On the positive side, polynomial-time algorithms for Disjoint Paths exist on graphs of bounded treewidth [23] and graphs of clique-width at most 2 [12], and for Induced Disjoint Paths on AT-free graphs [8] and chordal graphs [1].

We focus on the complexity of Induced Disjoint Paths on circular-arc graphs. A *circular-arc* graph is a graph that has a *representation* in which each vertex corresponds to an arc of a circle, and two vertices are adjacent if and only if their corresponding arcs intersect. Circular-arc graphs generalize *interval graphs*, which have a representation in which each vertex corresponds to an interval of the line, and two vertices are adjacent if and only if their corresponding intervals intersect. The complexity of Disjoint Paths is known: it is NP-complete on interval graphs [22]. In contrast, for Induced Disjoint Paths, the authors of the present work recently showed a polynomial-time algorithm on circular-arc graphs [9] (for a weaker problem variant, such an algorithm is also implied by a general framework [7]). This work, as well as the polynomial-time algorithms on AT-free graphs [8] and chordal graphs [1], imply a polynomial-time algorithm on interval graphs. These algorithms do not settle the complexity of Induced Disjoint Paths on circular-arc graphs (and interval graphs) completely, as the question whether a linear-time algorithm exists is left open.

In this paper, we exhibit a linear-time algorithm for Induced Disjoint Paths on circular-arc graphs. This improves on the known algorithm for circular-arc graphs as well as the known algorithms for interval graphs. We also introduce a generalization of Induced Disjoint Paths called Requirement Induced Disjoint Paths, which is to find r_i paths that connect s_i and t_i for $i = 1, \ldots, k$, such that all paths are mutually induced. We present a linear-time algorithm for Requirement Induced Disjoint Paths on interval graphs. In both cases, if a representation of the graph is given and the graph has n vertices, then the algorithms run in $O(n + k)$ time.

Our two new algorithms first preprocesses the instance. Some of the preprocessing rules build on our earlier work on Induced Disjoint Paths [8, 9], but care is required to adapt them for Requirement Induced Disjoint Paths and to execute them in $O(n + k)$ time on a graph on n vertices with k terminal pairs. Hence, most of our preprocessing rules are novel. After the preprocessing stage, the algorithms identify a set of candidate paths for each pair (s_i, t_i). For each candidate path for a pair (s_i, t_i), we add an arc with color i that corresponds to the path of an auxiliary graph H. Finally, we show that it suffices to find an independent set in H that contains r_i arcs of each color. We show that the algorithms perform all stages in $O(n + k)$ time.

2 Preliminaries

We only consider finite undirected graphs that have no loops and no multiple edges. We refer to the textbook of Diestel [4] for any standard graph terminology not defined here. Let $G = (V, E)$ be a graph. For a set $S \subseteq V$, the graph $G[S]$ denotes the subgraph of G *induced by* S, that is, the graph with vertex set S and edge set $\{uv \in E \mid u, v \in S\}$. We write $G - S = G[V \setminus S]$. The (open) neighborhood and closed neighborhood of a vertex u are denoted by $N_G(u) = \{v \mid uv \in E\}$ and $N_G[u] = N_G(u) \cup \{u\}$, respectively. The open and closed neighborhood of a set $U \subseteq V$ are denoted by $N_G(U) = \{v \in V \setminus U \mid uv \in E \text{ for some } u \in U\}$ and $N_G[U] = U \cup N_G(U)$, respectively. We denote the degree of a vertex u by $\deg_G(u) = |N_G(u)|$.

We denote an unordered pair of elements x, y by $\{x, y\}$ (i.e. $\{x, y\} = \{y, x\}$).

Problem Definition Let $P = v_1 \cdots v_r$ be a path (we call such a path a v_1v_r-*path*). The vertices v_1 and v_r are the *ends* or *end-vertices* of P, and the vertices $v_2, \ldots, v_{r-1}$ are the *inner vertices* of P. We say that an edge v_iv_j, $i+1 < j$, is an *inner chord* of P if v_i or v_j is an inner vertex of P. Distinct paths $P_1, \ldots, P_\ell$ in a graph G are *mutually induced* if:

(i) each P_i has no inner chords;
(ii) any distinct P_i, P_j may only share vertices that are ends of both paths;
(iii) no inner vertex u of any P_i is adjacent to a vertex v of some P_j for $j \neq i$, except when v is an end-vertex of both P_i and P_j.

Notice that condition (i) may be assumed without loss of generality. This definition is more general than the definition in Section 1, as it allows the end-vertices of distinct paths to be the same or adjacent.

We are now able to formally state our decision problem (where a *terminal* is some specified vertex).

Requirement Induced Disjoint Paths	
Instance:	a graph G, k pairs of distinct terminals $(s_1, t_1), \ldots, (s_k, t_k)$ such that $\{s_i, t_i\} \neq \{s_j, t_j\}$ for $0 \leq i < j \leq k$, and k positive integers $r_1, \ldots, r_k$.
Question:	does G have $\ell = r_1 + \ldots + r_k$ mutually induced paths $P_1, \ldots, P_\ell$ such that exactly r_i of these paths join s_i and t_i for $1 \leq i \leq k$?

If $r_1 = \ldots = r_k = 1$, then the problem is called Induced Disjoint Paths. The paths $P_1, \ldots, P_\ell$ are said to form a *solution* for a given instance, and we call every such path a *solution path*.

The problem definition allows a vertex v to be a terminal in two or more pairs (s_i, t_i) and (s_j, t_j). For instance, $v = s_i = s_j$ is possible. This corresponds to property (ii) of our definition of "being mutually induced". In order to avoid any confusion, we will view s_i and s_j as two different terminals "placed on" vertex v. Formally, we call v a *terminal vertex* that *represents* a terminal s_i or t_i if $v = s_i$

or $v = t_i$, respectively. We let T_v denote the set of terminals represented by v. If $T_v = \emptyset$, we call v a *non-terminal* vertex. We say that the two terminals s_i and t_i of a terminal pair (s_i, t_i) are *partners* of each other. If s_i is represented by u and t_i by v, then we also call a uv-path an $s_i t_i$-path. By our problem definition, each terminal pair (s_i, t_i) consists of two distinct terminals. Hence, two partners are never represented by the same vertex.

By Property (i), each solution path P has no inner chords and P is an induced path if and only if its ends are non-adjacent. If two adjacent vertices u and v represent terminals belonging to the same pair (s_i, t_i), then the path uv is called a *terminal path* for s_i, t_i. We need the following observation.

Observation 1 *Any yes-instance of* Requirement Induced Disjoint Paths *has a solution that contains all possible terminal paths. In particular, a terminal path for a pair (s_i, t_i) is the unique $s_i t_i$-path in this solution if $r_i = 1$.*

Graph Classes Recall the definition of circular-arc and interval graphs from the introduction. Both graph types can be recognized in linear time and a corresponding representation can be found in linear time:

Theorem 1 ([3], see also [13, 18]). *An interval graph G on n vertices and m edges can be recognized in $O(n + m)$ time. In the same time, a representation of G can be constructed with interval end-points $1, \ldots, 2n$.*

The first linear-time recognition algorithm for circular-arc graphs was given by McConnell [21] (see also [15]).

Theorem 2 ([21]). *A circular-arc graph G on n vertices and m edges can be recognized in $O(n + m)$ time. In the same time, a representation of G can be constructed with arc end-points clockwise enumerated as $1, \ldots, 2n$.*

By Theorems 1 and 2, we always assume that an interval or circular-arc graph is given both by its adjacency list and its representation. Moreover, we assume that all the end-points of the intervals/arcs in the representation are distinct integers $1, \ldots, 2n$. Notice that using a representation we can check adjacency in $O(1)$ time. By slight abuse of notation, we often do not distinguish between the vertices and their corresponding intervals/arcs; e.g., we may speak of terminal intervals/arcs instead of terminal vertices.

