

Efficient Enumeration of Non-isomorphic Interval Graphs

Patryk Mikos*

Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

received 27th Feb. 2020, revised 15th Sep. 2020, accepted 23rd Dec. 2020.

Recently, Yamazaki et al. provided an algorithm that enumerates all non-isomorphic interval graphs on n vertices with an $O(n^4)$ time delay. In this paper, we improve their algorithm and achieve $O(n^3 \log n)$ time delay. We also extend the catalog of these graphs providing a list of all non-isomorphic interval graphs for all n up to 15.

Keywords: combinatorics, graph theory, enumeration, interval graphs

1 Introduction

Graph enumeration problems, besides their theoretical value, are of interest not only for computer scientists, but also to other fields, such as physics, chemistry, or biology. Enumeration is helpful when we want to verify some hypothesis on a quite big set of different instances, or find a small counterexample. For graphs it is natural to say that two graphs are "different" if they are non-isomorphic. Many papers dealing with the problem of enumeration were published for certain graph classes, see Kiyomi et al. (2006); Saitoh et al. (2010, 2012); Yamazaki et al. (2018). A series of potential applications in molecular biology, DNA sequencing, network multiplexing, resource allocation, job scheduling, and many other problems, makes the class of *interval graphs*, i.e. intersection graphs of intervals on the real line, a particularly interesting class of graphs. In this paper, we focus on interval graphs, and our goal is to find an efficient algorithm that for a given n lists all non-isomorphic interval graphs on n vertices. It is well-known that the number of such graphs is roughly n^{nc} for some constant c , see Acan (2018); Hanlon (1982); J.C. Yang (2017). For that reason, we measure the efficiency of the enumeration algorithm by the worst-case time delay between output of any two successive graphs.

1.1 Previous work

Yamazaki et al. (2018) presented an enumeration algorithm for non-isomorphic interval graphs that works with the worst-case time delay $O(n^6)$, and recently Yamazaki et al. (2019) improved it to $O(n^4)$. Their algorithm is based on the *reverse search method*, invented by Avis and Fukuda (1996), and in its general

*Research was partially supported by the National Science Center of Poland under grant no. 2014/14/A/ST6/00138.

form works in the following way. Let \mathcal{C} be a family of graphs we want to enumerate, and let $G_1, \dots, G_k \in \mathcal{C}$ be some special graphs called *roots*. We define a *family forest* \mathcal{F} which spans \mathcal{C} , and consists of k rooted trees $\mathcal{T}_1, \dots, \mathcal{T}_k$ such that the root of \mathcal{T}_i is G_i . For graphs that are not roots, let $par : \mathcal{C} \setminus \{G_1, \dots, G_k\} \rightarrow \mathcal{C}$ be the *parent function*. In order to enumerate all graphs in the family \mathcal{C} , we enumerate all graphs that belong to each tree independently, and to enumerate all graphs in the tree \mathcal{T}_i we use any tree traversal algorithm like BFS or DFS. The most time consuming operation in the tree traversal is computing the children of a graph G . From the definition, children of G are those graphs $G' \in \mathcal{C}$ whose parent is G . Hence, if we want to design a fast enumeration algorithm that uses this technique, we need to carefully define the parent function. Yamazaki et al. (2019) used the fact that for every interval graph $G = (V, E)$ that is not a complete graph, there is at least one edge e such that the graph $G_e = (V, E \cup \{e\})$ is also an interval graph. They defined the only root G_1 to be a complete graph on n vertices, and $par(G) = G_e$, where the edge e is uniquely defined - for more details see Yamazaki et al. (2019). The consequence of this approach is the fact that for every graph G on n vertices there are at most $O(n^2)$ candidates for the children of G in \mathcal{F} , and so this number does not depend on the size of the enumerated family. Moreover, the authors observed that in order to enumerate only non-isomorphic graphs, it is enough to filter out isomorphic copies from the set of children. Isomorphism test in the class of interval graphs is an easy problem thanks to *MPQ*-trees, see Section 2. Hence, to compute the set of children for a graph $G = (V, E)$, the authors consider all graphs $G_e = (V, E \setminus \{e\})$. For each of them, they check whether G_e is an interval graph using some linear time recognition algorithm. Then, if G_e is an interval graph, they check whether $G = par(G_e)$, and build a corresponding *MPQ*-tree. Finally, they store a set of children trees, effectively removing all duplicates.

1.2 Our results

In this paper we revisit the work of Yamazaki et al. and show how to modify their enumeration algorithm to significantly reduce the worst-case time delay between the output of two successive graphs. Our key observation is the fact that having an *MPQ*-tree corresponding to a graph $G = (V, E)$ we are able to list all edges e such that a graph $G_e = (V, E \setminus \{e\})$ is an interval graph. Moreover, for each such edge we show how to build an *MPQ*-tree corresponding to the graph G_e without constructing it explicitly.

1.3 Organization of this paper

In the next section we introduce concepts and definitions that are widely used in this paper, and also provide a detailed description of *MPQ*-trees along with their most important properties. In Section 3, we present a total ordering over all *MPQ*-trees, define a canonical *MPQ*-tree using this ordering, and also present a fast algorithm that for a given *MPQ*-tree \mathcal{T} computes its canonical form \mathcal{T}' . In Section 4 we consider an *MPQ*-tree \mathcal{T} corresponding to an interval graph $G = (V, E)$, and characterize edges e such that the graph $G_e = (V, E \setminus \{e\})$ is also an interval graph. Moreover, for every edge e we either show a linear time algorithm that produces a string representing G_e if it is an interval graph, or show an induced chordless cycle on four vertices or an asteroidal triple in G_e that certifies that G_e is not an interval graph. In Section 5 we develop data structures and algorithms that make use of combinatorial characterization from Section 4 and present a fast algorithm which for a given *MPQ*-tree lists all edges e such that G_e is an interval graph. Finally, in Section 6 we show how to combine all parts together and build the graph enumeration algorithm. We also show the worst-case performance analysis of our algorithm in this section. The last section contains a discussion of some implementation heuristics that do not change the worst-case analysis, but significantly speedup the execution.

2 Preliminaries

In this paper we consider only simple graphs without loops and multiple edges. We use the standard notations for graphs, so $n = |V(G)|$ and $m = |E(G)|$. For a graph $G = (V, E)$ and a pair of vertices $i, j \in V$, we denote $G + (i, j)$ a graph $G' = (V, E \cup \{(i, j)\})$, and $G - (i, j)$ a graph $G' = (V, E \setminus \{(i, j)\})$. A graph $G = (V, E)$ with a vertex set $V = \{1, \dots, n\}$ is an *interval graph* if there is a set of intervals $\mathcal{I} = \{I_1, \dots, I_n\}$ on the real line such that $(i, j) \in E$ iff $I_i \cap I_j \neq \emptyset$. The set \mathcal{I} is called an *interval representation* of the graph G . For an interval graph G , we say that an edge $(i, j) \in E(G)$ is an *interval edge* if $G - (i, j)$ is also an interval graph. A sequence \mathcal{S} of length $2n$ is called a *string representation* if each element $x \in \{1, \dots, n\}$ appears exactly two times in \mathcal{S} . Note that a string representation \mathcal{S} encodes an interval graph in a natural way. For every $x \in \{1, \dots, n\}$ let $first(x)$ denote the index of the first appearance of x in \mathcal{S} , $second(x)$ denote the second one, and x is represented by an interval $I_x = [first(x), second(x)]$.

2.1 PQ-trees

It is easy to notice that an interval graph can have many different interval representations. Lueker and Booth (1979) introduced a data structure, called a *PQ-tree*, which encodes all normalized interval representations of an interval graph. A *PQ-tree* is a rooted labeled plane tree composed of leaves and two kinds of internal nodes called *P-nodes*, and *Q-nodes* respectively. The left to right ordering of the leaves of a *PQ-tree* T is called the *frontier* of T . We say that T encodes an interval graph G , if each maximal clique of the graph G is stored in exactly one leaf of T , and each vertex $v \in V(G)$ belongs to a consecutive sequence of cliques in the frontier of T . Having a *PQ-tree* T one can obtain another *PQ-tree* T' which is *equivalent* to T using the following two operations: arbitrarily permute the children of a *P-node*, or reverse the order of the children of a *Q-node*. The crucial property of a *PQ-tree* T is the fact that for every permutation σ of maximal cliques of the graph G such that each vertex belongs to a consecutive sequence of cliques, there is a *PQ-tree* T' that is equivalent to T , and frontier of T' represents σ . In other words, each normalized interval representation of the graph G is represented by some tree equivalent to T .

2.2 MPQ-trees

PQ-trees are quite a simple and easy to understand data structure representing interval graphs, but unfortunately they may occupy up to $O(n^2)$ space. To reduce the space consumption, Korte and Möhring (1989) presented *modified PQ-trees* called *MPQ-trees*. In an *MPQ-tree*, we do not store maximal cliques in leaves, but we assign to each *P-node* and each child of a *Q-node* a set of vertices in such a way that vertices laying on a path from the root of the tree to some leaf represent a maximal clique in G , see Figure 1C for an example. For a *Q-node* Q with children T_1, \dots, T_k , we denote S_i the set of vertices assigned to T_i , and call it the *i-th section* of Q . Note that, each vertex belongs to the consecutive sequence of maximal cliques, so it has to belong to consecutive sequence of sections of a *Q-node*. Hence, in order to limit the used space, we can store the information about the vertex x only in the first and last section it belongs to. Thanks to this modification, an *MPQ-tree* is an $O(n)$ space representation of an interval graph. In this paper we show several drawings of *MPQ-trees*. We represent *P-nodes* as circles, and *Q-nodes* as rectangles divided into smaller rectangles representing sections of the *Q-node*. For instance, in the Figure 1C the root is an empty *P-node*, and the vertex 6 belongs to the sections S_2 and S_3 of the only *Q-node*.

