# Structural and algorithmic aspects of (multiple) interval graphs

Virginia Ardevol Martinez

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Université Paris-Dauphine

# Structural and algorithmic aspects of (multiple) interval graphs


Soutenue par

**Virginia ARDÉVOL MARTÍNEZ**

Le 5 décembre 2024


École doctorale n°543

**École doctorale SDOSE**

Spécialité

**Informatique**

Composition du jury :

| | |
|---|---|
| Marthe BONAMY<br>Chargée de recherche, CNRS, Université de Bordeaux | *Rapportrice* |
| Michail LAMPIS<br>Maître de Conférences HDR, Université Paris Dauphine-PSL | *Directeur de thèse* |
| Christophe PAUL<br>Directeur de recherche, CNRS, Université de Montpellier | *Rapporteur* |
| Irena RUSU-ROBINI<br>Professeure, Nantes Université | *Examinatrice* |
| Florian SIKORA<br>Maître de Conférences, Université Paris Dauphine-PSL | *Examinateur* |
| Ioan TODINCA<br>Professeur, Université d'Orléans | *Président* |
| Stéphane VIALETTE<br>Directeur de recherche, CNRS, Université Gustave Eiffel | *Examinateur* |

Dauphine | PSL
UNIVERSITÉ PARIS

# Abstract

Intersection graphs represent the patterns of intersection of a family of sets. Depending on the nature of these sets, different subclasses can be defined. One of the most important is the class of interval graphs, which are the intersection graphs of intervals on the real line. Situations arising naturally in scheduling and allocation motivated the study of a generalization of interval graphs known as multiple interval graphs. For any natural number $d$, a (disjoint) $d$-interval is the union of $d$ (disjoint) intervals on the real line, and a graph $G$ is a (disjoint) $d$-interval graph if it is the intersection graph of a family of (disjoint) $d$-intervals. Such a family is then called a $d$-interval representation of $G$. In particular, a $d$-interval graph is unit if it admits a $d$-interval representation where all the intervals have unit length, and balanced if there exists a representation where all the intervals of a same $d$-interval have the same length.

In the first part of this manuscript, we study the class of (disjoint) unit $d$-interval graphs form a structural point of view, with the aim of generalizing Roberts' characterization of unit interval graphs to unit multiple interval graphs. Roberts proved that an interval graph is unit if and only if it does not contain the complete bipartite graph $K_{1,3}$ as an induced subgraph. Here, we generalize this result by proving that for any $d \geqslant 2$, if $G$ is a $K_{1,2d+1}$-free interval graph, then $G$ is a unit $d$-interval graph. However, somehow surprisingly, under the same assumptions $G$ is not always a *disjoint* unit $d$-interval graph, which implies that the class of disjoint unit $d$-interval graphs is strictly included in the class of unit $d$-interval graphs. We also study the relationships between the classes of intersection graphs obtained when we consider disjoint and non-disjoint $d$-intervals in the balanced case, and show that the classes of balanced and disjoint balanced 2-interval graphs coincide, but for $d > 2$, this is no longer true.

We then continue by investigating the complexity of recognizing unit multiple interval graphs. Whereas it is known that recognizing 2-interval graphs and other related classes such as 2-track interval graphs is NP-complete, the complexity of recognizing unit 2-interval graphs is a longstanding open question. We settle it by proving that the recognition of (disjoint) unit 2-interval graphs is also NP-complete. Furthermore, we extend the hardness result for (disjoint) unit $d$-interval graphs for any $d \geqslant 2$ (which does not follow directly in graph recognition problems), and obtain other implications of this result.

In the last part of this manuscript, we focus on an editing problem on interval graphs: PIG-completion. Given an interval graph $G$, PIG-completion asks to find the minimum number of edges that we need to add to $G$ so that it becomes a unit interval graph. We prove that if $G$ contains a vertex that is adjacent to every other vertex in the graph (and in some other specific settings), there exists a dynamic programming algorithm in the modified PQ-tree of $G$ (a well-known data structure used to represent interval

graphs) that solves the problem in polynomial time. Finally, we extend this algorithm for arbitrary interval graphs, where the complexity becomes exponential in the maximum number of non-disjoint centers of claws of $G$.