For a vertex u of an interval graph, l_u and r_u denote the left and right end-point of u, respectively. Note that the degree of u is at least $(r_u - l_u - 1)/2$. For circular-arc graphs, we equate "left" to "counterclockwise" and "right" to "clockwise". Then, in the same way as for interval graphs, we let l_u and r_u denote the left and right end-point of a vertex u, respectively. In this way we are able to define similar terminology for both interval and circular-arc graphs. For two points x, y on the line, we write $x \leq y$ if y lies to the right with respect to x, and $x < y$ if $x \leq y$ and $x \neq y$, and we say that a point z *lies between* points x and y, if $x \leq z \leq y$. If x, y, z are points on a circle we write $x \leq z \leq y$ (or $x \leq z$ and $z \leq y$) to indicate that z is in the interval with the left end-point x

and the right end-point y. We say that a vertex u *lies between* points x and y if $x \leq l_u < r_u \leq y$ (recall that l_u and r_u are distinct integers). Finally, a vertex u *lies between* two other vertices v, w if it lies between r_v and l_w; note that in that case we have in fact that $r_v < l_u < r_u < l_w$ by our assumption that no two end-points in the interval representation are the same.

An *independent set* in a graph G is a set of vertices that are pairwise non-adjacent. At some stage, our algorithm for Induced Disjoint Paths on circular-arc graphs needs to compute a largest independent set of a circular-arc graph. If a representation of the graph is given and the graph has n vertices, then the next result shows that this takes $O(n)$ time.

Theorem 3 ([11]). *If the arc end-points of a circular-arc graph G on n vertices are sorted, then a largest independent set of G can be found in $O(n)$ time.*

3 Interval Graphs

In this section we develop a linear-time algorithm that solves Requirement Induced Disjoint Paths on interval graphs. If we are given a representation of the interval graph and the graph has n vertices and k terminal pairs, then the algorithm actually runs in $O(n + k)$ time.

A possible approach would be the following greedy algorithm: find a terminal vertex with the leftmost right end-point and trace path(s) for the corresponding terminal pairs by a greedy procedure that iteratively chooses the non-terminal vertex with the leftmost right end-point that does not conflict with vertices already chosen. However, we do not elaborate on this approach for two reasons. First, this approach would not be substantially simpler than the approach of our algorithm, as both approaches require a similar (careful) analysis of a high number of corner cases. Second, and more importantly, the goal of this paper is to design a linear-time algorithm for Induced Disjoint Paths on circular-arc graphs, where we have no natural starting point for a similar greedy approach and guessing such a starting point would irrevocably lead to a quadratic-time algorithm.

We describe the main constructs of our algorithm. Consider an instance of Requirement Induced Disjoint Paths. Let P be an $s_i t_i$-path that is not a terminal path, i.e. that has at least one inner vertex. Let I_P be the interval on the line obtained by taking the union of the intervals that correspond to the inner vertices of P. We say that P *covers* the interval I_P. Because P is an $s_i t_i$-path, we say that I_P has color i.

Lemma 1. *Let $P_1, \ldots, P_\ell$ form a solution. The following statements hold:*

i) For $1 \leq i \leq k$, any interval I_{P_a} with color i intersects the intervals that represent s_i and t_i and does not intersect any other interval that represents a terminal;
ii) For $1 \leq a < b \leq \ell$, $I_{P_a} \cap I_{P_b} = \emptyset$;
iii) For $1 \leq i < j \leq k$, there is no interval with color j that lies between two intervals with color i, or vice versa.

Proof. Properties i) and ii) follow immediately from definition. In order to show iii), assume that an interval I_{P_c} with color j lies between two intervals I_{P_a} and I_{P_b}, both with color i, for some i, j with $i \neq j$. Let u and v represent s_i and t_i. By i), I_{P_a} and I_{P_b} each intersect u and v. Then I_{P_c} also intersects u and v. As $i \neq j$, we find that at least one of u, v represents neither s_j nor t_j, contradicting i). $\square$

3.1 An Outline of Our Algorithm

Before giving the precise details of our algorithm we first present an outline. Our algorithm roughly consists of three stages: preprocess the instance, construct an auxiliary graph H, and find an independent set in H.

In Stage 1, we perform eight preprocessing steps. The order in which we execute these steps is crucial for the correctness of our algorithm. In Step 1 we preprocess the instance by deleting non-terminal vertices adjacent to at least three terminal vertices (we will show that these vertices will not be used in any solution). In Step 2 we check if there exists a pair (s_i, t_i) with $r_i \geq 2$ that is represented by two non-adjacent terminal vertices (we will show that our instance is a no-instance if this is the case). In Steps 3–8 we preprocess the instance further and simultaneously start to determine a set of "candidate paths" that might or might not be used in the solution that we are constructing. This set of candidate paths is constructed in such a way that for any s_it_i solution path P there is a candidate path P' such that P' is also an s_it_i-path and $I_{P'} \subseteq I_P$. We will ensure that in the end the set of candidate paths has size $O(n)$.

By Lemma 1 ii), the paths that are selected in a solution must cover distinct parts of the line. Therefore, we create an auxiliary interval graph H that consists of all intervals covered by the candidate paths. We also assign a color to each interval, namely color i if the interval corresponds to a candidate path for a pair (s_i, t_i) (the reason to color these intervals will become clear later). In Step 3 we already start the construction of H by adding to H the interval I_{vuw} with color i for each non-terminal vertex u that is adjacent to terminal vertices v and w representing terminals s_i and t_i with $r_i \geq 2$. Following Observation 1, we also add intervals corresponding to terminal paths to H (this must be done in a careful way, as described in Steps 4–8).

In Stage 2, we finish the construction of H. At the start of Stage 2 we may have already processed a number of terminal pairs completely. For each remaining terminal pair we check three possible situations (Steps 9a–9c). Each of these three situation may apply for a certain terminal pair, and when a certain situation applies we add a number of additional colored intervals to H.

In Stage 3, we essentially search for an independent set in H that contains r_i vertices of color i for $i = 1, \ldots, k$ (Step 10). We will show that such an independent set corresponds to a solution for our instance. Moreover, we will prove that for any yes-instance, our algorithm will indeed continue to Stage 3 and will then find an independent set of H in this stage.

In Section 3.3–3.5, we describe all steps of the algorithm in detail. We say that a step is *safe* if it runs in time $O(n+k)$ and is correct in the following sense:

(i) a `No`-answer is given for no-instances only;
(ii) if a new instance is obtained, then it has a solution if and only if the original instance has a solution.
(iii) if a set of intervals that are all colored with color i is added to H, then this set has size $O(n)$ and corresponds to a *candidate set* of *candidate paths*.

The algorithm assumes that an interval representation of G is known, as given by Theorem 1. As mentioned, it also maintains an auxiliary interval graph H, initially empty. Recall that any vertex that we add to H will correspond to a candidate path for a solution. While adding vertices to H, we maintain an interval representation of H. Finally, the algorithm maintains a set $\mathcal{P}$ of paths, initially empty, which will form a solution for the instance (should it be a yes-instance). We let $T = \{s_1, t_1, \ldots, s_k, t_k\}$ be the set of all terminals. A terminal pair (s_i, t_i) is a *multi-pair* if $r_i \geq 2$, and a *simple pair* otherwise. As mentioned, the algorithm roughly consists of three stages: preprocess, construct H, and find an independent set.

3.2 Basic Tools

Before we describe the stages and preprocessing steps, we define the following basic tools that arise in the implementation of several preprocessing steps. Later we will apply the results in this section amongst others to auxiliary sets of intervals that do not have distinct endpoints. Hence, in the next definition, lemma and corollary we assume that interval end-points may coincide.

Definition 1. *Let C be a set of n intervals of a line with interval end-points in $1, \ldots, 2n$. For a given integer j, a subset $C' \subseteq C$ is* j-close *if for each interval $u \in C'$ it holds that $l_u \leq j$, and either u contains the point j or there is no interval $u' \in C \setminus C'$ that contains a point between r_u (exclusive) and j (inclusive).*

Lemma 2. *Let $c \geq 1$ be an integer, and let C be a set of n intervals of a line with interval end-points in $1, \ldots, 2n$. Then in $O(cn)$ time, we can construct a j-close set C_j^* of $\min\{c, c_j\}$ intervals of C for all $j = 1, \ldots, 2n$, where c_j is the number of intervals u with $l_u \leq j$.*

Proof. We perform a sweep-line algorithm. We start by initializing some data structures. For $j = 1, \ldots, 2n$, let $L_j = \{u \in C \mid l_u = j\}$. The sets L_j can be computed in $O(n)$ total time by for each $u \in C$ putting it into the set it should be in. Finally, initialize an empty deque D.