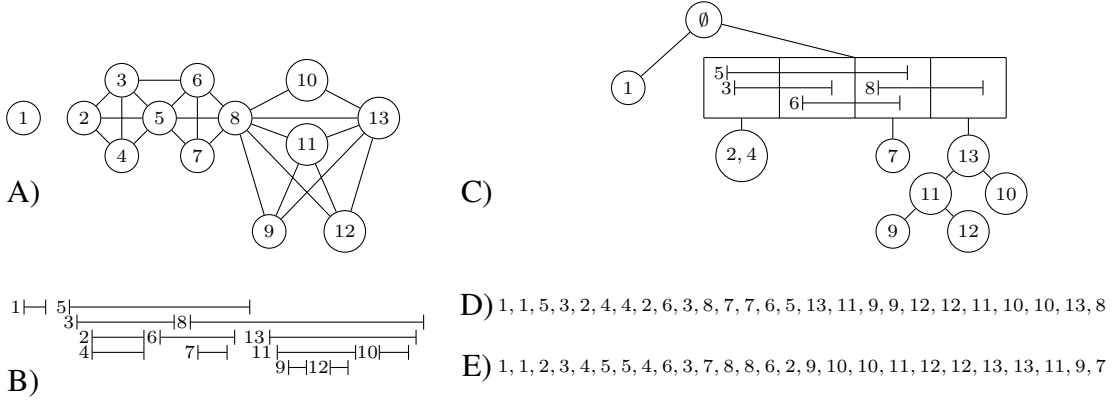


Fig. 1: A) An interval graph G , B) Its interval representation \mathcal{I} , C) Its MPQ -tree \mathcal{T} , D) String representation \mathcal{S} of tree \mathcal{T} , E) Canonical string representation

2.3 Known results

During past decades, many researchers published their results on constructing both PQ -trees and MPQ -trees. Those trees were mostly used to determine whether a given graph $G = (V, E)$ is an interval graph or not. Lueker and Booth (1979) in their recognition algorithm used PQ -trees and proved that for a given graph G the corresponding PQ -tree can be computed in $O(n + m)$ time. Korte and Möhring (1989) presented analogous result for MPQ -trees. In this paper we are most interested in work of T. Saitoh (2007) who presented an algorithm that constructs an MPQ -tree for a given interval graph representation and works in $O(n \log n)$ time, or $O(n)$ if the endpoints of intervals are given in an ascending order.

Theorem 1 (T. Saitoh (2007) Thm.12) *If the graph G is given as an interval representation such that the endpoints are sorted by the coordinates, then there is an algorithm that produces an MPQ -tree corresponding to G in $O(n)$ time.*

Clearly, having a string representation of the graph G , we can produce an interval representation satisfying the conditions of Theorem 1 in $O(n)$ time. Hence, we have the following corollary.

Corollary 2 *There is an algorithm that for a given string representation \mathcal{S} of the graph G builds a corresponding MPQ -tree \mathcal{T} in $O(n)$ time.*

Before we proceed to technical definitions and lemmas, we provide some naming conventions we are going to use in the rest of this paper. To avoid a confusion when talking about elements of a graph and elements of a tree, we always refer elements of a graph as *vertices* and elements of a tree as *nodes*. For a vertex v of a graph G , we denote $node(v)$ the node of a corresponding MPQ -tree \mathcal{T} such that v belongs to the set assigned to that node. For a node with k subtrees T_1, \dots, T_k , we denote V_i the set of all vertices that are assigned to the nodes of a subtree T_i . If $V_i = \emptyset$, then we say that a subtree T_i is empty. For a Q -node we say that a vertex v has its *left endpoint* in a section $S_{l(v)}$, if v belongs to $S_{l(v)}$ and does not belong to any other section S_b with $b < l(v)$. Analogously, we say that v has its *right endpoint* in $S_{r(v)}$, if v belongs to $S_{r(v)}$ and does not belong to any other section S_b with $b > r(v)$. Vertex v is *contained* in sections S_a, \dots, S_b , if $a \leq l(v) < r(v) \leq b$.

For an *MPQ*-tree \mathcal{T} we define a $2n$ -element string \mathcal{S} called *string representation* of \mathcal{T} . This string is built recursively over the structure of \mathcal{T} . For a *P*-node we first output all vertices that belong to that node, then recursively string representations of the children from left to right, and at the end yet again all vertices that belong to that node, but now in the reversed order. Hence, the string representation for a *P*-node with vertices $\{1, \dots, k\}$ and no children is $123 \dots (k-1)kk(k-1) \dots 321$. A string representation for a *Q*-node is a concatenation of string representations for its sections. The string for a section S_i starts with vertices that have its left endpoint in S_i , then there is a string for a subtree T_i , and finally vertices that have its right endpoint in S_i . It is easy to see that string representation of \mathcal{T} is also a string representation of the graph corresponding to \mathcal{T} . We also define a *normalized string representation* of \mathcal{T} . Consider a permutation $\sigma : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$, and a string $\sigma(\mathcal{S})$, which results from the application of σ to each element of \mathcal{S} . Normalized string representation is the lexicographically smallest string $\sigma(\mathcal{S})$ among all permutations σ . Finally, we recall some properties of *MPQ*-trees produced by the Algorithm from Theorem 1.

Lemma 3 (Korte and Möhring (1989); Uehara (2005)) *In the MPQ-tree constructed in Theorem 1 for every Q-node with k children we have:*

- a) $V_1 \neq \emptyset$ and $V_k \neq \emptyset$,
- b) $S_1 \subset S_2$ and $S_k \subset S_{k-1}$,
- c) $S_{i-1} \cap S_i \neq \emptyset$ for $2 \leq i \leq k$,
- d) $S_{i-1} \neq S_i$ for $2 \leq i \leq k$,
- e) $(S_i \cap S_{i+1}) \setminus S_1 \neq \emptyset$ and $(S_{i-1} \cap S_i) \setminus S_k \neq \emptyset$ for $2 \leq i \leq k-1$, and
- f) $(S_{i-1} \cup V_{i-1}) \setminus S_i \neq \emptyset$ and $(S_i \cup V_i) \setminus S_{i-1} \neq \emptyset$ for $2 \leq i \leq k$.

Moreover:

- g) there are no two empty *P*-nodes such that one of them is a parent of the other,
- h) there is no *P*-node that have only one child which root is also a *P*-node, and
- i) *P*-nodes have no empty children

see Figure 2.

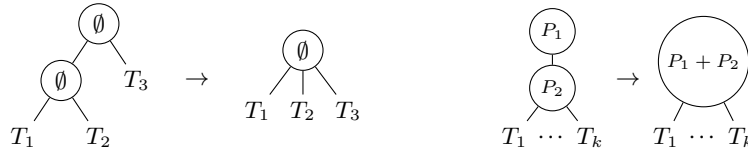


Fig. 2: *MPQ*-trees do not contain two consecutive empty *P*-nodes (left), or a *P*-node with only one child which root is also a *P*-node (right).

It is worth noting that *MPQ*-trees describe interval graph in a natural recursive way. If G consists of at least 2 connected components, then the root node of *MPQ*-tree \mathcal{T} corresponding with G is an empty P -node, and each subtree corresponds to each connected component of G . If G is connected and vertices v_{i_1}, \dots, v_{i_k} are universal in G (vertex is universal if it is connected to all other vertices), then the root of \mathcal{T} node is a P -node containing v_{i_1}, \dots, v_{i_k} . In the remaining cases the root of \mathcal{T} is a Q -node.

3 Canonical *MPQ*-tree

In this section, we define a total ordering \prec on *MPQ*-trees. One may notice that the lexicographical order on string representations is a total ordering on *MPQ*-trees, but for complexity reasons we introduce a different one. Denote $|\mathcal{T}|$ the number of vertices contained in the tree \mathcal{T} , $c(\mathcal{T})$ the number of children of the root of \mathcal{T} , and $ex(\mathcal{T})$ the number of vertices that belong to the root of \mathcal{T} . We assign a tuple $t_{\mathcal{T}} = \langle |\mathcal{T}|, ex(\mathcal{T}), c(\mathcal{T}) \rangle \in \mathbb{N}^3$ to every tree \mathcal{T} , and say that if $t_{\mathcal{T}_1}$ is lexicographically smaller than $t_{\mathcal{T}_2}$, then $\mathcal{T}_1 \prec \mathcal{T}_2$. For two trees \mathcal{T}_1 and \mathcal{T}_2 such that $t_{\mathcal{T}_1} = t_{\mathcal{T}_2}$, we say that $\mathcal{T}_1 \prec \mathcal{T}_2$, if the normalized string representation of \mathcal{T}_1 is lexicographically not greater than normalized string representation of \mathcal{T}_2 . We say that *MPQ*-tree \mathcal{T} is in *canonical form*, if for every other tree \mathcal{T}' representing the same graph G we have $\mathcal{T} \prec \mathcal{T}'$. A string \mathcal{S} is a *canonical string*, if it is a normalized string representation of a canonical tree. Observe that if \mathcal{T} is in a canonical form, then all subtrees of \mathcal{T} are in a canonical form. Clearly, if some subtree of \mathcal{T} is not in a canonical form, then we may rotate it and obtain a lexicographically smaller string.

Theorem 4 *Two interval graphs G_1 and G_2 are isomorphic if and only if their canonical strings S_1 and S_2 are equal.*

Theorem 5 *There is an algorithm that for every *MPQ*-tree \mathcal{T} computes its canonical form in $O(n \log n)$ time.*

Proof: At the very beginning, we shall compute a function g that for every vertex v will describe its relative position among all vertices from $node(v)$. We compute this function for each P -node independently, and for all Q -nodes collectively. For a P -node with j vertices z_1, \dots, z_j , we assign $g(z_i) = i$. Thus, we can compute the function g for all P -nodes in $O(n)$ time. In order to compute this function for all Q -nodes, at first we assign a tuple $\langle l(v), r(v) \rangle$ to each vertex v which belongs to some Q -node. Then we sort all tuples using radix sort algorithm, and visit vertices in the order determined by their tuples. For each Q -node we keep a local counter that starts with 1 and increases each time we visit a vertex from this node. Thus, because all vertices are from the set $\{1, \dots, n\}$, and each Q -node has a linear in terms of n number of sections, we compute this function for all vertices in $O(n)$ time.

We shall construct a function f that assigns an integer $f(\mathcal{T}') > n$ to every subtree \mathcal{T}' of a tree \mathcal{T} in such a way that $\mathcal{T}_1 \prec \mathcal{T}_2 \Leftrightarrow f(\mathcal{T}_1) \leq f(\mathcal{T}_2)$. Simultaneously we will rotate subtrees so that they are in canonical form. At first, we compute tuples $t_{\mathcal{T}'}$ for each subtree \mathcal{T}' of a tree \mathcal{T} . Clearly, it can be easily done in $O(n)$ time. Then, we sort the tuples lexicographically in $O(n)$ using radix sort algorithm. In the next phases, we inspect nodes of tree \mathcal{T} that have the same tuple $\langle |\mathcal{T}'|, ex(\mathcal{T}'), c(\mathcal{T}') \rangle$, and we do it from the smallest tuples to the biggest ones. Observe that, when we define the value of the function f for \mathcal{T}' , the values for all subtrees of \mathcal{T}' are already computed.