# Contents

# List of Publications

## Journal publications

[ARSV24] Virginia ARDÉVOL MARTÍNEZ, Romeo RIZZI, Florian SIKORA and Stéphane VIALETTE: Recognizing Unit Multiple Interval Graphs Is Hard. *Discrete Applied Mathematics*, volume 360, pages 258–274, 2025. DOI: 10.1016/j.dam.2024.09.011.

[ASV24] Virginia ARDÉVOL MARTÍNEZ, Florian SIKORA and Stéphane VIALETTE: Parity Permutation Pattern Matching. *Algorithmica*, volume 86, number 8, pages 2605–2624. Springer, 2024. DOI: 10.1007/s00453-024-01237-0.

## In proceedings of conferences

[ARSSV24] Virginia ARDÉVOL MARTÍNEZ, Romeo RIZZI, Abdallah SAFFIDINE, Florian SIKORA and Stéphane VIALETTE: Generalizing Roberts' Characterization of Unit Interval Graphs. *49th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, August 26-30, 2024, Bratislava, Slovakia. *LIPIcs*, volume 306, pages 12:1–12:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. DOI: 10.4230/LIPIcs.MFCS.2024.12.

[ARSV23] Virginia ARDÉVOL MARTÍNEZ, Romeo RIZZI, Florian SIKORA and Stéphane VIALETTE: Recognizing Unit Multiple Intervals Is Hard. *34th International Symposium on Algorithms and Computation (ISAAC)*, December 3-6, 2023, Kyoto, Japan. *LIPIcs*, volume 283, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. DOI: 10.4230/LIPIcs.ISAAC.2023.8.

[ARS23] Virginia ARDÉVOL MARTÍNEZ, Romeo RIZZI and Florian SIKORA: Hardness of Balanced Mobiles. *Combinatorial Algorithms - 34th International Workshop,*

*IWOCA 2023, Tainan, Taiwan*, June 7-10, 2023, Proceedings, volume 13889 of *LNCS*, pages 25–35. Springer, 2023. DOI: 10.1007/978-3-031-34347-6_3.

[ASV23] Virginia ARDÉVOL MARTÍNEZ, Florian SIKORA and Stéphane VIALETTE: Parity Permutation Pattern Matching. *Algorithms and Computation - 17th International Conference and Workshops, WALCOM* 2023, Hsinchu, Taiwan, March 22-24, 2023, Proceedings, volume 13973 of *LNCS*, pages 384–395. Springer, 2023. DOI: 10.1007/978-3-031-27051-2_32.

# 1. Introduction

Consider the schedule of courses depicted in Figure 1.1, where every course is represented by a time interval indicating its duration, and suppose that we want to solve two different problems. The first one is computing the maximum number of courses that a student can register for, assuming that attendance is mandatory and they must be present for the entire duration of each lecture. The second one is computing the minimum amount of professors that we need to hire so that every course can take place, assuming that every professor is competent in every subject and can teach any set of courses as long as they do not overlap on time.

To solve these problems, one can give a modelization in graph-theoretical terms by representing every course by a vertex and adding an edge between two vertices if and only if the corresponding courses clash, i.e., they overlap in time (see the graph on the right of Figure 1.1). Then, the solution to our problems is given by two very well-known graph problems: Maximum Independent Set and Minimum Coloring, respectively. Indeed, Maximum Independent Set asks for a set of pairwise non-adjacent vertices of maximum cardinality, so given an optimal solution, the courses associated to the chosen vertices comprise a largest set of courses that a student can register for at the same time. On the other hand, Minimum Coloring asks to color the vertices of a graph with the minimum amount of colors such that no two adjacent vertices have the same color. Thus, an optimal solution can be translated to a solution of our original problem by assigning each color to a different professor, and considering that a professor teaches the courses associated to vertices of their color.

In general, both Maximum Independent Set and Minimum Coloring are hard to solve, but in this particular example, the input graph has a very specific structure that allows us to easily compute an optimal solution. Indeed, if the input graph is an *interval graph*, given an interval representation (for example, the schedule in Figure 1.1), we can