We now perform the sweep. For each $j = 1, \ldots, 2n$ in order, perform the following action. While there is a $u \in L_j$, remove u from L_j; if D contains less than c elements, then add u to D; otherwise, if the right endpoint of u lies further to the right than the right endpoint of the interval $v \in D$ with the leftmost right endpoint among all intervals in D, then remove v from D and add u instead. If L_j is or becomes empty, then we make the set C_j^* equal to the contents of the deque D.

The correctness of the algorithm is immediate. Since the deque D contains at most c elements at any moment during the execution of the algorithm, the running time of the algorithm is $O(cn)$. $\square$

Corollary 1. *Let X be a set of n intervals of a line with interval end-points in $1, \ldots, 2n$. Let $d \geq 1$ be an integer and let C, D be disjoint subsets of X. Then one can identify in $O(dn)$ time a set of intervals u in D that intersect at least d intervals in C, as well as for each such u a set of d intervals in C that intersect u, or else conclude that no such u exists.*

Proof. A trivial application of Lemma 2 with $c = d$ and C yields in $O(cn) = O(dn)$ time for each $j = 1, \ldots, 2n$ the set C_j^* of (at most) d j-close intervals. Observe that an interval $u \in D$ intersects at least d intervals of C if and only if $|C_{r_u}^*| = d$ and each $v \in C_{r_u}^*$ intersects u. Hence, by a straightforward inspection of each interval in D, we can report all those that are adjacent to at least d intervals of D in $O(dn)$ time. $\square$

3.3 Stage I: Preprocess

The only operations performed on G by our algorithm are vertex deletions. Hence, the graph that we obtain after each step is still interval. For simplicity, we denote this graph by G as well.

Step 1. Delete all non-terminal vertices that are adjacent to at least three terminal vertices.

Lemma 3. *Step 1 is safe.*

Proof. Any internal vertex of a path of a solution is adjacent to at most two terminal vertices, which are the end-vertices of the path. Hence, any non-terminal vertex that is adjacent to at least three terminal vertices cannot be used in any solution. Therefore, Step 1 is correct.

A trivial application of Corollary 1 with $d = 3$, C equal to the set of terminal vertices, and D equal to the set of non-terminal vertices yields all non-terminal vertices that are adjacent to at least three terminal vertices in $O(n)$ time. Then we delete all such non-terminal vertices, which takes $O(n)$ time again. $\square$

Step 2. Check if there is a multi-pair that is represented by two non-adjacent terminal vertices. If so, then return a `No`-answer.

Lemma 4. *Step 2 is safe.*

Proof. Step 2 is correct, because there must exist at least two solution paths between the terminal vertices of a multi-pair. If the two terminal vertices are not adjacent, the union of the vertices of these two paths induces a cycle on at least four vertices in G. This is not possible in an interval graph. Using the list of terminal pairs, Step 2 takes $O(k)$ time. $\square$

Suppose that we have not returned a `No`-answer after performing Step 2. In the next step, for each multi-pair, we identify a set of paths that together with the terminal paths form all candidate paths.

Step 3. For each non-terminal vertex u adjacent to terminal vertices v and w representing multi-pair terminals s_i and t_i, add I_{vuw} with color i to V_H, and delete u from G.

Lemma 5. *Step 3 is safe. Moreover, for any multi-pair (s_i, t_i), if P is a solution s_it_i-path with at least one inner vertex, then there is a candidate s_it_i-path P' with $I_{P'} \subseteq I_P$.*

Proof. We first prove that Step 3 is correct. Let u be a non-terminal vertex adjacent to terminal vertices v and w representing terminals s_i and t_i from a multi-pair (s_i, t_i). By Lemma 3, we find that u is not adjacent to any other terminal vertices. Hence, vuw may be considered as a candidate path for a solution. Moreover, because u is adjacent to both v and w, we deduce the following. Firstly, every s_it_i-path in a solution has at most one inner vertex; otherwise its vertices would induce a cycle on at least four vertices in G, as v, w are adjacent by Step 2. Hence, the set of intervals added to V_H for each multi-pair (s_i, t_i) contains all possible solution paths for (s_i, t_i), and as such corresponds to a candidate set for (s_i, t_i). Secondly, u may not be used in a solution path for a terminal pair (s_j, t_j) with $j \neq i$. Hence, we can safely remove u from G. Because we only added intervals to H that correspond to distinct vertices, we added $O(n)$ vertices to V_H in total.

We now show how to perform Step 3 in $O(n+k)$ time. We create an auxiliary set of intervals X. First, add the intervals of all non-terminal vertices to X. Then, for each pair of terminal vertices v and w representing multi-pair terminals s_i and t_i, create a new interval p_i equal to the intersection of v and w and associate with it the number i; note that v and w are adjacent by Step 2, and therefore the interval is well defined. Observe that $|X| = O(n+k)$. Now apply Corollary 1 to X with $d = 1$, C equal to the set of non-terminal vertices, and D equal to the set of intervals p_i. For each interval reported by the algorithm of Corollary 1, let u be the non-terminal vertex and p_i be the interval of D that intersects it. From p_i, and in particular from i, we derive the corresponding multi-pair, and thus the terminal vertices v and w. Then u is adjacent to v and w, and thus we add I_{vuw} with color i to V_H, and delete u from G. This takes $O(|X|) = O(n+k)$ time in total. The correctness of the algorithm follows from the correctness of Corollary 1, and from the fact that after Step 1, each non-terminal vertex is adjacent to at most one pair of terminal vertices v and w representing a multi-pair. □

In the next two steps, which are inspired by our earlier work on Induced Disjoint Paths [8, 9], we get rid of all adjacent terminal vertices that represent the same terminal pair. This includes (but is not limited to) all multi-pairs.

Step 4. Find the set Z of all terminal vertices v such that v only represents terminals whose partners are in $N_G(v)$. Delete the vertices of Z and all non-terminal vertices of $N_G(Z)$ from G. Delete from T the terminals of all terminal pairs (s_i, t_i) with $s_i \in T_v$ or $t_i \in T_v$ for some $v \in Z$. Put all terminal paths corresponding to deleted terminal pairs in $\mathcal{P}$.

Lemma 6. *Step 4 is safe.*

Proof. We first show that Step 4 is correct. Let $\{s_{i_1}, \ldots, s_{i_p}, t_{j_1}, \ldots, t_{j_q}\}$ be the union of all terminals represented by vertices in Z. By Observation 1, we may assume that each terminal path for (s_{i_a}, t_{i_a}) for $a = 1, \ldots, p$ and each terminal path for (s_{j_b}, t_{j_b}) for $b = 1, \ldots, q$ is in a solution, if our instance is a yes-instance. Hence, we can safely put these terminal paths in $\mathcal{P}$. Moreover, as we already identified a candidate set for all multi-pairs in Step 3, we may safely remove each of the two terminals of every pair (s_{i_a}, t_{i_a}) for $a = 1, \ldots, p$ and every pair (s_{j_b}, t_{j_b}) for $b = 1, \ldots, q$ from T.

Let u be a non-terminal vertex in $N_G(Z)$. Then u is not adjacent to two terminal vertices representing two terminals from a multi-pair, as otherwise we would have removed u in Step 3 already. Moreover, u is not used as an inner vertex of a solution path for a simple terminal pair (s_i, t_i) either, for the following two reasons. Firstly, if s_i or t_i is represented by a vertex in Z, we would use the corresponding terminal path for a solution due to Observation 1. Secondly, if both s_i and t_i are not represented by a vertex in Z, we could still not use u as an inner vertex for an $s_i t_i$-path, as u is adjacent to some terminal vertex in Z.