All subtrees of \mathcal{T}' are in a canonical form, so in order to compute a canonical form of \mathcal{T}' , we need to determine the order of its children. If the root of \mathcal{T}' is a P -node, then we use integers $f(\mathcal{T}_i)$ as keys for children, and sort them in $O(c \log c)$ time, where $c = c(\mathcal{T}')$. If the root of \mathcal{T}' is a Q -node Q , then we may

leave it in the form it is or reverse it. To decide what to do, we compute a special string representation \mathcal{S}^* , which is similar to the string representation, but for each vertex v that belongs to Q we put $g(v)$ instead of v , and instead of inserting the whole string for a subtree \mathcal{T}_i , we put a single number $f(\mathcal{T}_i)$. Hence, the produced string has length $2 * ex(\mathcal{T}') + c(\mathcal{T}')$ and is produced in time proportional to its length. We also produce similar string for a rotated node, and if that string is lexicographically smaller than the original one, then we rotate Q . Otherwise, we do nothing.

We have just computed canonical forms for all subtrees with the same tuple. For each of them we produce a special string, and sort those strings lexicographically. Finally, we assign values from the set $\{F + 1, F + 2, \dots\}$, where F is the maximal number assigned to trees with lexicographically smaller tuples (or $F = n$ if there are no smaller). We assign those numbers according to the computed order giving the same value to the subtrees with the same special string representation, and that finishes the algorithm description.

Now, we prove the algorithm works in the declared time. As we mentioned before, the computation of the function g is linear in time. The same applies to the computation and sorting for the node tuples. Sorting children of a P -node with c children takes $O(c \log c)$. Hence, because all P -nodes cannot have more than $O(n)$ children in total, we conclude that sorting children for all P -nodes takes no more than $O(n \log n)$ time. The length of a special string for a Q -node with j vertices and k sections is $O(j + k)$. Thus, the total processing time for all Q -nodes is linear in terms of n .

The only thing we have not counted yet is the time spent on sorting subtrees with the same tuple. Note that for a tuple $\langle s, e, c \rangle$, each special string has length exactly $2e + c$. Let n_{sec} be the number of subtrees having a tuple $\langle s, e, c \rangle$. Sorting process for those subtrees takes no more than $O((e + c)n_{sec} \log n_{sec})$. Thus, all sortings together take $O(\sum_{sec} (e + c)n_{sec} \log n_{sec})$. Note that $n_{sec} \leq n$, so we only need to show that $\sum_{sec} (e + c)n_{sec}$ is $O(n)$. But, clearly $\sum_{sec} en_{sec} = n$ since this sum counts vertices in all nodes. Similarly, $\sum_{sec} cn_{sec}$ equals the number of edges in \mathcal{T} , and we are done. \square

4 Interval edges

In this section we present a series of lemmas that characterize the interval edges for the interval graph G . Moreover, for each interval edge (x, y) , we also present a linear in terms of n algorithm that produces a string representation for the interval graph $G - (x, y)$. For an edge (x, y) that is not an interval edge, we prove the existence of an induced chordless cycle on four vertices or an asteroidal triple in $G - (x, y)$. The characterization does not use the mere graph G , but the corresponding MPQ -tree \mathcal{T} instead.

First, let us introduce an useful definition. We say that x is over y in \mathcal{T} , if $(x, y) \in E(G)$ and $node(x)$ is the lowest common ancestor of $node(x)$ and $node(y)$ in \mathcal{T} . Notice that, if there is an edge (x, y) in the graph G , then x is over y , or y is over x . Now, we make an easy observation on interval edges.

Observation 6 *If there are at least two vertices z_1 and z_2 such that both x and y are over z_1 and z_2 , and there is no edge (z_1, z_2) , then (x, y) is not an interval edge.*

Proof: Vertices x, z_1, y and z_2 in that order form a cycle of length 4. We assumed that there is no edge between z_1 and z_2 , so if there is no edge between x and y , then this cycle is chordless in $G - (x, y)$, see Figure 3. Hence, $G - (x, y)$ is not a chordal graph and so not an interval graph. \square

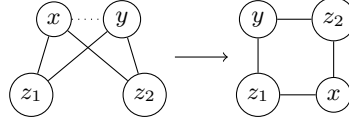


Fig. 3: An induced C_4 after removing edge (x, y) .

The above observation is useful when we want to prove that the edge (x, y) is not an interval edge. However, in cases when (x, y) is an interval edge, we want to show a linear time algorithm that produces a string representation for a graph $G - (x, y)$. The following lemma, we call the *swapping lemma*, comes handy when we try to produce the mentioned string. It shows when we can swap two consecutive elements in a string representation without adding or removing any edges to the represented graph.

Lemma 7 *Let \mathcal{S}_1 and \mathcal{S}_2 be string representations, such that \mathcal{S}_2 is created from \mathcal{S}_1 by swapping elements at positions i and $i + 1$ for some i . Denote a the element at the i -th position in \mathcal{S}_1 , and b the element at the $(i + 1)$ -th position ($\mathcal{S}_1 = \dots ab\dots$ and $\mathcal{S}_2 = \dots ba\dots$). \mathcal{S}_1 and \mathcal{S}_2 represent the same interval graph iff both elements at swapped positions represent either left endpoints or right endpoints.*

Proof: Clearly, at most one edge can be added or removed by swapping those two elements. If we swap the left endpoint of a and the right endpoint of b , then we remove an edge (a, b) . If we swap the right endpoint of a and the left endpoint of b , then we add an edge (a, b) which is not present in \mathcal{S}_1 . If both elements represent left endpoints, then right endpoints for both a and b are to the right of $i + 1$, hence no edge is added or removed. Similar argument works when both elements represent right endpoints. \square

Our aim is to characterize all interval edges encoded by an *MPQ*-tree \mathcal{T} . Hence, as an input we are given an *MPQ*-tree \mathcal{T} and some edge $(x, y) \in E(G)$. Without loss of generality, we assume that x is over y in \mathcal{T} , and we work under this assumption in the following subsections. We split our argument into cases according to the relative position of $node(x)$ and $node(y)$ in \mathcal{T} .

4.1 x and y belong to the same *P*-node

At first, we consider the case where both x and y belong to the same *P*-node P in \mathcal{T} . We show that under this assumption, the edge (x, y) is an interval edge if and only if P is a leaf in \mathcal{T} .

Lemma 8 *If $node(x) = node(y)$ is a *P*-node that is not a leaf, then (x, y) is not an interval edge.*

Proof: Let P be the considered common *P*-node, and assume it has j subtrees for some $j \geq 1$. If $j \geq 2$, then by Lemma 3i let $z_1 \in V_1$ and $z_2 \in V_2$. Clearly, there is no edge between z_1 and z_2 and both x and y are over z_1 and z_2 . Hence, Observation 6 implies that (x, y) is not an interval edge. Thus, P has exactly one subtree, and according to Lemma 3h, its root has to be a *Q*-node, see Figure 4. Moreover, Lemma 3a implies, that the first and last sections of a *Q*-node have nonempty subtrees. Let z_1 belong to the first subtree, and z_2 belong to the last one. Yet again, conditions of the Observation 6 are satisfied, so (x, y) is not an interval edge. \square

Lemma 9 *If $node(x) = node(y)$ is a *P*-node that is a leaf, then (x, y) is an interval edge. Moreover, there is a linear time algorithm that produces a string representation for the graph $G - (x, y)$.*

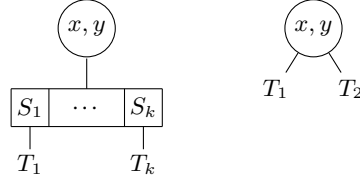


Fig. 4: Removing an edge from a P -node that is not a leaf leads to an induced C_4 cycle.

Proof: Without loss of generality assume that $x < y$, and consider the canonical string \mathcal{S} for the MPQ -tree \mathcal{T} . Clearly, \mathcal{S} is of the form $\mathcal{S} = L A x B y C \bar{C} y \bar{B} x \bar{A} R$, see Figure 5. In order to remove the edge (x, y) we find the first and the last occurrence of x in \mathcal{S} . Then, until x does not occupy two consecutive positions, we swap $\mathcal{S}[i]$ with $\mathcal{S}[i + 1]$, and $\mathcal{S}[j]$ with $\mathcal{S}[j - 1]$, where i denotes the first occurrence of x and j denotes the second. Next, we do the same for y , and as a result we get a string such that both y 's are next to each other and are surrounded by both x 's, see Figure 5c. Finally, we swap the first occurrence of y with the second occurrence of x , effectively removing the edge (x, y) . Clearly, this procedure runs in $O(n)$ time, but we have to ensure that it does not add or remove any other edge. Note that, all modifications are performed in a substring of \mathcal{S} that represents a clique, and is of the form $z_1 z_2 \dots z_k z_k \dots z_2 z_1$. Hence, each swapping operation - except the last one - swapped either two left endpoints or two right endpoints. Thus, Lemma 7 ensures that no edge was added or removed during this process. Finally, during the last swap x and y occupy four consecutive indexes. Hence, the only affected vertices are x and y .

$$\begin{array}{l}
 \text{prefix} \quad \text{PNode} \quad \text{suffix} \\
 \hline
 a) \ L | A x B y C \bar{C} y \bar{B} x \bar{A} | R \\
 b) \ L | A B y C x x \bar{C} y \bar{B} \bar{A} | R \\
 c) \ L | A B C x y y x \bar{C} \bar{B} \bar{A} | R \\
 d) \ L | A B C x x y y \bar{C} \bar{B} \bar{A} | R
 \end{array}$$

Fig. 5: Removing an edge from a leaf P -node.

□

4.2 x and y belong to the same Q -node

The next case is when both vertices belong to the same Q -node. Here we show that (x, y) is an interval edge if and only if x and y have exactly one common section and the subtree of this section represents a clique (possibly empty).

Lemma 10 *If $\text{node}(x) = \text{node}(y)$ is a Q -node, then (x, y) is not an interval edge if:*

1. x and y have more than one common section in $\text{node}(x) = \text{node}(y)$, or
2. a subtree of the common section does not represent a clique (possibly empty).