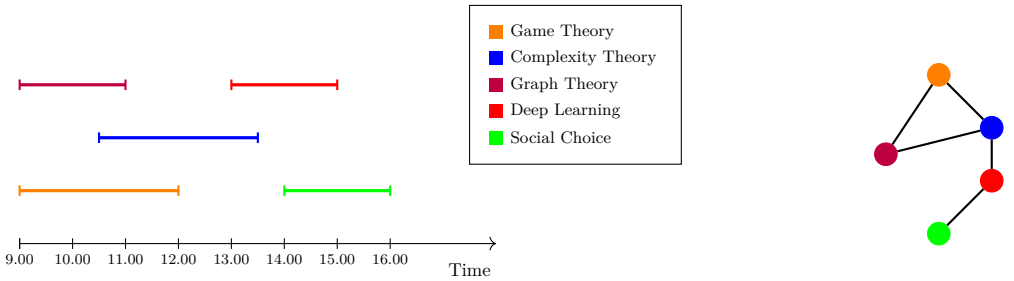



Figure 1.1.: On the left, a schedule of courses. On the right, the graph representing the incompatibilities between the courses. Such a graph is an interval graph.

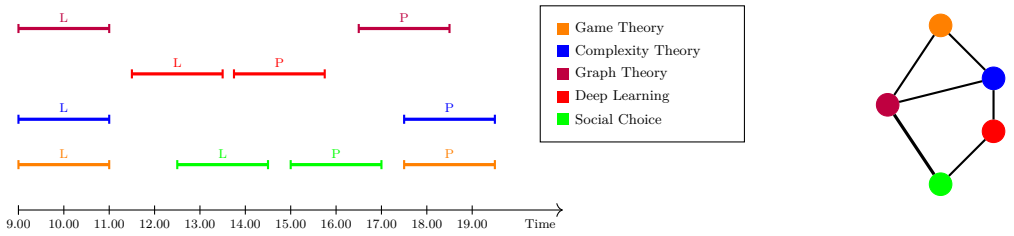



Figure 1.2.: On the left, a schedule of courses where every course is divided into a lecture (L) and a problem session (P). On the right, the graph representing the incompatibilities between courses. Such a graph is a 2-interval graph.

compute an optimal solution of Maximum Independent Set by greedily choosing the interval that finishes first among the intervals that do not intersect any of the already chosen ones, starting from the leftmost interval of the representation. Similarly, we can compute an optimal solution of Minimum Coloring by considering the intervals in the order given by their right endpoints and greedily assigning the first available color to each interval (where a color becomes available if the considered interval does not intersect an interval of this color).

One can also ask for additional constraints in the schedule of the courses. For example, a very natural constraint would be to enforce that all the lectures have the same duration, which means that the time intervals representing the courses would all have the same length. The graph representing the incompatibilities of such a schedule is called a *unit interval graph*.

Let us now consider a different scenario where every course is divided into two classes: a lecture and a problem session, each scheduled at a different time (see Figure 1.2). In this new setting, to register for a course, a student is required to attend both classes for their entire duration. Likewise, to teach a course a professor must be available during the lecture and the problem session.

Now, in order to model this as a graph problem, we need to add an edge between two courses if either part of one course clashes with either part of the other (see the graph on the right of Figure 1.2). This graph is called a 2-interval graph, because it is the intersection graph of 2-intervals (the union of the two intervals representing a course). As before, we can also enforce that all the classes have the same duration, which would yield a unit 2-interval graph. However, when our graph is not interval but 2-interval, computing the maximum number of courses that we can take or the minimum number of professors that we need is not as easy!

Of course, this example can be extended: if every course is divided in three classes, a lecture, a problem session, and a computer session, then the graph representing the incompatibilities would be a 3-interval graph; and in general, for any natural number $d$, if the course is divided in $d$ separate classes, then the incompatibilities graph is a $d$-interval graph.

The class of $d$-interval graphs, and more specifically, the class of unit $d$-interval graphs, is the main focus of this thesis. Given any graph class, the first fundamental question

that arises is how to determine whether an arbitrary graph belongs to the class. This is a question of theoretical importance, but has also been motivated by practical applications In particular, Maas gave the following application of the recognition of $d$-interval graphs in scheduling [115]:

> "In a laboratory there is a machine that can be used for a variety of experiments. For every type of experiment, the configuration of the machine has to be changed by a specialized staff. For every day, we know which groups of scientists want to carry out their experiments and which configurations they need. We look for a time-table for the laboratory that equally minimizes the number of times each configuration has to be implemented, and the number of times each team is interrupted by another one. Such a time-table is a $d$-interval model for the bipartite graph $G$ whose vertices are the teams of researchers and the configurations of the machine, and whose edges connect teams with the configurations they need."