We now show how to perform Step 4 in $O(n + k)$ time. We “mark” each terminal vertex. Then we go through the list of terminal pairs, and if a pair (s_i, t_i) is not represented by adjacent terminal vertices, then we “unmark” these terminal vertices. The set Z is the set of all “marked” terminal vertices that are left in the end. By using the interval representation, obtaining Z takes $O(k)$ time. A trivial application of Corollary 1 with $d = 1$, C equal to Z, and D equal to the set of non-terminal vertices yields all non-terminal vertices in $N_G(Z)$ in $O(n)$ time. Then we delete all such non-terminal vertices and all vertices of Z, which takes $O(n)$ time again. Finally, we go through the list of terminal pairs, and if a terminal s_i or t_i is in Z, we delete both s_i and t_i from T and add its terminal path to $\mathcal{P}$. This takes $O(k)$ time. We conclude that the total running time of performing Step 4 is $O(n + k)$. □

After Step 4, each terminal vertex represents at least one terminal whose partner is at distance at least 2. There may still be terminal pairs whose terminals are represented by adjacent vertices. We deal with such pairs in the next step.

Step 5. Delete all terminals s_i and t_i represented by adjacent terminal vertices from the terminal list, and delete all common non-terminal neighbors of the terminal vertices that represent s_i and t_i. Put all terminal paths corresponding to deleted terminals in $\mathcal{P}$.

Lemma 7. *Step 5 is safe.*

Proof. We first show that Step 5 is correct. First, we may assume without loss of generality that a solution contains all terminal paths by Observation 1. Hence, we may safely put these terminal paths in $\mathcal{P}$, and delete terminals that are represented by adjacent terminal vertices if (s_i, t_i) is not a multi-pair; if (s_i, t_i) is a multi-pair, then all candidate paths have already been identified in Step 3, and thus s_i and t_i may be deleted as well.

Second, if a solution path contains an inner vertex u adjacent to a terminal vertex v representing a terminal that we remove in Step 5, then the reason is that u belongs to a solution path for a terminal pair (s_j, t_j) where s_j or t_j is represented by v as well (note that v represents at least one terminal whose partner is not represented by a neighbor of v, as otherwise we would have removed v in Step 4). Hence, u is allowed to be adjacent to v by definition, except if u is adjacent to both the terminal vertex that represents s_i and the terminal vertex that represents t_i. Since these common neighbors are removed in Step 5, the latter is not possible though.

We can perform Step 5 in $O(n+k)$ time using a similar idea as in Step 3. We create an auxiliary set of intervals X. First, add the intervals of all non-terminal vertices to X. Then, for each pair of terminals (s_t, t_i) represented by adjacent terminal vertices, create a new interval p_i equal to the intersection of v and w and associate with it the number i; clearly, the interval is well defined. Observe that $|X| = O(n+k)$. Now apply Corollary 1 to X with $d = 1$, C equal to the set of non-terminal vertices, and D equal to the set of intervals p_i. This takes $O(|X|) = O(n+k)$ time. Then we delete each non-terminal vertex whose interval was reported by Corollary 1, which takes $O(n)$ time. The rest of Step 5 can be trivially performed in $O(k)$ time. $\square$

Call a terminal pair *long* if its two terminals are represented by vertices of distance at least 2. After Step 5, all terminal pairs are long. Therefore, by Step 2, there are no multi-pairs anymore. Assume that there are $k' \leq k$ terminal pairs left; note that $k' = 0$ is possible.

Step 6. Check if there exists a terminal vertex that represents three or more terminals. If so, then return a `No`-answer.

Lemma 8. *Step 6 is safe.*

Proof. We first prove that Step 6 is correct. For contradiction, assume that a terminal vertex u represents at least three terminals s_h, s_i, s_j. Due to Step 5, these terminals belong to long pairs. Let v_1, v_2, v_3 denote the terminal vertices that represent t_h, t_i, t_j, respectively. Because u is not adjacent to any of v_1, v_2, v_3, every solution has $s_h t_h$, $s_i t_i$, and $s_j t_j$-paths that each contain at least one inner vertex x_1, x_2, x_3, respectively. Assume without loss of generality that x_1, x_2, x_3 are adjacent to u. The intervals x_1, x_2, x_3 do not intersect each other but they do intersect u. Assume without loss of generality that x_2 lies between x_1 and x_3. Then all the vertices of the $s_i t_i$-path except u lie between x_1 and x_3. Therefore, u and v_2 are adjacent. This contradicts with the fact that the pair (s_j, t_j) is long. Hence, our instance is a no-instance if this situation occurs.

Step 6 can be performed in $O(n+k)$ time by going through the list of terminals and counting how often each terminal vertex occurs. $\square$

After Step 6, a terminal vertex may represent at most two terminals (which by definition must belong to different terminal pairs). We therefore assume from now on that each terminal vertex has associated with it the list of at most two terminal pairs i, i' for which it represents a terminal; these lists take $O(k)$ time to compute in a straightforward manner.

We now observe that terminals should be ordered, and we let our algorithm find this ordering.

Step 7. Check if there exist three terminal vertices u, v, w such that u and w represent terminals from the same pair such that $l_u \leq l_v < l_w$. If so, then return a `No`-answer. Otherwise, order and rename the terminals such that $r_{u_i} < l_{v_i}$ and $l_{v_i} \leq l_{u_{i+1}}$ for $i = 1, \ldots, k' - 1$, where u_i, v_i are the vertices representing s_i, t_i, respectively.

Lemma 9. *Step 7 is safe.*

Proof. We first prove that Step 7 is correct. Suppose that there exist three terminal vertices u, v, w such that u and w represent terminals from the same pair and $l_u \leq l_v < l_w$. Assume that u, v, w represent s_i, s_j, t_i, respectively, and let x represent t_j. Let P_1 and P_2 be the $s_i t_i$-path and $s_j t_j$-path, respectively, in a solution. Because (s_i, t_i) and (s_j, t_j) are long, both P_1 and P_2 contain at least one inner vertex. By Lemma 1, $I_{P_1} \cap I_{P_2} = \emptyset$. However, this is not possible as $l_u \leq l_v < l_w$. Hence, our instance is a no-instance.

We now show how to perform Step 7 in $O(n + k)$ time. Recall that each end-point of an interval is an integer between 1 and $2n$. Construct $2n$ buckets $B_1, \ldots, B_{2n}$. Then go through the list of terminal pairs T and put a terminal in bucket B_{l_u} if u is the vertex of G that represents the terminal. Go through the non-empty buckets among $B_1, \ldots, B_{2n}$ in increasing order and verify whether the partner of a terminal of a terminal pair not seen before is in the next non-empty bucket. Stop and return a `No`-answer if this does not hold. Otherwise, as each bucket contains at most two terminals due to Step 6, this gives the desired ordering of the terminal pairs in $O(n + k)$ time. □

Step 8. For $i \in \{1, \ldots, k' - 1\}$, if t_i and s_{i+1} are represented by distinct vertices v and w, delete all non-terminal vertices adjacent to both v and w.

Lemma 10. *Step 8 is safe.*

Proof. Any non-terminal vertex deleted in Step 8 can never be used as an inner vertex of a solution path by the definition of the Requirement Induced Disjoint Paths problem.

We now show how to perform Step 8 in $O(n)$ time. Apply Corollary 1 with $d = 2$, C equal to the set of terminal vertices, and D equal to the set of non-terminal vertices. For each interval reported by the algorithm of Corollary 1, let u be the non-terminal vertex and v, w be the terminal vertices that it intersects. From v and w we can derive the corresponding terminal pairs, using that each terminal vertex has associated with it the list of at most two terminal pairs i, i' for which it represents a terminal. Then we can straightforwardly determine if u should be deleted. This takes $O(n)$ time in total. □

3.4 Stage II: Construct H

We now construct the auxiliary H. Note that some intervals were already added to H as part of our preprocessing stage (namely in Steps 3–5).

Step 9. For each $i \in \{1, \ldots, k'\}$, perform steps 9a–9d (where v and w are terminal vertices that represent s_i and t_i, respectively).

9a. For every common neighbor u of v and w, add the interval I_{vuw} to H with color i, and delete u from G.

9b. For each neighbor x of v not adjacent to w, determine whether there exists a neighbor y of w adjacent to x. If so, then choose y such that the right end-point of y is leftmost amongst all such neighbors of w. Add the interval I_{vxyw} to H with color i.