Proof: Assume that there is a common section S_i and its nonempty subtree T_i that does not represent a clique. Hence, there are at least two vertices z_1 and z_2 in V_i such that there is no edge between them, otherwise T_i would represent a clique. Thus, Observation 6 implies that (x, y) is not an interval edge. Moreover, if there are two common sections S_i and S_j such that both of them have nonempty subtrees T_i and T_j respectively, then we may choose $z_1 \in V_i$ and $z_2 \in V_j$ and use the same argument. This proves that common sections have empty subtrees except at most one which represents a clique. Now, assume that there is more than one common section ie. $S_i, S_{i+1}, \dots, S_{j-1}, S_j$, and without loss of generality S_i has an empty subtree. Lemma 3f implies that there is a vertex z_1 for which the section S_i is the last one (z_1 does not belong to sections S_{i+1}, \dots, S_j). Notice that $z_1 \notin \{x, y\}$, otherwise $j = i$. If there is a nonempty subtree T_a for some $i < a \leq j$, then we choose $z_2 \in V_a$, and Observation 6 leads to an induced chordless cycle. Hence, all common subtrees are empty and Lemma 3f gives us a vertex z_2 for which S_j is the first section. Observation 6 for vertices x, y, z_1 and z_2 finishes the proof. \square

Lemma 11 *If $node(x) = node(y)$ is a Q-node, x and y have exactly one common section S_i in it, and the subtree T_i represents a clique, then (x, y) is an interval edge. Moreover, there is a linear time algorithm that produces a string representation for the graph $G - (x, y)$.*

Proof: Again, we assume that $x < y$ and consider the canonical string \mathcal{S} for the MPQ-tree \mathcal{T} , but in this case \mathcal{S} has a more complex form than in the Lemma 9. In fact, it is of the form $P_1xP_2L_1yL_2V_i\bar{V}_iR_1xR_2S_1yS_2$, where $L_1 \cup L_2$ represents the left endpoints of vertices from S_i , $R_1 \cup R_2$ represents the right endpoints of vertices from S_i , and $V_i \cup \bar{V}_i$ represents a clique from the subtree, see Figure 6. In order to remove the edge (x, y) , at first we need to determine for each element in \mathcal{S} whether it represents the left or the right endpoint. It can be easily done in $O(n)$, since all elements in \mathcal{S} belong to the set $\{1, \dots, n\}$. The next phase swaps the first occurrence of y with its successor until the next element represents a right endpoint. Analogously, we swap the second occurrence of x with its predecessor until the next element represents a left endpoint. Clearly, because of Lemma 7 we did not add or remove any edge till this moment, and \mathcal{S} looks like in Figure 6c. Finally, we can remove the edge (x, y) by swapping the first occurrence of y with the second occurrence of x , that in fact occupy consecutive positions in \mathcal{S} .

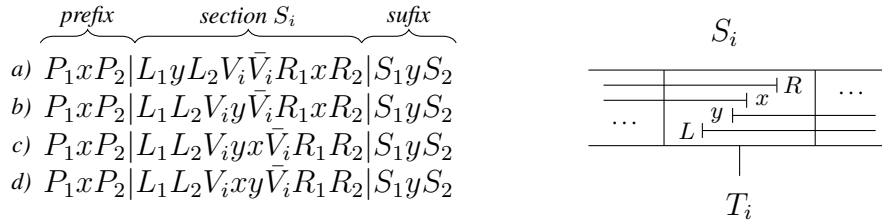


Fig. 6: Removing an edge from the same Q-node.

\square

4.3 x and y belong to different nodes.

In the previous two subsections, we provided a full classification for the cases where x and y belong to the same node in \mathcal{T} . In this subsection, we consider the cases where x and y belong to different nodes in

\mathcal{T} . Before we present our results in those cases, we introduce a terminology that allows us to describe a relative position of $node(x)$ and $node(y)$ in \mathcal{T} .

For a Q -node with k sections S_1, \dots, S_k , we say that the section S_a is a *central* section if $1 < a < k$. Sections S_1 and S_k are called *non-central* sections. For a vertex v we say that the section S_a is a *v-central* section if $l(v) < a < r(v)$. Sections $S_{l(v)}$ and $S_{r(v)}$ are *v-non-central* sections. For every two vertices x and y , there is exactly one path in the tree \mathcal{T} between $node(x)$ and $node(y)$. We say that this unique path is an $\langle x, y \rangle$ -*tree-path* if x is over y in \mathcal{T} . For an $\langle x, y \rangle$ -tree-path: $node(x) = n_1 - n_2 - \dots - n_t = node(y)$, we say that this path *goes through the central section* if there is a Q -node n_i for $1 < i < t$ such that n_{i+1} belongs to a subtree of some central section of n_i , see Figure 10. Moreover, we say that the path *starts in a central section* if n_1 is a Q -node, and n_2 belongs to a subtree of some x -central section. Analogously, it *starts in a non-central section* if n_2 belongs to a subtree of some x -non-central section. We also say that an $\langle x, y \rangle$ -tree-path *starts in a P-node*, if $node(x)$ is a P -node, and *ends in a P-node* if $node(y)$ is a P -node. Analogous definitions apply to Q -nodes. Finally, we say that an $\langle x, y \rangle$ -tree-path is *almost rotatable* if it does not go through the central section, and ends in a P -node that is a leaf. An $\langle x, y \rangle$ -tree-path is *rotatable* if it is almost rotatable, and either starts in a P -node, or starts in a non-central section. Intuitively, if an $\langle x, y \rangle$ -tree-path is almost rotatable, then we are able to rotate all the nodes on the path $n_2 - \dots - n_t$ in such a way that y is the leftmost vertex in the subtree which root is n_2 .

Now we are ready to characterize interval edges in the case where x and y belong to different nodes in \mathcal{T} . We prove that if $\langle x, y \rangle$ -tree-path is rotatable, then (x, y) is an interval edge. Unfortunately, the reverse implication is not true and sometimes $\langle x, y \rangle$ -tree-path is not rotatable, but (x, y) is still an interval edge. We shall prove that this happens only for almost rotatable $\langle x, y \rangle$ -tree-paths satisfying some additional, and quite technical, conditions. First, we show how to compute a string representation for an interval graph $G - (x, y)$ if $\langle x, y \rangle$ -tree-path is rotatable.

Lemma 12 *If an $\langle x, y \rangle$ -tree-path is rotatable, then (x, y) is an interval edge. Moreover, there is a linear time algorithm that produces a string representation for the graph $G - (x, y)$.*

Proof: In order to remove the edge (x, y) , we do not produce a canonical string \mathcal{S} immediately. At first, we need to adjust the tree \mathcal{T} using some preprocessing. If $node(x)$ is a Q -node, then we rotate $node(x)$ so that $\langle x, y \rangle$ -tree-path starts in the section $S_{l(x)}$. Then, we rotate \mathcal{T} so that the path from $node(x)$ to $node(y)$ goes through the leftmost children of P -nodes and the leftmost sections of Q -nodes. Let \mathcal{T}' be the result of the described adjustment, and let \mathcal{S} be a string representation of \mathcal{T}' . Clearly, \mathcal{S} is of form $LxAByC\bar{C}y\bar{B}DxR$, see Figure 7, and both occurrences of y lay in between occurrences of x in \mathcal{S} . Node $node(y)$ is a P -node that is a leaf, so we start with moving the first occurrence of y to the right and the second occurrence of y to the left until they both meet, as in the Lemma 9. All vertices that lay between the first occurrence of x and the first occurrence of y represent the left endpoints. Hence, we can swap the first occurrence of x with its successor until it meets the second occurrence of y . Lemma 7 ensures that no edge is added or removed during this process. Finally, moving x once more to the right swaps the left endpoint of x with the right endpoint of y effectively removing the edge (x, y) . □

Now, we consider those $\langle x, y \rangle$ -tree-paths that are not rotatable, but are almost rotatable. Hence, all those paths start in some x -central section of some Q -node. We denote S_1, \dots, S_k the sections of considered Q -node, and S_a the x -central section where the $\langle x, y \rangle$ -tree-path starts. As we already mentioned, sometimes

$$\begin{array}{l}
\text{a) } \overbrace{LxA|ByCCy\bar{B}}^{\text{prefix node}(y) \text{ suffix}}|DxR \\
\text{b) } LxA|BCyC\bar{B}|DxR \\
\text{c) } \overbrace{LA|BCyxy\bar{C}\bar{B}}^{\text{prefix node}(y)+x \text{ suffix}}|DxR \\
\text{d) } LA|BCyC\bar{D}|DxR
\end{array}$$

Fig. 7: Removing an edge (x, y) in the case where $\langle x, y \rangle$ -tree-path is rotatable.

in that case the edge (x, y) is an interval edge. The next two lemmas establish the required conditions for that to happen.

Lemma 13 *If an $\langle x, y \rangle$ -tree-path is almost rotatable, starts in a central section S_a , y has no neighbor in a subtree T_a and:*

1. $\exists_{1 < b \leq l(x)} : S_a \setminus \{x\} \subset S_b$ and $S_{b-1} \cap S_b \subset S_a$, or
2. $\exists_{r(x) \leq b < k} : S_a \setminus \{x\} \subset S_b$ and $S_b \cap S_{b+1} \subset S_a$, or
3. $S_a \setminus \{x\} \subset S_1$, or
4. $S_a \setminus \{x\} \subset S_k$,

then (x, y) is an interval edge. Moreover, there is a linear time algorithm that produces a string representation for the graph $G - (x, y)$.