In this thesis, we study (multiple) interval graphs from a structural and algorithmic point of view, with many of the results being closely related to the recognition of unit multiple interval graphs. The main content of Chapter 5 appeared in [7], and is the result of joint work with Romeo Rizzi, which began while he was visiting Paris as an invited professor, and Abdallah Saffidine. The main content of Chapter 6 appeared in [9] and is also joint work with Romeo Rizzi. Finally, Chapter 7 is the result of joint work with Andrea Craco and Romeo Rizzi, which started while I was visiting them in Italy, and will appear in a future publication.

During this thesis I have also worked on other problems which are not related to interval graphs. These results, which correspond to publications [10] and [8], are presented in the appendix. Finally, I have also carried out research in a topic related to intersection graphs, though not to interval graphs: readability, a graph parameter introduced in [38], motivated by applications of overlap graphs in bioinformatics. The readability of a digraph $D$ is the smallest integer $r$ such that there exists an injective overlap labeling of $D$ (a labeling $l$ of the vertices such that there exists an edge $(u, v)$ if and only if $l(u)$ and $l(v)$ overlap) with strings of length $r$. However these results are not dicussed here as they are still preliminary.

# 2. Preliminaries

In this chapter, we introduce some basic notions of graph theory and give a brief overview of complexity theory. For more background on these topics, we refer to [52] and [70] respectively.

## 2.1. Basic notions of graph theory

**Definitions.** A *graph* $G = (V, E)$ consists of a set of *vertices* $V$ and a set of *edges* $E \subseteq V \times V$, formed by pairs of vertices. We will write $V(G)$ and $E(G)$ to denote the vertex and edge set of graph $G$ when the graph is not clear from the context. We denote an edge between vertices $u$ and $v$ by $(u, v)$. Throughout this manuscript, all the graphs considered are *simple* (without loops or multiple edges) and *undirected* (the set of edges is formed by unordered pairs of vertices), unless explicitly stated otherwise.

Two vertices $u$ and $v$ are *adjacent* if there exists an edge joining them, that is, an edge of the form $(u, v)$. Given an edge $(u, v)$, the vertices $u$ and $v$ are called its *endpoints*, and we say that the edge is *incident* to both $u$ and $v$.

Given a vertex $v \in V$, the *neighborhood* of $v$ is the set of vertices adjacent to it, formally defined as $N(v) := \{u \in V \mid (u, v) \in E\}$. The *closed neighborhood* of $v$ is the set $N[v] := \{v\} \cup N(v)$.

A vertex $v$ is *universal* if it is adjacent to every other vertex of the graph, that is, $N[v] = V$. A vertex $v$ is a *private neighbor* of vertex $u$ if $N(v) = \{u\}$.

The *degree* of a vertex $v$, denoted $\deg(v)$, is the cardinality of $N(v)$, i.e., $\deg(v) := |N(v)|$. The *maximum degree* $\Delta$ of a graph is defined as the highest degree over all the vertices of the graph, i.e., $\Delta := \max_{v \in V}\{\deg(v)\}$.

A graph is *regular* if every vertex has the same degree.

A *subgraph* of $G$ is a graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$ such that $(u, v) \in E'$ implies that $u$ and $v$ are both contained in $V'$.

An *induced subgraph* of $G$ is a graph $G' = (V', E')$ with $V' \subseteq V$ and $E' = \{(u, v) \mid (u, v) \in E, u, v \in V'\}$. We say that $G'$ is the subgraph induced by $V'$, and we denote it by $G[V']$.

The *complement* of a graph $G = (V, E)$ is the graph with vertex set $V$ and edge set $\overline{E} := \{(u, v) \mid u, v \in V, (u, v) \notin E\}$, and we denote it by $\overline{G}$.

A *path* of length $n$ is a set of vertices $P_{n+1} = (v_1, \ldots, v_{n+1})$ such that $(v_i, v_{i+1}) \in E$ for every $i \in \{1, \ldots, n\}$. A *cycle* of length $n$ is a path $P_{n+1}$ with $v_1 = v_{n+1}$.

A graph is *connected* if there exists a path between every pair of vertices.

A *clique* or *complete graph* of size $n$, denoted $K_n$, is a set of $n$ pairwise adjacent vertices.

A *maximal clique* of a graph is a complete subgraph that is not properly contained in any other complete subgraph.