9c. Determine the connected components $C_1, \ldots, C_p$ of $G-(N[v] \cup N[w])$ whose vertices lie between r_v and l_w. For each C_j, determine the vertex $l(C_j)$ with the leftmost left end-point and the vertex $r(C_j)$ with the rightmost right end-point. For each neighbor x of v, let C_x be the component among $C_1, \ldots, C_p$ with the rightmost $r_{r(C_j)}$ for which $r_x > l_{l(C_j)}$. Then let y_x be the neighbor of w with the leftmost right endpoint amongst all neighbors y of w with $l_y < r_{r(C_x)}$. If y_x exists, then add the interval between $l(x)$ and $r(y_x)$ to H with color i.

Lemma 11. *Step 9 is safe. Moreover, for $i = 1, \ldots, k'$, if P is a solution $s_i t_i$-path, then there is a candidate $s_i t_i$-path P' with $I_{P'} \subseteq I_P$.*

Proof. We first prove that Step 9 is correct. Let $i \in \{1, \ldots, k'\}$. Let v and w be the (non-adjacent) vertices of G representing s_i and t_i, respectively. Let P be a solution path for (s_i, t_i).

Suppose that P has length 2. Then P has exactly one inner vertex u, which is adjacent to both v and w. By Step 9a, H contains the interval I_P.

Suppose that P has length 3. Then P has exactly two inner vertices x and y' that are adjacent to v and w, respectively. Let y be the neighbor of w that is adjacent to x and has the leftmost right end-point among all such vertices. Then $P' = vxyw$ is an $s_i t_i$-path. Notice that $I_{P'} \subseteq I_P$ by the choice of y and by the fact that u and v have no common neighbors after Step 9a. Therefore, in any solution that contains P, we can replace P by P'. By Step 9b, H contains $I_{P'}$.

Finally, suppose that P has length at least 4. Because P is an induced path, there is a connected component C_j of $G - (N[v] \cup N[w])$ whose vertices all lie between r_v and l_w, such that all inner vertices of P except two neighbors of v and w are in C_j. Let x and y be the neighbors of v and w on P, respectively. Since P is an induced path, x and y are not adjacent, and thus $C_j = C_x$ by definition and the fact that $I_{C_1}, \ldots, I_{C_p}$ are pairwise disjoint. Then from P we can construct an $s_i t_i$-path P' by replacing y with y_x. Notice that $I_{P'} \subseteq I_P$ by the choice of y_x and by the fact that v and w have no common neighbors after Step 9a. Therefore, in any solution that contains P, P can be replaced P'. By Step 9c, H contains $I_{P'}$.

Observe that the above arguments prove that for $i = 1, \dots, k'$, if P is a solution s_it_i-path, then there is a candidate s_it_i-path P' with $I_{P'} \subseteq I_P$.

We now show how to perform Step 9 in $O(n+k)$ time. It is important to note that we can consider each terminal pair separately during Step 9, as each non-terminal vertex will be involved in the actions of Step 9 for at most two terminal pairs. This is because a non-terminal vertex u is involved for the terminal pair (s_i, t_i), represented by terminal vertices v and w, only if u is adjacent to v or w, or if u lies between v and w. By Step 1, Step 6, and Step 8, each non-terminal vertex is adjacent either to one terminal vertex that represents at most two terminals or to two terminal vertices that represent the terminals of the same terminal pair. Hence, if there are $n_i - 2$ non-terminal vertices involved with terminal pair (s_i, t_i) and we spend $O(n_i)$ time for each i, then the whole algorithm runs in $\sum_{i=1}^{k'} O(n_i) = O(n + k)$ time.

So assume that there are two fixed terminal vertices v and w representing the terminal pair (s_i, t_i), and assume that all $n_i - 2$ other (non-terminal) vertices are either adjacent to v or w, or lie between v and w. Moreover, a straightforward modification enables us to assume that the representation has interval end-points $1, \dots, 2n_i$. We now show that Steps 9a, 9b, and 9c can indeed be performed in $O(n_i)$ time, and $O(n_i)$ intervals are added to H.

For Step 9a, we simply evaluate all non-terminal vertices u whether they are adjacent to both v and w. Using the interval representation, adjacency can be tested in $O(1)$ time. Therefore, it takes $O(n_i)$ time to evaluate all non-terminal vertices u, and possibly add I_{vuw} to H and delete u. Clearly, $O(n_i)$ intervals are added to H.

For Step 9b, we perform a sweep-line algorithm. Let $X = N_G(v)$ and $Y = N_G(w)$. After Step 9a, $X \cap Y = \emptyset$. For any $j \in \{1, \dots, 2n_i\}$, let $L_j^Y = \{u \in Y \mid l_u = j\}$ and let $R_j^X = \{u \in X \mid r_u = j\}$. By assumption, $|L_j^Y| + |R_j^X| \leq 1$ for each $j \in \{1, \dots, 2n_i\}$. Let y be a non-terminal vertex, which initially points nowhere and which we denote $y = \perp$. We now perform the sweep. For each $j = 1, \dots, 2n_i$ in order, we perform the following actions in order. If there is a $u \in L_j^Y$, and either $y = \perp$ or $r_u < r_y$, then let $y = u$. If there is a $x \in R_j^X$ and $y \neq \perp$, then add the interval I_{vxyw} to H with color i. The vertex y that is maintained ensures that for each $x \in X$, we know the vertex that is adjacent to x and w with the leftmost right end-point (if it exists). It is clear that the sweep-line algorithm takes $O(n_i)$ time and that $O(n_i)$ intervals are added to H.

For Step 9c, we perform several sweep-line algorithms. In the first sweep, we determine the connected components $C_1, \dots, C_p$ and their vertices with leftmost left end-point and rightmost right end-point. Let Z denote the non-terminal vertices that are not adjacent to v nor w; this set can straightforwardly be determined in $O(n_i)$ time, using the interval representation to test adjacency in $O(1)$ time. For any $j \in \{1, \dots, 2n_i\}$, let $L_j^Z = \{u \in Z \mid l_u = j\}$. By assumption, $|L_j^Z| \leq 1$ for each $j \in \{1, \dots, 2n_i\}$. Let s, t be non-terminal vertices, which initially point nowhere and which we denote $s = t = \perp$. Initialize an integer $q = 0$. Also, for each $j \in \{1, \dots, 2n_i\}$, initialize a table $A[j]$, which will store for which component $C \in \{C_1, \dots, C_p\}$ the singleton set that contains the interval

I_C is j-close. For each $j = 1, \dots, 2n_i$ in order, we perform the following actions in order. If there is a $u \in L_j^Z$, then do the following:

- if $t \neq \perp$, u is adjacent to t, and $r_u > r_t$, then let $t = u$;
- if $t \neq \perp$ and u is not adjacent to t, then report a new interval c_q from l_s to r_t, increase q by 1, and let $s = t = u$;
- if $t = \perp$, then let $s = t = u$ and increase q by 1.

Finally, set $A[j] = q$.

This algorithm reports in $O(n_i)$ time a set $O(n_i)$ of intervals that correspond to $I_{C_1}, \dots, I_{C_p}$ for the components $C_1, \dots, C_p$ respectively; denote this set by $\mathcal{C} = \{c_q \mid q \in \{1, \dots, p\}\}$.

We now "combine" the intervals of $\mathcal{C}$ and the intervals of $N_G(v)$. Let $X = N_G(v)$. For each interval $x \in X$ such that $A[r_x] \neq 0$, create a "combined interval" $a_x^{A[r_x]}$ from l_x to $r_{c_{A[r_x]}}$. We then perform the same algorithm as in Step 9b, where instead of $X = N_G(v)$ we use the set of combined intervals as X. This enables us to perform Step 9c in $O(n_i)$ time, and add $O(n_i)$ intervals to H. □

3.5 Stage III: Find Independent Set

It remains to find a particular independent set in H.