Proof: Proofs for all four cases are very similar, so we show only the proof for the first case. Assume that there is a $1 < b \leq l(x)$ such that $S_a \setminus \{x\} \subset S_b$ and $S_{b-1} \cap S_b \subset S_a$. As in the proof of Lemma 12, before we produce the string representation \mathcal{S} , we need to make some adjustments in the tree \mathcal{T} . At first, we insert a new section $S^* = S_a \setminus \{x\}$ in between sections S_{b-1} and S_b , see Figure 8. Clearly, an insertion of a new section does not remove the old edges, but it might add some new ones. However, the section S^* is a subset of an already existing section, so it is not the case. Moreover, the condition that each vertex belongs to the sequence of consecutive sections is preserved. That's because we assumed that $S_{b-1} \cap S_b \subset S_a$ and $S_a \setminus \{x\} \subset S_b$. Next, we remove the vertex y from the subtree T_a . We also define a subtree T^* of the section S^* to be a single P -node containing y . Note that y has no neighbor in T_a , so no edges were removed, except the edge (x, y) . Thus, we obtained a tree \mathcal{T}' that encodes the graph $G - (x, y)$. In order to get the string representation for the graph $G - (x, y)$ it is enough to compute the string representation for \mathcal{T}' . We proved only the first case, but observe that the second case is symmetric, and cases 3 and 4 are the corner cases, so in the third case we simply insert the section S^* before S_1 and in the fourth case we insert S^* after S_k . □

Lemma 14 *If an $\langle x, y \rangle$ -tree-path is almost rotatable, starts in a central section S_a , y has a neighbor in a subtree T_a and:*

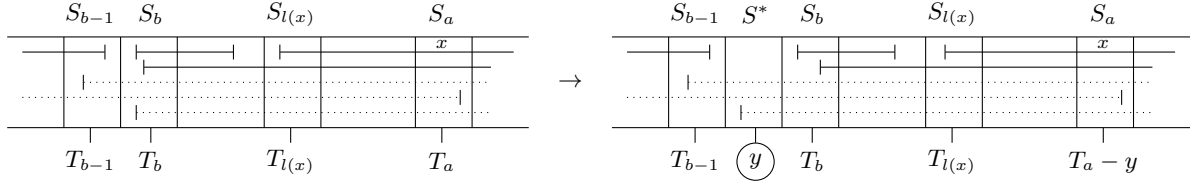


Fig. 8: Removing an edge (x, y) in the case where $\langle x, y \rangle$ -tree-path is almost rotatable, starts in some x -central section S_a and y does not have a neighbor in a subtree T_a . (Case 1)

1. $l(x) > 1$ and $S_a \setminus \{x\} \subset S_{l(x)}$ and $S_{l(x)-1} \cap S_{l(x)} \subset S_a$, or
2. $r(x) < k$ and $S_a \setminus \{x\} \subset S_{r(x)}$ and $S_{r(x)} \cap S_{r(x)+1} \subset S_a$, or
3. $l(x) = 1$ and $S_a \setminus \{x\} \subset S_1$, or
4. $r(x) = k$ and $S_a \setminus \{x\} \subset S_k$,

then (x, y) is an interval edge. Moreover, there is a linear time algorithm that produces a string representation for the graph $G - (x, y)$.

Proof: The proof of this lemma is similar to the proof of Lemma 13, but now y has at least one neighbor in T_a , so we cannot simply remove y from T_a . That's also the reason why conditions of this lemma are more strict than in the previous one. Yet again, we are going to prove only the first case, so assume that $S_a \setminus \{x\} \subset S_{l(x)}$ and $S_{l(x)-1} \cap S_{l(x)} \subset S_a$. Instead of inserting a new section, we remove the section S_a and insert it in between sections $S_{l(x)-1}$ and $S_{l(x)}$, see Figure 9. Clearly, no edge is added or removed, and the condition that each vertex belongs to the sequence of consecutive sections is preserved. Now, we remove x from the section S_a . This, removes the edge (x, y) , but also all the edges (x, v) , where $v \in V_a$. In the next phase, we are going to restore those edges. In order to do it, at first we rotate the subtree T_a in such a way that the path from $node(x)$ to $node(y)$ goes through the leftmost children of P -nodes and the leftmost sections of Q -nodes. Then, we compute the string representation \mathcal{S} of the modified tree, and move the first occurrence of y to the right and the second occurrence of y to the left until both meet, as in the Lemma 9. After this operation is done, y is the leftmost vertex of the tree T_a - its right endpoint appears first in the string representation of T_a . In order to restore all the removed edges except (x, y) , we move the first occurrence of x in \mathcal{S} to the left, until its predecessor is the second occurrence of y . Clearly, this procedure restores all the removed edges except (x, y) . \square

The above two lemmas show the only cases where $\langle x, y \rangle$ -tree-path is not rotatable, but (x, y) is an interval edge. Now, we are going to show that if $\langle x, y \rangle$ -tree-path is not rotatable, and the conditions of those lemmas are not satisfied, then (x, y) is not an interval edge. At first, we prove that if $\langle x, y \rangle$ -tree-path does not end in a P -node that is a leaf, then (x, y) is not an interval edge.

Lemma 15 *If an $\langle x, y \rangle$ -tree-path ends in a Q -node, then (x, y) is not an interval edge.*

Proof: If y belongs only to central sections ($y \notin S_1 \cup S_k$), then Lemma 3f implies that there are vertices $z_1 \in (S_{l(y)} \cup V_{l(y)}) \setminus S_{l(y)+1}$ and $z_2 \in (S_{r(y)} \cup V_{r(y)}) \setminus S_{r(y)-1}$. If y belongs to S_1 (or S_k), then just

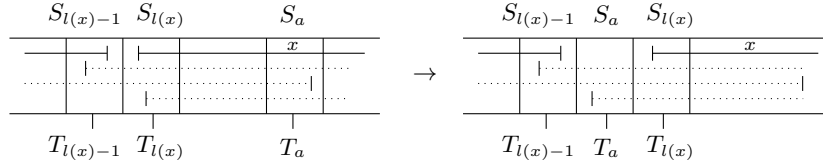


Fig. 9: Removing an edge (x, y) in the case where $\langle x, y \rangle$ -tree-path is almost rotatable, starts in some x -central section S_a , and y has some neighbor in a subtree T_a . (Case 1)

take $z_1 \in V_1$ (or $z_2 \in V_k$). Clearly, there is no edge between z_1 and z_2 , and both x and y are over z_1 and z_2 . Thus, Observation 6 finishes the proof. \square

Lemma 16 *If an $\langle x, y \rangle$ -tree-path ends in a P-node that is not a leaf, then (x, y) is not an interval edge.*

Proof: The same argument as in Lemma 8. \square

Now, we are going to prove that if an $\langle x, y \rangle$ -tree-path goes through the central section, then (x, y) is not an interval edge. First, we define a *nested path*. Consider a Q -node with k sections S_1, \dots, S_k . For a clarity let $S_0 = S_{k+1} = \emptyset$, and say that a set of vertices $\mathcal{P}_{i,j} = (S_i \cup \dots \cup S_j) \setminus (S_{i-1} \cup S_{j+1})$ is a *nested path*, if for every $i \leq a < j$ there is at least one vertex $v_a \in \mathcal{P}_{i,j}$ that belongs to $S_a \cap S_{a+1}$. We call it a nested path, because vertices $v_i, v_{i+1}, \dots, v_{j-1}$ in that order form (possibly not simple) path in the graph G , and all of them are contained in the sections S_i, \dots, S_j . In the following lemma, we show that if $\langle x, y \rangle$ -tree-path goes through a central section of some Q -node, then we can find an asteroidal triple in the graph $G - (x, y)$. Nested paths help us to find this triple.

Lemma 17 *If $\langle x, y \rangle$ -tree-path goes through the central section, then (x, y) is not an interval edge.*

Proof: Assume that the $\langle x, y \rangle$ -tree-path goes through some central section S_i of a Q -node Q , and let S_1, \dots, S_k be the sections of Q . Subtrees of the first and last section are nonempty, so let $z_1 \in V_1$ and $z_2 \in V_k$. We show that vertices y, z_1 and z_2 form an asteroidal triple in the graph $G - (x, y)$.

There is no edge (x, y) , so a path $z_1 - x - z_2$ avoids the neighborhood of y . To prove that there is a path between z_1 and y that avoids the neighborhood of z_2 , we show that there is a nested path $\mathcal{P}_{1,k-1}$, and so the shortest path of form $z_1 - \mathcal{P}_{1,k-1} - y$ fulfill our requirements. Let $\mathcal{P}_{1,j}$ be a nested path with maximum $j < k$. Clearly, such a path exists. Otherwise, either S_1 is empty, or all vertices that belong to S_1 belong to all sections. In both cases, properties listed in Lemma 3 are violated. Moreover, if j is less than $k - 1$, then either $S_j \cap S_{j+1} = \emptyset$, or $S_j \cap S_{j+1} \subset S_k$. Again, this contradicts Lemma 3, and we are done. A path from z_2 to y that avoids the neighborhood of z_1 is constructed in a similar way. \square

A consequence of the previous three lemmas is the following corollary.

Corollary 18 *If an $\langle x, y \rangle$ -tree-path is not almost rotatable, then (x, y) is not an interval edge.*

We are almost done with the classification. In Lemma 12 we proved that if an $\langle x, y \rangle$ -tree-path is rotatable, then (x, y) is always an interval edge. On the other hand, in Lemmas 15, 16 and 17 we considered $\langle x, y \rangle$ -tree-paths that are not almost rotatable, and showed that for such paths (x, y) is not an interval edge. Hence, the only remaining cases are the $\langle x, y \rangle$ -tree-paths that are almost rotatable, but not rotatable. In Lemmas 13

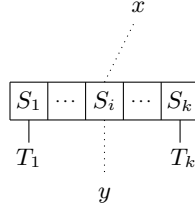


Fig. 10: An $\langle x, y \rangle$ -tree-path which goes through a central section S_i of some Q -node.

and 14 we investigated such paths, and proved that under some additional conditions (x, y) is an interval edge. Now we show that if an $\langle x, y \rangle$ -tree-path is almost rotatable, but is not rotatable, and conditions of Lemmas 13 and 14 are not satisfied, then (x, y) is not an interval edge. This case is the hardest one, and before we prove it, we need to prove two auxiliary lemmas.