**Some fundamental graph classes.** A *class* of graphs $\mathcal{C}$ is the set of graphs satisfying a certain property $\Pi$, that is, $\mathcal{C} := \{G \mid G \text{ satisfies } \Pi\}$. Many graph theoretical problems involve characterizing the members of a class, whether by enumerating the members of a class, characterizing the class in terms of forbidden structures, studying the relationships among different classes, or finding efficient algorithms to decide membership in a class.

A class of graphs $\mathcal{C}_1$ is a *subclass* of the class $\mathcal{C}_2$, denoted $\mathcal{C}_1 \subseteq \mathcal{C}_2$, if every graph that belongs to $\mathcal{C}_1$ also belongs to $\mathcal{C}_2$. If $\mathcal{C}_1 \subseteq \mathcal{C}_2$ and there exists a graph that belongs to $\mathcal{C}_2$ but not to $\mathcal{C}_1$, we say that $\mathcal{C}_1$ is a *proper subclass* of $\mathcal{C}_2$, and we denote it by $\mathcal{C}_1 \subsetneq \mathcal{C}_2$. On the other hand, if $\mathcal{C}_1 \subseteq \mathcal{C}_2$, we also say that $\mathcal{C}_2$ is a *superclass* of $\mathcal{C}_1$ (resp., *proper superclass* if $\mathcal{C}_1 \subsetneq \mathcal{C}_2$).

A class of graphs $\mathcal{C}$ is *hereditary* if for every graph $G \in \mathcal{C}$ and every induced subgraph $H$ of $G$, we have that $H \in \mathcal{C}$. A *minimal forbidden induced subgraph* for $\mathcal{C}$ is a graph $F$ such that for every induced subgraph $H$ of $F$, $H \in \mathcal{F}$. Every hereditary graph class $\mathcal{C}$ is characterized by its set of minimal forbidden subgraphs.

We now describe some well-known classes of graphs that will appear throughout this manuscript. The more specific classes that constitute the focus of this thesis will only be introduced in Chapter 3.

A graph is *bipartite* if its vertex set can be partitioned into two sets, $V_1, V_2$ such that no edge connects vertices within the same set.

A graph is *co-bipartite* if its complement is bipartite.

A graph $G = (V_1 \cup V_2, E)$ is a *complete bipartite* graph if $E = \{(u, v) \mid u \in V_1, v \in V_2\}$. When $|V_1| = m$ and $|V_2| = t$, we denote the complete bipartite graph by $K_{m,t}$. In particular, if $m = 1$, we call the graph $K_{1,t}$ a *$t$-claw*. The unique vertex contained in $V_1$ is then called the *center* of the claw, and the other $t$ vertices are called *leaves*. For any $t \geqslant 3$, if the set of vertices $\{v_0, v_1, \ldots, v_t\}$ induces a $K_{1,t}$ with center $v_0$, we will sometimes denote it by $[v_0; v_1, \ldots, v_t]$. We say that a graph is $K_{1,t}$-free if it does not contain any induced $K_{1,t}$'s, and we say that an induced $t$-claw is *maximal* if it is not contained in an induced $K_{1,t'}$ with $t' > t$.

A *tree* is an undirected connected graph without cycles. A tree is *rooted* if one of its vertices has been designated as the root. Given a rooted tree, the *parent* of a vertex $v$ is the vertex connected to $v$ in the path to the root. If $u$ is the parent of $v$, we say that $v$ is a *child* of $u$, and every other child of $u$ is called a *sibling* of $v$. A vertex is a *leaf* if it has no children, and an *internal vertex* otherwise. The *ancestors* of a vertex $v$ are the vertices in the path from the root to $v$. The root is an ancestor of every vertex. If $u$ is an ancestor of $v$, we say that $v$ is a *descendant* of $u$. Given a vertex $v$, the *subtree rooted at $v$* is the subgraph of the tree induced by $v$ and all its descendants.

A tree $T$ is a *caterpillar* if removing all its leaves results in a path.