Step 10. Find an independent set in H that, for $i = 1, \dots, k$, contains exactly $r_i - 1$ or r_i vertices colored i depending on whether (s_i, t_i) is a multi-pair or not. If such a set exists, add the corresponding candidate paths to $\mathcal{P}$ and return $\mathcal{P}$. Otherwise, return a `No`-answer.

Lemma 12. *Step 10 is safe.*

Proof. We first prove that Step 10 is correct. We do this by proving that our instance is a yes-instance if and only if H has an independent set as described in Step 10. First, suppose that H has such an independent set $\mathcal{I}$. For each interval u of color i, we can find an s_it_i-path in G with inner vertices that are used to construct u. Taking into account the terminal paths that are already included in $\mathcal{P}$, we obtain r_i s_it_i-paths for each $i \in \{1, \dots, k\}$. We have to show that these paths are mutually induced. Because $\mathcal{I}$ is an independent set, distinct paths have no adjacent inner vertices. It remains to show that each $u \in \mathcal{I}$ does not intersect any terminal vertex (interval) of G except the vertices representing s_i and t_i. If u is added to H in Step 3, then this follows immediately from the fact that all non-terminal vertices that are adjacent to at least three terminals are deleted in Step 1 and from the description of Step 3. If u is added to H in Step 9, then u does not intersect any terminal vertex deleted in Step 4, because we delete such terminal vertices together with adjacent non-terminal vertices. Similarly, u does not interfere with any terminal deleted in Step 5, as proved in Lemma 7. Moreover, each interval added in Step 9 intersects exactly two remaining terminal vertices that are partners by Step 8. Hence, the instance is a yes-instance.

Now suppose that our instance is a yes-instance. Let $\ell_i = r_i - 1$ if (s_i, t_i) is a multi-pair, and let $\ell_i = r_i$ otherwise. By Observation 1, we can assume that the solution includes all terminal paths. Therefore, the solution contains exactly ℓ_i $s_i t_i$-paths with inner vertices. By Lemmas 5 and 11, for each such solution $s_i t_i$-path P, there is a candidate $s_i t_i$-path P' such that $I_{P'} \subseteq I_P$. Therefore, we can replace each solution path by a candidate path to obtain a solution that only uses candidate paths. Let $\mathcal{I}$ denote the set of intervals covered by these paths. By Lemma 1, the intervals of $\mathcal{I}$ do not intersect each other. Moreover, by construction, $\mathcal{I}$ contains ℓ_i intervals with color i. Hence these intervals correspond to an independent set of H that has the required properties.

We now show how to perform Step 10 in $O(n)$ time. We do this by performing the following procedure, which is a modification of the well-known greedy algorithm for finding a largest independent set in an interval graph.

1. Construct $2n$ buckets $L_1, \ldots, L_{2n}$ and $2n$ buckets $R_1, \ldots, R_{2n}$.

2. For each vertex u of H, put u in buckets L_{l_u} and R_{r_u}.

3. Set $\mathcal{I} = \emptyset$ and $h = 2n$. For $i = 1, \ldots, k$, set $\ell_i = r_i - 1$ if (s_i, t_i) is a multi-pair, and set $\ell_i = r_i$ otherwise.

4. Scan the buckets $L_h, \ldots, L_1$ until we find a bucket L_j that contains a vertex u of H of some color i such that $\ell_i > 0$. Then u is included in $\mathcal{I}$. Find the set of vertices X from the buckets $R_j, \ldots, R_h$, and delete all vertices of X from H. Then set $\ell_i = \ell_i - 1$, $h = j$, and repeat the procedure. We stop as soon as we cannot find the next bucket L_j.

If $\mathcal{I}$ contains less than ℓ_i vertices of color i for some $i \in \{1, \ldots, k\}$, then stop and return a `No`-answer. Otherwise, return $\mathcal{I}$. This procedure takes $O(|V(H)|) = O(n)$ time, and the corresponding paths can be found in $O(n)$ time (by assuming that all intervals are annotated with their corresponding paths of at most four vertices, or in the case of the intervals generated by Step 9c, by the four vertices and component number that generated the interval). Hence, it remains to show that the procedure is correct. We need the following claim, which implies that between the left endpoints of two intervals with a color i there can be no left endpoint of an interval with color $j \neq i$.

Claim 1. *Let U_i, U_j be the set of vertices (intervals) of H colored by distinct colors i and j respectively. Then for any $u \in U_i$ and $v \in U_j$, $l_u \neq l_v$. Moreover, if $l_u < l_v$ for some $u \in U_i$ and $v \in U_j$, then $l_x < l_y$ for any $x \in U_i$ and $y \in U_j$.*

Proof: Let $u \in U_i$ and $v \in U_j$. Suppose that u and v are added to H in Step 3 of the algorithm. Then $l_u \neq l_v$, because u and v are distinct vertices of G. We assume without loss of generality that $l_u < l_v$. Note that the intervals of U_i correspond to the non-terminal vertices of G that are adjacent to two adjacent terminal vertices w_1, z_1 of G representing s_i, t_i and that are not adjacent to other terminal vertices by Steps 1 and 3. Similarly, the intervals of U_j correspond to the non-terminal vertices of G that are adjacent to two adjacent terminal vertices w_2, z_2 of G representing s_j, t_j and that are not adjacent to other terminal vertices. Consider the interval $I = w_1 \cap z_1$. Because $l_u < l_v$, the left end-point of any $x \in U_i$ lies to the left of the right end-point of I and the left end-point of any $y \in U_j$ lies to the right of the right end-point of I. Hence, $l_x < l_y$ for any $x \in U_i$ and $y \in U_j$.

Suppose now that u is added to H in Step 3 and v is added to H in Step 9. The intervals of U_i correspond to the non-terminal vertices of G that are adjacent to two adjacent terminal vertices w_1, z_1 of G representing s_i, t_i and that are not adjacent to other terminal vertices. The intervals of U_j are the unions of non-terminal vertices of G and these intervals intersect two non-adjacent terminal intervals w_2, z_2 of G representing s_j, t_j. Observe that the intervals of U_i could not be used for construction of the intervals of U_j because all non-terminal vertices that are adjacent to w_1, z_1 are deleted in Steps 4 and 8. Moreover, the intervals of U_j do not intersect any terminal vertex of G except w_2, z_2. Hence, $l_u \neq l_v$. Consider the interval $I = w_1 \cap z_1$. Without loss of generality, $l_u < l_v$. Then the left end-point of any $x \in U_i$ lies to the left of the right end-point of I and the left end-point of any $y \in U_j$ lies to the right of the right end-point of I. Hence, $l_x < l_y$ for any $x \in U_i$ and $y \in U_j$.

Finally, suppose that u and v are added to H in Step 9 of the algorithm. The intervals of U_i intersect two non-adjacent terminal intervals w_1, z_1 of G representing s_i, t_i and they do not intersect other terminal vertices of G, and the intervals of U_j intersect two non-adjacent terminal intervals w_2, z_2 of G representing s_j, t_j and they do not intersect other terminal vertices of G. Recall that the terminals are ordered in Step 7. Hence, we can assume without loss of generality that $r_{w_1} < l_{z_1} \leq l_{w_2} < r_{z_2}$. It remains to observe that each interval of U_i has its left end-point to the left of r_{w_1} and each interval of U_j has its left end-point to the right of r_{w_1}. This proves Claim 1.

Claim 1 implies that between the left endpoints of two intervals with color i there can be no left endpoint of an interval with color $j \neq i$. Then, similar as the correctness of the well-known greedy algorithm for finding a largest independent set in an interval graphs, we can argue that the above procedure outputs the required independent set. $\square$

As each step in our algorithm is safe, we obtain the following result.

Theorem 4. *The* Requirement Induced Disjoint Paths *problem can be solved in time $O(n + k)$ for interval graphs on n vertices with k terminal pairs if a representation of the graph is given.*

If no representation of the graph is given, then combined with Theorem 1 this implies a linear-time algorithm for Requirement Induced Disjoint Paths.