Lemma 19 *Let S_1, \dots, S_k be the sections of some Q -node in an MPQ-tree \mathcal{T} , and for the simplicity assume that $S_0 = S_{k+1} = \emptyset$. For every pair of indices $(a, b) \neq (1, k)$, $a < b$ there is a vertex $v \in ((S_{a-1} \cap S_a) \setminus S_b) \cup ((S_b \cap S_{b+1}) \setminus S_a)$.*

Proof: By contradiction, assume that for some pair of indices (a, b) there is no such vertex, so each vertex $u \in S_a \cup \dots \cup S_b$ is either contained in S_a, \dots, S_b , or belongs to $S_a \cap \dots \cap S_b$. Without loss of generality assume that $b < k$. Each vertex belongs to at least two sections. Hence, no vertex has its right endpoint in S_a or left endpoint in S_b , and Lemma 3 implies that V_a, V_b and V_k are not empty. Let $v_a \in V_a, v_b \in V_b, v_k \in V_k$, and consider any maximal cliques C_a, C_b and C_k such that $v_a \in C_a, v_b \in C_b$ and $v_k \in C_k$. Note that, \mathcal{T} encodes only those orderings of maximal cliques in which either $C_a < C_b < C_k$ or $C_k < C_b < C_a$. Now, consider a modified tree \mathcal{T}' in which we reverse the order of sections S_a, \dots, S_b . Clearly, because of our assumptions, \mathcal{T}' represents the same interval graph as \mathcal{T} , but \mathcal{T}' encodes an ordering $C_b < C_a < C_k$, which is not encoded by \mathcal{T} . Thus, \mathcal{T} is not a valid MPQ-tree, and we are done. \square

Lemma 20 *If an $\langle x, y \rangle$ -tree-path starts in a central section S_a , and there is a vertex $q \in S_a \setminus (S_{l(x)} \cup S_{r(x)})$, then (x, y) is not an interval edge.*

Proof: By contradiction, assume that (x, y) is an interval edge, Let P_{l_1, r_1} for $r_1 < a$ be a nested path that intersects q and have the smallest l_1 . Analogously, let P_{l_2, r_2} for $a < l_2$ be a nested path that intersects q and have the biggest r_2 . Notice that these paths may not exist. For instance, if q has its right endpoint in S_a , then P_{l_2, r_2} does not exist.

At first, we consider the case where both paths do not exist, and prove that there is an asteroidal triple in the graph $G - (x, y)$. By Lemma 19, without loss of generality, there is a vertex $v \in (S_{r(q)} \cap S_{r(q)+1}) \setminus S_{l(q)}$. Both paths do not exist, so v has to belong to S_a . Moreover, Lemma 3f gives two vertices $v_L \in (S_{l(q)} \cup V_{l(q)}) \setminus S_{l(q)+1}$, and $v_R \in (S_{r(q)+1} \cup V_{r(q)+1}) \setminus S_{r(q)}$. Thus, the paths: $v_L - x - v_R, v_L - q - y$, and $v_R - v - y$ certify the asteroidal triple $\{v_L, v_R, y\}$, and we are done in this case.

Now, assume that both paths exist, and put $L = \max\{l_1, l(x)\}$ and $R = \min\{r_2, r(x)\}$. Lemma 3f implies that there is a vertex v_L which has its right endpoint in S_L or $v_L \in V_L$. Analogously, define a

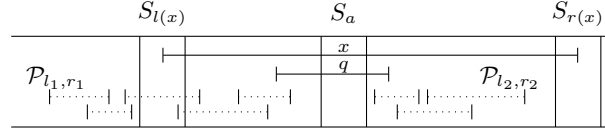


Fig. 11: Paths \mathcal{P}_{l_1, r_1} and \mathcal{P}_{l_2, r_2} .

vertex v_R for the section S_R . Clearly, both v_L and v_R are neighbors of x , but they do not belong to the neighborhood of q . Hence, $\{v_L, v_R, y\}$ is an asteroidal triple in the graph $G - (x, y)$. To see this, consider the paths: $v_L - x - v_R$, $v_L - P_{l_1, r_1} - q - y$, and $v_R - P_{l_2, r_2} - q - y$. Thus, if both paths exist, then (x, y) is not an interval edge.

Without loss of generality, assume that \mathcal{P}_{l_1, r_1} exists, but \mathcal{P}_{l_2, r_2} does not. Lemma 19 implies that there is a vertex v such that $v \in (S_{l_1-1} \cap S_{l_1}) \setminus S_{r(q)}$, or $v \in (S_{r(q)} \cap S_{r(q)+1}) \setminus S_{l_1}$. Note that, at this point v and x might be the same vertex. We assumed that right path does not exist, so $S_{r(q)} \cap S_{r(q)+1} \subset S_a$. Moreover, l_1 is the smallest index such that there is a nested path P_{l_1, r_1} for $r_1 < a$ that intersects q . Hence, $S_{l_1-1} \cap S_{l_1} \subset S_a$, and so in both cases v belongs to S_a . Lemma 3f gives three vertices: $v_L \in (S_{l_1} \cup V_{l_1}) \setminus S_{l_1-1}$, $v_R \in (S_{r(q)} \cup V_{r(q)}) \setminus S_{r(q)-1}$, and $v_{R+1} \in (S_{r(q)+1} \cup V_{r(q)+1}) \setminus S_{r(q)}$. Note that, both v_R and v_{R+1} are neighbors of x , and both v_L and v_{R+1} are not neighbors of q .

If $l_1 < l(x)$, then the paths: $v_L - P_{l_1, r_1} - x - v_{R+1}$, $v_L - P_{l_1, r_1} - q - y$, and $v_{R+1} - x - q - y$ certify the asteroidal triple $\{v_L, v_{R+1}, y\}$. Thus, we conclude that $l(x) \leq l_1 < l(q)$, which means that the whole path P_{l_1, r_1} is contained in x . Moreover, v and x are different vertices, and v_L is a neighbor of x . If $v \in (S_{l_1-1} \cap S_{l_1}) \setminus S_{r(q)}$, then there is an asteroidal triple $\{v_L, v_R, y\}$ certified by paths: $v_L - x - v_R$, $v_L - v - y$ and $v_R - q - y$. On the other hand, if $v \in (S_{r(q)} \cap S_{r(q)+1}) \setminus S_{l_1}$, then $\{v_L, v_{R+1}, y\}$ is an asteroidal triple certified by paths: $v_L - x - v_{R+1}$, $v_L - P_{l_1, r_1} - q - y$, and $v_{R+1} - v - y$. Thus, in this case the edge (x, y) is also not an interval edge, and we are done. \square

Finally, we are ready to prove the last two negative results on interval edges.

Lemma 21 *If an $\langle x, y \rangle$ -tree-path starts in a central section S_a , y has a neighbor in subtree T_a , and:*

1. $i > 1 \Rightarrow (\exists_{z_i \neq x} : z_i \in S_a \setminus S_i \text{ or } \exists_{\bar{z}_i} : \bar{z}_i \in (S_{i-1} \cap S_i) \setminus S_a)$, and
2. $j < k \Rightarrow (\exists_{z_j \neq x} : z_j \in S_a \setminus S_j \text{ or } \exists_{\bar{z}_j} : \bar{z}_j \in (S_j \cap S_{j+1}) \setminus S_a)$, and
3. $i = 1 \Rightarrow \exists_{z_1 \neq x} : z_1 \in S_a \setminus S_1$, and
4. $j = k \Rightarrow \exists_{z_k \neq x} : z_k \in S_a \setminus S_k$,

where $i = l(x)$ and $j = r(x)$, then (x, y) is not an interval edge.

Proof: By contradiction, assume that (x, y) is an interval edge, and let z be the neighbor of y in T_a . If both vertices z_i and z_j exist, and $z_i = z_j$, then $z_i \in S_a \setminus (S_i \cup S_j)$, and Lemma 20 leads to a contradiction. Hence, if both of them exist, then $z_i \neq z_j$. But, in this case we can find an asteroidal triple $\{v_L, v_R, y\}$, where $v_L \in (S_i \cup V_i) \setminus S_{i+1}$ and $v_R \in (S_j \cup V_j) \setminus S_{j-1}$, certified by paths: $v_L - x - v_R$, $v_L - z_j - y$, and $v_R - z_i - y$.

Thus, without loss of generality z_j does not exist, and there is a vertex \bar{z}_j which belongs to $(S_j \cap S_{j+1}) \setminus S_a$. If \bar{z}_i also exists, then consider vertices $v_L \in (S_{i-1} \cup V_{i-1}) \setminus S_i$, and $v_R \in (S_{j+1} \cup V_{j+1}) \setminus S_j$ that are given by Lemma 3f, and notice that paths: $v_L - \bar{z}_i - x - \bar{z}_j - v_R$, $v_L - \bar{z}_i - x - z - y$, and $v_R - \bar{z}_j - x - z - y$ certify the asteroidal triple $\{v_L, v_R, y\}$.

Hence, \bar{z}_i does not exist, and the only remaining case is where \bar{z}_j and z_i exist. Note that if z_i does not belong to S_j , then $z_i \in S_a \setminus (S_i \cup S_j)$ and Lemma 20 gives a contradiction. Hence, $z_i \in S_j$ and there is an edge between z_i and \bar{z}_j . Lemma 3f gives vertices $v_L \in (S_i \cup V_i) \setminus S_{i+1}$ and $v_R \in (S_{j+1} \cup V_{j+1}) \setminus S_j$, and the asteroidal triple $\{v_L, v_R, y\}$ is certified by paths: $v_L - x - \bar{z}_j - v_R$, $v_L - x - z - y$, and $v_R - \bar{z}_j - z_i - y$. \square

Lemma 22 *If an $\langle x, y \rangle$ -tree-path starts in a central section S_a , y does not have a neighbor in subtree T_a , and:*

1. $\forall_{1 < b \leq l(x)} : \exists_{z_b \neq x} : z_b \in S_a \setminus S_b$ or $\exists_{\bar{z}_b} : \bar{z}_b \in (S_{b-1} \cap S_b) \setminus S_a$, and
2. $\forall_{r(x) \leq b < k} : \exists_{z_b \neq x} : z_b \in S_a \setminus S_b$ or $\exists_{\bar{z}_b} : \bar{z}_b \in (S_b \cap S_{b+1}) \setminus S_a$, and
3. $\exists_{z_1 \neq x} : z_1 \in S_a \setminus S_1$, and
4. $\exists_{z_k \neq x} : z_k \in S_a \setminus S_k$,

then (x, y) is not an interval edge.