## 2.2. Algorithms and complexity

Algorithms are sets of rules to be followed in order to solve a specific problem. They take an input, called an *instance* of the problem, and return an output: the *solution*. These problems can be classified into two categories: *decision problems* and *optimization*

*problems*. In the first case, we aim to answer a yes or no question, while in the second one, we seek to find a solution that minimizes or maximizes a certain objective. For example, consider the well-known graph coloring problem, where the goal is to color all the vertices of a graph in such a way that no two vertices are given the same color if there is an edge connecting them. Determining whether a graph can be properly colored with $k$ different colors is a decision problem, while finding the minimum number of colors required to obtain a proper coloring is an optimization problem. Note that every optimization problem has an associated decision problem, defined by adding a parameter $k$ to the input, which consists on determining whether there exists a solution of value at most $k$ (or at least $k$ for maximization problems).

A particular type of decision problems are graph *recognition* problems, where given a graph $G$ as input, we ask whether $G$ belongs to a specific class of graphs:

Graph Recognition for class $\mathcal{C}$

***Instance:*** A graph $G = (V, E)$.
***Goal:*** Decide whether $G$ belongs to $\mathcal{C}$.

In order to be able to solve these problems for any given input, we need to design algorithms. However, this is not always an easy task, as there are multiple factors that need to be considered. Indeed, when designing an algorithm, we are not only interested in the fact that it returns an acceptable solution: it is also crucial to study its efficiency. There are different aspects that play an important role in this matter, such as the memory usage or the execution time. In this thesis, we focus only in measuring the complexity in terms of the *running time* of the algorithm, which is the number of basic operations it performs for the worst case input scenario, measured as a function of the input length. However, the exact number of operations performed is inconsequential: our interest lies in the asymptotic behavior of this function. This allows us to understand how the running time of the algorithm scales with the size of the input, which is the key factor when comparing the efficiency of different algorithms. Thus, we generally use the "big-O" notation to express the running time of an algorithm. That is, let $n$ be the input size and $f(n)$ the function that describes the number of operations executed by the algorithm. Using the big-O notation, we can express $f(n)$ as $O(g(n))$ for some function $g(n)$ if there exists a constant $c$ such that $f(n) \leqslant c \cdot g(n)$ for every sufficiently large $n$. Intuitively, this means that $f(n)$ is smaller or equal to $g(n)$ if we ignore constant factors.

We say that an algorithm is *polynomial* if it has a time complexity of $O(n^c)$ for some constant $c$. In particular, we say that the algorithm is *linear* if $c = 1$, and *quadratic* if $c = 2$. If $c = 0$, that is, if the algorithm has a complexity of $O(1)$, we say that it runs in *constant* time. On the other hand, if the complexity of the algorithm is of the form $O(c^n)$ for some constant $c$, we say that the algorithm is *exponential*.

**P vs NP.** Computational complexity allows us to categorize problems into different classes according to the running time of an algorithm that solves them optimally.

A problem is in the class P if there exists a polynomial-time algorithm that solves it. To prove that the problem belongs to the class P, it suffices to provide such an algorithm. On

the other hand, we say that a problem is in the class NP (non-deterministic polynomial time) if given a solution of an instance, one can verify in polynomial time whether it is a correct solution. Equivalently, NP is the class of all problems that can be solved by a non-deterministic algorithm (that is, an algorithm that can return different solutions given the same input) in polynomial time.

Clearly, P$\subseteq$ NP. However, whether NP$\subseteq$ P is one of the most important open questions in computer science, although it is widely believed that P$\neq$ NP. This believe is motivated by the fact that there exist many problems for which nobody has been able to provide a polynomial-time algorithm, despite decades of effort. For such problems, we can attempt to prove that they are NP-hard, that is, they are as hard as the hardest problem in NP, and so we do not expect a polynomial-time algorithm to exist. To prove that a problem is NP-hard, one generally seeks to give a polynomial-time reduction from a known NP-hard problem. Given two problems $P_1$ and $P_2$, we say that $P_1$ is *polynomial-time reducible* to $P_2$ if for every instance $I_1$ of $P_1$, we can construct in polynomial time an instance $f(I_1)$ of $P_2$ such that $I_1$ is a yes-instance if and only if $f(I_1)$ is a yes-instance.

A decision problem is said to be NP-complete if and only if it is in NP and it is NP-hard. The first problem that was proven to be NP-complete is Satisfiability [41], i.e., the problem of determining whether there exists a variable assignment that satisfies a given Boolean formula. Shortly after, Karp extended the knowledge of this class by providing a list of problems that were also NP-complete [98]. Since then, a large number of problems has been proven to belong to this class, supporting the belief that P $\neq$ NP. Some of the most important ones have been gathered by Garey and Johnson in *Computers and Intractability* [70], which stands as a cornerstone reference in the field.