4 Circular-Arc Graphs

In this section, we modify the algorithm of the previous section to work for the Induced Disjoint Paths problem on circular-arc graphs. The general idea of the approach remains the same, but some preprocessing steps are no longer needed, and some steps need modification. In particular, we do not need colors here. We will again show that each step of the algorithm is *safe*, where the definition of a safe step remains the same, mutatis mutandis. The algorithm assumes that an arc representation of G is known, as given by Theorem 2. It maintains an auxiliary circular-arc graph H, initially empty, in a similar manner and function as before. It also maintains a set $\mathcal{P}$ of paths, initially empty.

4.1 Basic Tools

We need to extend the basic tools we defined for interval graphs to circular-arc graphs. For this, only minor modifications are necessary.

Definition 2. *Let C be a set of n arcs of a circle with arc end-points in $1, \ldots, 2n$. For a given integer j, a subset $C' \subseteq C$ is j-close if for each arc $u \in C'$ either u contains the point j or there is no arc $u' \in C \setminus C'$ that contains a point between r_u (exclusive) and j (inclusive) in "clockwise" direction.*

Lemma 13. *Let $c \geq 1$ be an integer, and let C be a set of n arcs of a circle with arc end-points in $1, \ldots, 2n$. Then in $O(cn)$ time, we can construct a j-close set C_j^* of $\min\{c, n\}$ arcs of C for all $j = 1, \ldots, 2n$.*

Proof. We perform a sweep-line algorithm. We start by initializing some data structures. For $j = 1, \ldots, 2n$, let $L_j = \{u \in C \mid l_u = j\}$ and $L_{2n+j} = \{u \in C \mid l_u = j\}$. The sets L_j, and L_{2n+j} can be computed in $O(n)$ time by for each $u \in C$ putting it into the two sets it should be in. Finally, initialize an empty deque D.

We now perform the sweep. For each $j = 1, \ldots, 2n$ in order, perform the following action. While there is a $u \in L_j$, remove u from L_j; if D contains less than c elements, then add u to D; otherwise, if the right endpoint of u lies further to the right than the right endpoint of the arc $v \in D$ with the leftmost right endpoint among all arcs in D, then remove v from D and add u instead. After finishing this, we again consider each $j = 1, \ldots, 2n$ in order, but now with respect to L_{2n+j}. We repeat the same action as above, but if L_{2n+j} is or becomes empty, then we make the set C_j^* equal to the contents of the deque D.

The first sweep initializes the deque, so that the second sweep can correctly report the C_j^*. Since the deque D contains at most c elements, the running time of the algorithm is $O(cn)$. □

Lemma 13 has the following corollary, which can be be proven in the same way as the proof of Corollary 1, except that we rely on Lemma 13 instead of Lemma 2.

Corollary 2. *Let X be a set of n arcs of a circle, with arc end-points in $1, \ldots, 2n$. Let $d \geq 1$ be an integer and let C, D be disjoint subsets of X. Then one can identify in $O(dn)$ time a set of arcs u in D that intersect at least d arcs in C, as well as for each such u a set of d arcs in C that intersect u,. or else conclude that no such u exists.*

5 Steps

We now describe the steps that the algorithm takes in detail. The algorithm first performs Step 1. Note that Steps 2 and 3 are not necessary, as there are no multi-pairs now, and thus we do not apply them. We then continue with Steps 4 and 5.

Lemma 14. *Steps 1, 4, and 5 are safe.*

The proof of this lemma is obtained in the same way as the proofs of Lemmas 3, 6, and 7 with the following caveats. We now rely on Corollary 2 instead of Corollary 1. In the implementation of Lemma 7 we must be careful that the intersection of the arcs of two terminal vertices can consist of two disjoint arcs; however, by adding both arcs to the set X constructed in Lemma 7 in this case, the algorithm goes through as before.

After Step 5, for every remaining terminal pair (s_i, t_i), s_i and t_i are represented by vertices at distance at least 2, and as before, we call such pairs *long*. Let k' be the number of remaining terminal pairs. Notice that it can happen that $k' \leq 1$ after Step 5. It is convenient to handle this case separately.

Step 5^+. If $k' = 0$, then stop and return the solution $\mathcal{P}$. If $k' = 1$, then consider the terminal vertices u and v representing the terminals of the unique pair of T. Find an induced uv-path P if it exists. If P exists, then add P to $\mathcal{P}$, and return the solution $\mathcal{P}$. Otherwise, stop and return a `No`-answer.

Lemma 15. *Step 5^+ is safe.*

Proof. Step 5^+ can be executed in $O(n)$ time, because if $k' = 1$, then the required path P can be found by tracing a path along the circle. The cases that $k' = 0$ and that $k' = 1$ and P does not exist are trivially correct. If $k' = 1$ and P does exist, then P cannot have any inner (non-terminal) vertices that are adjacent to the terminal vertices that are deleted in Step 4, because any such non-terminal vertices are deleted as well. Moreover, P cannot have any inner (non-terminal) vertices that are adjacent to the terminals that are deleted in Step 5, as any such non-terminal vertex would either be adjacent to three terminals and thus removed in Step 1, or be adjacent to a terminal vertex of the single remaining terminal pair. □

Now we can assume that $k' \geq 2$. Since all pairs are long and $k' \geq 2$, there is only one direction around the circle that a solution path can go, and therefore, intuitively, the problem starts to behave roughly as it does on interval graphs. We perform Steps 6, 7, 8, and 9, where in Step 9 we do not color the vertices.

Lemma 16. *Steps 6, 7, 8, and 9 are safe. Moreover, for $i = 1, \ldots, k'$, if P is a solution $s_i t_i$-path, then there is a candidate $s_i t_i$-path P' with $I_{P'} \subseteq I_P$.*

Proof. The lemma follows immediately from Lemmas 8, 9, 10, and 11. In the proof of Lemma 9, we need to be slightly careful: if the first two non-empty buckets contain terminals from different terminal pairs, then since we are dealing with circular-arc graphs, this does not immediately mean that we should return a `No`-answer. Instead, we should restart the procedure with the second non-empty bucket, and move the first non-empty bucket to the end of the list (as bucket B_{2n+1}). In the other lemmas, we now rely on Corollary 2 instead of Corollary 1. In Lemma 11 it is important to note that for each $i = 1, \ldots, k'$, the terminal vertices representing s_i and t_i and the $n_i - 2$ non-terminal vertices involved with (s_i, t_i) induce an interval graph. Hence, the proof of the lemma goes through without further modifications. □

Finally, we execute the following simplified version of Step 10.

Step 10*. Find a largest independent set in H using Theorem 3. If such a set exists, add the corresponding candidate paths to $\mathcal{P}$ and return $\mathcal{P}$. Otherwise, return a `No`-answer.

Lemma 17. *Step 10* is safe.*

Proof. A largest independent set can be found in $O(n)$ time using Theorem 3. Then the corresponding paths can be found in $O(n)$ time (by assuming that all arcs are annotated with their corresponding paths of at most four vertices, or in the case of the arcs generated by Step 9c, by the four vertices and component number that generated the interval). To prove that Step 10* is correct, we prove that the instance is a yes-instance if and only if H has an independent set of size at least k'.

Suppose that $\mathcal{I}$ is an independent set of H of size at least k'. By the construction of H, the set of vertices of H can be partitioned into k' sets $X_1, \ldots, X_{k'}$ such that for each $i \in \{1, \ldots, k'\}$, X_i contains only intervals that intersect the vertices u, v representing s_i, t_i, respectively, in r_u and l_v. Hence, $\mathcal{I}$ has exactly one vertex from each $X_1, \ldots, X_{k'}$. For each interval w in $\mathcal{I}$ from X_i, we can find an s_it_i-path in G with inner vertices that are used to construct w. Taking into account the paths that are already included in $\mathcal{P}$, we obtain s_it_i-paths for each $i \in \{1, \ldots, k\}$. We have to show that these paths are mutually induced. Because $\mathcal{I}$ is an independent set, distinct paths have no adjacent inner vertices. It remains to show that each $w \in \mathcal{I}$ does not intersect any terminal vertex (interval) of G except the vertices representing s_i and t_i. Notice that w does not intersect any terminal vertex deleted in Step 4, because we delete such terminal vertices together with adjacent non-terminal vertices. Similarly, as argued in Lemma 7, w does not interfere with any terminals deleted in Step 5. Recall that non-terminal vertices that are adjacent to at least three distinct terminal vertices are deleted in Step 1. By Step 8 and the fact that the common neighbors of two terminals are deleted in the first phase of the construction of H in Step 9a, we find that w does not intersect any terminal except s_i, t_i. Hence, the instance is a yes-instance.