Proof: By contradiction, assume that (x, y) is an interval edge. Let p_L be the longest sequence $z_{l(x)}, z_{l(x)-1}, \dots, \bar{z}_L$, and p_R be the longest sequence $z_{r(x)}, z_{r(x)+1}, \dots, \bar{z}_R$. If both sequences are empty, then there are vertices $z_{l(x)} \in S_a \setminus S_{l(x)}$ and $z_{r(x)} \in S_a \setminus S_{r(x)}$. But, the same argument as in Lemma 21, shows that there is an asteroidal triple in the graph $G - (x, y)$. Thus, without loss of generality p_L is not empty. If both sequences are not empty, then Lemma 3f gives vertices $v_L \in (S_{L-1} \cup V_{L-1}) \setminus S_L$ and $v_R \in (S_{R+1} \cup V_{R+1}) \setminus S_R$. Moreover, both sequences are the longest possible, so there are vertices $z_{L-1} \in S_a \setminus S_{L-1}$ and $z_{R+1} \in S_a \setminus S_{R+1}$. Thus, the paths $v_L - p_L - x - p_R - v_R$, $v_L - p_L - x - z_{R+1} - y$, and $v_R - p_R - x - z_{L-1} - y$ certify the asteroidal triple $\{v_L, v_R, y\}$. Hence, p_R is empty, but p_L is not. In that case, let v_R be a vertex that belongs to $S_{r(x)} \cup V_{r(x)}$, but does not belong to $S_{r(x)-1}$. Sequence p_R is empty, so there is a vertex $z_{l(x)}$ that belongs to S_a , but does not belong to $S_{r(x)}$. Yet, again we can find an asteroidal triple $\{v_L, v_R, y\}$ certified by paths: $v_L - p_L - x - v_R$, $v_L - p_L - x - z_{r(x)} - y$, and $v_R - x - z_{L-1} - y$. \square

Finally, we are done with the classification of interval edges. For each edge (x, y) which is an interval edge, we provided a linear time algorithm that produces a string representation for the graph $G - (x, y)$, see Lemmas 9, 11, 12, 13, and 14. Moreover, for every edge (x, y) that is not an interval edge we presented structures certifying that $G - (x, y)$ is not an interval graph, see Lemmas 8, 10, 15, 16, 17, 21 and 22. The consequence of all those lemmas is the following theorem.

Theorem 23 *There is a linear time algorithm, that for every MPQ-tree \mathcal{T} representing graph G , and every interval edge (x, y) , produces a string representation S' for the graph $G - (x, y)$.*

5 Listing interval edges

In this section we present an efficient algorithm that for a given *MPQ*-tree \mathcal{T} lists all interval edges of the graph represented by \mathcal{T} . Let S_1, \dots, S_k be the sections of a Q -node Q . For $b \in \{2, \dots, k\}$, denote $f_r(b)$ the maximal index of a section such that $S_{b-1} \cap S_b \subset S_{f_r(b)}$. Analogously, for $b \in \{1, \dots, k-1\}$, denote $f_l(b)$ the minimal index of a section such that $S_b \cap S_{b+1} \subset S_{f_l(b)}$. For every vertex x which belongs to Q , and every $l(x) \leq a \leq r(x)$ let:

$$L^*(x, a) = \begin{cases} 1 & , S_a = \{x\} \\ \max_{v \in S_a \setminus \{x\}} l(v) & , S_a \setminus \{x\} \neq \emptyset \end{cases} \quad R^*(x, a) = \begin{cases} k & , S_a = \{x\} \\ \min_{v \in S_a \setminus \{x\}} r(v) & , S_a \setminus \{x\} \neq \emptyset \end{cases}$$

In other words, if $S_a \neq \{x\}$, then $L^*(x, a) = i$ if S_i is the rightmost section such that there is a vertex $v \neq x$ which belongs to S_a and v has its left endpoint in S_i . Now, we are ready to express the inclusion conditions on Q -node sections in terms of the functions f_l, f_r, L^* , and R^* .

Observation 24 *In respect to the above definitions, we have the following equivalences:*

1. $\forall_{1 < b \leq a} : S_{b-1} \cap S_b \subset S_a \Leftrightarrow f_r(b) \geq a$.
2. $\forall_{a \leq b < k} : S_b \cap S_{b+1} \subset S_a \Leftrightarrow f_l(b) \leq a$.
3. $\forall_{l(x) \leq a \leq r(x)} : S_a \setminus \{x\} \subset S_b \Leftrightarrow b \in [L^*(x, a), R^*(x, a)]$.

Using those equivalences, we are able to reformulate the conditions of Lemmas 13 and 14.

Observation 25 *For an $\langle x, y \rangle$ -tree-path that starts in a central section S_a , the conditions (1), (2), (3), and (4) of Lemma 13 are equivalent to the following:*

1. $L^*(x, a) \leq l(x)$ and $\min \{l(x), R^*(x, a)\} > 1$ and $f_r(\min \{l(x), R^*(x, a)\}) \geq a$, or
2. $R^*(x, a) \geq r(x)$ and $\max \{r(x), L^*(x, a)\} < k$ and $f_l(\max \{r(x), L^*(x, a)\}) \leq a$, or
3. $L^*(x, a) = 1$, or
4. $R^*(x, a) = k$.

Proof: Note that, conditions (3) and (4) translate directly, and conditions (1) and (2) are symmetrical. Hence, we only show the equivalence of the first cases. At first, we show that if there is $1 < b \leq l(x)$ such that $S_a \setminus \{x\} \subset S_b$ and $S_{b-1} \cap S_b \subset S_a$, then $L^*(x, a) \leq l(x)$ and $f_r(\min \{l(x), R^*(x, a)\}) \geq a$. From Observation 24 we have $b \in [L^*(x, a), R^*(x, a)]$ and $f_r(b) \geq a$. Thus, $L^*(x, a) \leq l(x)$, and $1 < b \leq \min \{l(x), R^*(x, a)\}$. Note that f_r is a non-decreasing function, so $f_r(b) \geq a$, implies $f_r(\min \{l(x), R^*(x, a)\}) \geq a$, and we are done.

Now, we take $b = \min \{l(x), R^*(x, a)\}$, and show that if $b > 1$, $f_r(b) \geq a$, and $L^*(x, a) \leq l(x)$, then $S_{b-1} \cap S_b \subset S_a$ and $S_a \setminus \{x\} \subset S_b$. Clearly, $1 < b \leq l(x)$ and $f_r(b) \geq a$, so $S_{b-1} \cap S_b \subset S_a$. Hence, the only thing we need to show is $S_a \setminus \{x\} \subset S_b$. According to Observation 24 it is equivalent to $b \in [L^*(x, a), R^*(x, a)]$. It is easy to see that for every x and a we have $L^*(x, a) \leq R^*(x, a)$. Thus, the interval $[L^*(x, a), R^*(x, a)]$ is never empty. If $b = R^*(x, a)$, then we are done, so assume that $b = l(x)$. But, in this case $L^*(x, a) \leq l(x) = b \leq R^*(x, a)$, and we are done too. \square

A very similar proof applies to the following observation, so we leave it to the reader.

Observation 26 For an $\langle x, y \rangle$ -tree-path that starts in a central section S_a , the conditions (1), (2), (3), and (4) of Lemma 14 are equivalent to the following:

1. $l(x) > 1$ and $l(x) \in [L^*(x, a), R^*(x, a)]$ and $f_r(l(x)) \geq a$, or
2. $r(x) < k$ and $r(x) \in [L^*(x, a), R^*(x, a)]$ and $f_l(r(x)) \leq a$, or
3. $l(x) = 1$ and $L^*(x, a) = 1$, or
4. $r(x) = k$ and $R^*(x, a) = k$.

Observations 25 and 26 provide fast and easy tests under the assumption that all the functions l , r , f_l , f_r , L^* and R^* are already computed. To represent functions l , r , f_l and f_r in computer's memory we use $O(n)$ -element arrays of integers. Functions L^* and R^* in their explicit forms may require $O(n^2)$ space. Hence, for performance purposes, we do not represent those functions as two-dimensional arrays. Instead, we observe that the function L^* can be defined using some one-dimensional functions L_1 and L_2 . We set $L_1(a) = \max \{l(v) : v \in S_a\}$, and denote $v_1(a)$ a vertex for which $l(v_1(a)) = L_1(a)$. We also define $L_2(a) = \max \{l(v) : v \in S_a \setminus \{v_1(a)\}\}$, or $L_2(a) = 1$ if $S_a = \{v_1(a)\}$. Using those two functions we are able to compute the function L^* in the following way:

$$L^*(x, a) = \begin{cases} L_1(a), & L_1(a) \neq l(x) \\ L_2(a), & \text{otherwise} \end{cases}$$

Analogously, we can represent the function R^* using two functions R_1 and R_2 . Now we are ready to show the interval edges enumeration algorithm.

Lemma 27 Functions l , r , f_l and f_r for all Q -nodes of the tree \mathcal{T} can be computed in $O(n \log n)$ time.

Proof: We compute these functions for each Q -node separately. Assume that we are computing the function f_r for a Q -node Q_d . Let i be the minimum index of the section containing a right endpoint of some vertex from $S_{b-1} \cap S_b$. Clearly, $S_{b-1} \cap S_b \subset S_i$ and $\neg(S_{b-1} \cap S_b \subset S_{i+1})$. Hence, $f_r(b) = i$, and in order to compute the function f_r for all b , it is enough to scan the sections of Q_d from left to right maintaining a heap of right endpoints. When algorithm enters the section S_{b+1} it adds to the heap all right endpoints of vertices that have its left endpoint in S_b , assigns $f_r(b)$ to be the minimum value stored in the heap, and removes all the endpoints of vertices that have its right endpoint in S_b . Thus, the computation of the function f_r for Q_d takes $O(n_d \log n_d)$ time, where n_d denotes the number of vertices in Q_d . As $\sum n_d \leq n$, the computation of all functions f_r takes no more than $O(n \log n)$ time. A similar argument applies to the functions f_l . \square

Lemma 28 Functions L_1 , L_2 , R_1 and R_2 for all Q -nodes of the tree \mathcal{T} can be computed in $O(n \log n)$ time.

Proof: The proof is very similar to the proof of Lemma 27. Yet again, we use a heap of right or left endpoints, but now we are not only interested in the minimal element but also in the second one. \square

Theorem 29 There is an algorithm that for a given MPQ-tree \mathcal{T} lists all its interval edges in $O(\max \{n + m, n \log n\})$ time.

Proof: The algorithm is pretty straightforward. At first, we compute all the functions $l, r, f_l, f_r, L_1, L_2, R_1$ and R_2 . Then, we inspect all P -nodes and all sections of Q -nodes listing all edges (x, y) that satisfy the conditions of Lemmas 9 or 11. For a P -node that is a leaf, we list all pairs (v_a, v_b) for $a \neq b$, while for the section S_i with subtree that is either empty or is a P -node with no children, we list the edges of the form (v_L, v_R) , where v_L is a vertex that has its right endpoint in S_i , and v_R is a vertex that has its left endpoint in S_i . Note that, an MPQ -tree has no more than $O(n)$ nodes and sections. Moreover, we have a constant time access to all vertices v_L and v_R . Thus, this phase works in $O(n + m)$.