Suppose now that we have a yes-instance of Induced Disjoint Paths. Consider a solution for this instance. By Observation 1, we can assume that this solution includes all terminal paths from $\mathcal{P}$. We consider the remaining k' paths that have at least one inner vertex. By Lemma 16, for each solution s_it_i-path P, there is a candidate $s_i t_i$-path with $I_{P'} \subseteq I_P$. Hence, we may assume that each solution path is a candidate path. Let $\mathcal{I}$ be the set of intervals covered by these paths. Because the paths are mutually induced, the intervals of $\mathcal{I}$ do not intersect each other. Hence, H has an independent set of size k'. □

As each step in our algorithm is safe, we obtain the following result.

Theorem 5. *The* Induced Disjoint Paths *problem can be solved in time $O(n+k)$ for circular-arc graphs on n vertices with k terminal pairs if a representation of the graph is given.*

If no representation of the graph is given, then combined with Theorem 2 this implies a linear-time algorithm for Induced Disjoint Paths.

6 Conclusion

We gave a linear-time algorithm for Requirement Induced Disjoint Paths on interval graphs, and for Induced Disjoint Paths on circular-arc graphs. Both algorithms actually run in $O(n+k)$ time if a representation of the graph is given and the graph has n vertices and k terminal pairs. By the application of the same ideas, we can solve Requirement Induced Disjoint Paths on n-vertex circular-arc graphs in time $O(n^2)$. The increase in running time is because to solve the auxiliary problem of finding a multicolored independent set we must "guess" a starting point for the greedy selection of such a set. The question whether there exists a linear-time algorithm for Requirement Induced Disjoint Paths restricted to circular-arc graphs remains open.

A natural question is whether the multicolored independent set problem that we solve in Step 10 of the algorithm can be solved in polynomial time on interval graphs when no order on the colors is known. In the appendix, we answer this question negatively.

References

1. R. Belmonte, P.A. Golovach, P. Heggernes, P. van 't Hof, M. Kaminski and D. Paulusma, Detecting fixed patterns in chordal graphs in polynomial time. *Algorithmica* **69** (2014) 501–521.
2. D. Bienstock. On the complexity of testing for odd holes and induced odd paths. *Discrete Mathematics* **90** (1991) 85–92. See also Corrigendum, Discrete Mathematics **102** (1992) 109.
3. K.S. Booth and G.S. Lueker, Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.* **13** (1976), 335–379.
4. R. Diestel, *Graph Theory.* Springer-Verlag, Electronic Edition, 2005.
5. M.R. Fellows. The Robertson–Seymour theorems: A survey of applications. *Proc. of the AMS-IMS-SIAM Joint Summer Research Conference*, Contemporary Mathematics, vol. 89, American Mathematical Society, Providence (1989) 1–18.

6. J. Fiala, M. Kamiński, B. Lidicky, and D. Paulusma. The k-in-a-path problem for claw-free graphs. *Algorithmica* **62** (2012) 499-519.
7. F.V. Fomin, I. Todinca, and Y. Villanger. Large induced subgraphs via triangulations and CMSO. In: *Proc. SODA 2014*, SIAM (2014) 582–593.
8. P.A. Golovach, D. Paulusma and E.J. van Leeuwen, Induced disjoint paths in AT-free graphs. In: *Proc. SWAT 2012*, LNCS 7357 (2012) 153–164.
9. P.A. Golovach, D. Paulusma and E.J. van Leeuwen, Induced disjoint paths in claw-free graphs. In: *Proc. ESA 2012*, LNCS 7501 (2012) 515–526.
10. P.A. Golovach, D. Paulusma and E.J. van Leeuwen, Induced disjoint paths in circular-arc graphs in linear time. In: *Proc. WG 2014*, LNCS 8747 (2014) 225–237.
11. M.C. Golumbic and P.L Hammer. Stability in circular arc graphs. *J. Algorithms* **9** (1988) 56–63.
12. F. Gurski, E. Wanke. Vertex disjoint paths on clique-width bounded graphs. *Theor. Comput. Sci.* **359** (2006) 188–199.
13. M. Habib, R.M. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theor. Comput. Sci.* **234**(2000) 59–84.
14. P. Heggernes, P. van 't Hof, R. Saei, E.J. van Leeuwen. Finding disjoint paths in split graphs. In: *Proc. SOFSEM 2014*, LNCS 8327 (2014) 315–326.
15. H. Kaplan and Y. Nussbaum. A Simpler Linear-Time Recognition of Circular-Arc Graphs. *Algorithmica* **61** (2011) 694–737.
16. R.M. Karp. On the complexity of combinatorial problems. *Networks* **5** (1975) 45–68.
17. Y. Kobayashi and K. Kawarabayashi. A linear time algorithm for the induced disjoint paths problem in planar graphs. *J. Comput. Syst. Sci.* **78** (2012) 670–680.
18. N. Korte and R.H. Möhring. An incremental linear–time algorithm for recognizing interval graphs *SIAM J. Computing* **18** (1989) 68–81.
19. M. Kramer, J. van Leeuwen. The complexity of wirerouting and finding minimum area layouts for arbitrary VLSI circuits. *Adv. Comput. Res.* **2** (1984), 129–146.
20. J.F. Lynch. The equivalence of theorem proving and the interconnection problem. *SIGDA Newsletter* **5** (1975) 31–36.
21. R.M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica* **37** (2003) 93–147.
22. S. Natarajan, A.P. Sprague. Disjoint paths in circular arc graphs. *Nordic J. Computing* **3** (1996) 256–270.
23. B.A. Reed. Tree width and tangles: A new connectivity measure and some applications. In: *Surveys in Combinatorics* Cambridge University Press, (1997) 87–162.
24. B.A. Reed, N. Robertson, A. Schrijver, P.D. Seymour. Finding disjoint trees in planar graphs in linear time. In: *Contemporary Mathematics* vol. 147, American Mathematical Society (1993) 295–301.
25. N. Robertson and P.D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B* **63** (1995) 65–110.

A Multicolored Independent Set

In Step 10 of the algorithm for interval graphs, we solve an instance of a generalization of the following problem:

Multicolored Independent Set
Instance: a graph G, an integer k, and a function $c : V(G) \to \{1, \ldots, k\}$. *Question:* does G have an independent set I with $\bigcup_{v \in I} c(v) = \{1, \ldots, k\}$?

In Step 10, we essentially show that such an instance can be solved in polynomial time on interval graphs if for any two vertices u, w with $c(u) = c(w) = i$ there is no vertex v with $c(v) = j$ and $l_u < l_v < l_w$. However, on general interval graphs, this problem becomes NP-complete.

Theorem 6. Multicolored Independent Set *on interval graphs is NP-complete.*

Proof. We show in fact that the problem is already NP-complete on disjoint unions of double stars (i.e. graphs obtained from two disjoint stars by adding an edge between the two central vertices), which form a subclass of interval graphs. We reduce from 3-SAT. Consider an instance of 3-SAT with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$. We construct a graph G and a function c as follows. For each x_i, we create two adjacent vertices x_i and $\bar{x}_i$ with $c(x_i) = c(\bar{x}_i) = i$. For each C_j, we create three vertices and set $c(\cdot)$ of these vertices to $j + n$. We then make these three vertices adjacent to the corresponding literal vertices (for example, if C_j contains $x_i, \bar{x}_j, x_l$, then we join the first vertex with the vertex x_i, the second with $\bar{x}_j$ and the third with x_l). This completes the construction. Note that it is indeed a disjoint union of double stars. The correctness can be seen as follows: we set variable x_i to true if and only if the vertex x_i is not in the independent set. $\square$

It is easy to show that Multicolored Independent Set is fixed-parameter tractable on interval graphs when parameterized by the number of colors: guess an ordering of the colors, and for each choice, run a procedure similar to the one described for Step 10. A faster algorithm can be obtained using dynamic programming.