Now, we want to find all interval edges (x, y) such that x and y belong to different nodes in \mathcal{T} , and x is over y . Lemmas 15 and 16 imply that it is enough to consider only those pair of vertices (x, y) , where y belongs to a leaf of \mathcal{T} . Hence, for each leaf L of a tree \mathcal{T} , we traverse a unique path between L and the root of \mathcal{T} , starting from L , and listing edges of the form (x, y) , where x belongs to the currently visited node, and y belongs to L . We do not list all such edges, but for each candidate we need to decide whether (x, y) is an interval edge or not. In order to do it efficiently, we keep two boolean variables: 1. does y have a neighbor in the visited subtree, and 2. does the path go through some central section. Using those two variables and precomputed functions, we can decide whether (x, y) is an interval edge in a constant time thanks to Lemma 12, and Observations 25 and 26. Thus, the time spent by the algorithm in this phase is bounded from above by the sum of lengths of all paths we have visited plus the number of tested edges. Lemma 3g, implies that on those paths there are no two consecutive empty P -nodes, so we can bound the sum of lengths of those paths by $O(m)$. Thus, the time spent by the algorithm in this phase is $O(n + m)$, and the whole algorithm works in $O(\max\{n + m, n \log n\})$ time. \square

6 Parent-Child Relationship

In this section we present the graph enumeration algorithm. We define a parent-child relationship in the same way as authors in Yamazaki et al. (2019) did using the following lemma.

Lemma 30 (Kiyomi et al. (2006)) *If $G = (V, E)$ is an interval graph which is not a clique, then there is at least one edge $e \in E$ such that $G + e$ is also an interval graph.*

Theorem 31 (Yamazaki et al. (2019) Thm. 5) *Let $G = (V, E)$ be any interval graph. Then its parent can be computed in $O(n + m)$ time.*

Yamazaki et al. used Lemma 30 and defined the parent of G to be a graph $G + e$ such that $G + e$ is an interval graph and e is lexicographically the smallest possible. They proved that for every interval graph G its parent can be computed in $O(n + m)$ time - see Theorem 31. Thanks to the fact that we work with MPQ -trees and string representations, we are able to get rid of the $O(m)$ factor using the algorithm from Theorem 1.

Theorem 32 *There is a linear time algorithm that for every canonical MPQ -tree \mathcal{T} representing an interval graph G that is not a clique, produces a string representation S' of a graph $G + e$, where e is the lexicographically smallest edge.*

Proof: Let S be the canonical string for \mathcal{T} . Let $12\dots j$ be the longest prefix of S such that $j\dots 21$ is a suffix of S . Clearly, such prefix can be found in $O(n)$ time. Moreover, all vertices $1, \dots, j$ have degree $n - 1$ in the graph G . Hence there is no pair $(x, y) \notin E$ such that $x \in \{1, \dots, j\}$. We prove that there is a pair $(j + 1, y)$ for some $y \geq j + 2$ such that $G + (j + 1, y)$ is an interval graph. Let y be the leftmost

endpoint of an interval that is to the right of the right endpoint of $j + 1$. If such y does not exist, then to the right of the right endpoint of $j + 1$ there are only right endpoints. Hence, G is a clique and we reached a contradiction. Thus, y exists, $(j + 1, k)$ is an edge for all $k \leq y$, and there is no edge between $j + 1$ and y . So, $G + (j + 1, y)$ is the parent of G . In order to produce a string representation for the graph $G + (j + 1, y)$, we move the right endpoint of $j + 1$ to the right until it passes the left endpoint of y . Lemma 7 guarantees that no edge except $(j + 1, y)$ was added. \square

Finally, we take all the pieces together and present our main result.

Theorem 33 *Let \mathcal{T} be a canonical MPQ-tree representing a non-empty ($m \geq 1$) interval graph G . The set of canonical trees for children of G in the family tree \mathcal{F}_n can be computed in $O(nm \log n)$ time.*

Proof: At first, we list all interval edges for the tree \mathcal{T} in $O(\max\{n + m, n \log n\})$ time, see Theorem 29. Then, for each interval edge e , we produce a string representation \mathcal{S}' of $G - e$ in linear time, build MPQ-tree \mathcal{T}' for the interval graph represented by \mathcal{S}' in $O(n)$, and finally compute the canonical form of \mathcal{T}' in $O(n \log n)$ time, see Theorems 23, 1, and 5. Hence, we compute all canonical strings for children candidates in $O(nm \log n)$ time. Theorem 4 implies that in order to check isomorphism, it is enough to remove duplicates from this set. It can be easily done by storing all computed strings in a trie. Finally, we need to filter out those graphs for which G is not a parent. For each candidate canonical string, we compute its parent string in $O(n)$ time, see Theorem 32, build a canonical MPQ-tree in $O(n \log n)$ time and check whether its canonical string equals with a canonical string for the graph G in $O(n)$ time. Thus, we filter out non-children in $O(nm \log n)$ time, and the whole process takes $O(nm \log n)$ time. \square

7 Performance

In the previous section we presented theoretical analysis of our enumeration algorithm. In this section we present two lemmas, that helped us to significantly speedup the execution of the algorithm. Unfortunately, presented tricks do not improve the worst-case time delay, and it is quite easy to show an MPQ-tree that even with those tricks take $O(nm \log n)$ time to process. We leave it as an exercise.

Lemma 34 *Let $v_1 \neq v_2$ be two vertices of the graph $G = (V, E)$. If $N(v_1) \setminus \{v_2\} = N(v_2) \setminus \{v_1\}$, then for every $y \notin \{v_1, v_2\}$ graphs $G - (v_1, y)$ and $G - (v_2, y)$ are isomorphic.*

Proof: One can easily check that the function $f : V \rightarrow V$ such that $f(v_1) = v_2$, $f(v_2) = v_1$ and f identifies other vertices, encodes an isomorphism between $G - (v_1, y)$ and $G - (v_2, y)$. \square

A consequence of the above lemma is the fact that if a P -node contains more than one vertex namely v_1, \dots, v_j , then graphs $G - (v_1, y)$ and $G - (v_a, y)$ for every $a > 1$ are isomorphic. Thus, when listing potential candidates for the children of the graph G , we may omit all edges of the form (v_a, y) for $a > 1$ and list only the edges of the form (v_1, y) . The same argument applies to a Q -node and vertices that belong to the same sections.

Lemma 35 *Let v_1, \dots, v_k be a subset of vertices of the graph $G = (V, E)$ such that $\forall_{i \neq j} : N(v_i) \setminus \{v_j\} = N(v_j) \setminus \{v_i\}$, and v_1, \dots, v_k form a clique in G . All graphs $G_{i,j} = G - (v_i, v_j)$ are pairwise isomorphic.*

Proof: Consider two graphs $G_{i,j}$ and $G_{i',j'}$ and a function $f : V \rightarrow V$ such that $f(v_i) = v_{i'}$, $f(v_j) = v_{j'}$. For the other vertices f is an identity. \square

A consequence of this lemma is the fact that if a P -node contains more than 2 vertices, then we may omit all the edges between them, except (v_1, v_2) . The same applies to the vertices that occupy the same sections of a Q -node. Both presented optimizations do not improve the worst case complexity of our algorithm, but significantly speed it up.

We implemented the proposed algorithms, and generated all non-isomorphic interval graphs on n vertices for all $n \in \{1, \dots, 15\}$. The results are available at <https://patrykmikos.staff.tcs.uj.edu.pl/graphs/>.

References

- H. Acan. Counting unlabeled interval graphs, 2018.
- D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46, 1996. ISSN 0166-218X. doi: [https://doi.org/10.1016/0166-218X\(95\)00026-N](https://doi.org/10.1016/0166-218X(95)00026-N). First International Colloquium on Graphs and Optimization.
- P. Hanlon. Counting interval graphs. *Transactions of The American Mathematical Society - TRANS AMER MATH SOC*, 272, 1982. doi: 10.2307/1998705.
- N. P. J.C. Yang. On the enumeration of interval graphs. *Proceedings of the American Mathematical Society Series B*, 4:1–3, 2017. ISSN 2330-1511. doi: <https://doi.org/10.1090/bproc/27>.
- M. Kiyomi, S. Kijima, and T. Uno. Listing chordal graphs and interval graphs. In *Graph-Theoretic Concepts in Computer Science*, pages 68–77, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- N. Korte and R. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM Journal on Computing*, 18(1):68–81, 1989. doi: 10.1137/0218005.
- G. Lueker and K. Booth. A linear time algorithm for deciding interval graph isomorphism. *J. ACM*, 26(2):183–195, 1979. ISSN 0004-5411. doi: 10.1145/322123.322125.
- R. McConnell and F. de Montgolfier. Algebraic operations on pq trees and modular decomposition trees. In D. Kratsch, editor, *Graph-Theoretic Concepts in Computer Science*, pages 421–432, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- T. Saitoh, K. Yamanaka, M. Kiyomi, and R. Uehara. Random generation and enumeration of proper interval graphs. *IEICE Transactions on Information and Systems*, E93.D(7):1816–1823, 2010. doi: 10.1587/transinf.E93.D.1816.
- T. Saitoh, Y. Otachi, K. Yamanaka, and R. Uehara. Random generation and enumeration of bipartite permutation graphs. *Journal of Discrete Algorithms*, 10:84–97, 2012. ISSN 1570-8667. doi: <https://doi.org/10.1016/j.jda.2011.11.001>.
- R. U. T. Saitoh, M. Kiyomi. Simple efficient algorithm for mpq-tree of an interval graph. *IEICE Technical Report*, 107(127):49–54, 2007. ISSN 0913-5685.

- R. Uehara. Canonical data structure for interval probe graphs. In R. Fleischer and G. Trippen, editors, *Algorithms and Computation*, pages 859–870, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- K. Yamazaki, T. Saitoh, M. Kiyomi, and R. Uehara. Enumeration of nonisomorphic interval graphs and nonisomorphic permutation graphs. In *WALCOM: Algorithms and Computation*, pages 8–19, Cham, 2018. Springer International Publishing.
- K. Yamazaki, T. Saitoh, M. Kiyomi, and R. Uehara. Enumeration of nonisomorphic interval graphs and nonisomorphic permutation graphs. *Theoretical Computer Science*, 2019. ISSN 0304-3975. doi: <https://doi.org/10.1016/j.tcs.2019.04.017>.