As a final remark, note that if an optimization problem on graphs is polynomial-time solvable, then every instance of the problem on any graph can be solved efficiently. However, when a problem is NP-hard, it doesn't necessarily imply that it is intractable for all graphs: there might exist certain graphs where the problem can still be solved efficiently. In particular, if we restrict the problem to a specific class of graphs, then it might become polynomial-time solvable within that class. This polynomial-time solvability then extends to all of its subclasses. Conversely, if the problem is NP-hard when restricted to a specific class, then it will remain NP-hard in every superclass.

Recognition problems are quite different: being able to recognize a particular class of graphs in polynomial time does not guarantee that every subclass will also be recognizable in polynomial time. Likewise, if the recognition of a certain class is an NP-complete problem, we cannot infer anything about the complexity of recognizing a superclass. As a trivial example, note that the class of all graphs can be recognized in polynomial time, while the class of graphs that can be properly colored with 3 colors does not admit an efficient recognition algorithm.

**ETH.** Suppose now that we have proven that a problem is NP-complete and we have an exponential-time algorithm to solve it. For practical applications, whether we have an $O(2^n)$ algorithm or an $O(2^{\sqrt{n}})$ algorithm has a big significance. Therefore, one might be interested in knowing whether an exponential time algorithm can be improved or

not. The *Exponential Time Hypothesis* (ETH) was proposed to tackle this problem [91]. Intuitively, much as the theory of NP-hardness is based in the assumption that SATISFIABILITY cannot be solved in polynomial time, the ETH is based on the assumption that 3-SAT (SATISFIABILITY restricted to formulas where every clause has three literals) cannot be solved in time significantly better than $2^n$, where $n$ is the number of variables in the input formula.

**Definition 2.1** (Exponential Time Hypothesis)**.** *Let $\delta$ be the infimum of the set of constants $c$ for which there exists an algorithm solving 3-SAT in time $O(2^{cn})$. Then $\delta > 0$.*

Equivalently, the ETH states that 3-SAT cannot be solved in subexponential time, i.e., in time $2^{o(n)}$, where $o(n)$ stands for "small-o" of $n$ and means that there cannot exist a function $f$ such that for every $\epsilon > 0$, $f(n) < \epsilon n$ for $n$ sufficiently large.

Suppose now that the ETH holds and we want to use it to infer lower bounds on other problems. Since most reductions from 3-SAT output instances that depend on the size of the formula (not just the size $n$ of the variable set) and an arbitrary instance of 3-SAT can have up to $O(n^3)$ clauses, the ETH can only exclude an algorithm for the target problem with a running time of $O(2^{|x|^{1/3}})$, where $x$ is the input of the target problem. To exclude an algorithm with running time $O(2^{|x|})$, the number of clauses would actually need to be linear in terms of $n$. The sparsification lemma allows us to make this assumption [92], which yields the following result:

**Lemma 2.1.** *Unless the ETH fails, 3-SAT cannot be solved in time $2^{o(n+m)}$.*

Now, if we assume that the ETH is true and we have reduced an instance $\Phi$ of 3-SAT to an instance of a problem whose size is linear in the size of $\Phi$, then the target problem cannot be solved in time $2^{o(|x|)}$. For a more detailed introduction to the ETH, see Chapter 14 of *Parameterized Algorithms* [49].

**Circumventing NP-hardness.** Different approaches have been proposed to deal with NP-hard problems. One of them is *parameterized complexity*, where the aim is to confine the exponential explosion of the runtime of an algorithm to some well-chosen parameter.

A parameterized problem consists of an instance $X$ and a parameter $k$. The standard parameterization takes $k$ to be the size of the desired solution (for example, the size of a maximum independent set), but the parameter can also be chosen to be some structural property of the instance (for example, the maximum degree of a graph).

A parameterized problem is in the class XP if it can be solved in time $n^{f(k)}$ for some computable function $f$ that depends on the parameter. In particular, a problem is fixed-parameter tractable (FPT) if there exists an algorithm with running time $f(k) \cdot n^c$ for some computable function $f$ and some constant $c$ (that is, the running time is exponential on the parameter but polynomial on the size of the input). Nevertheless, not every problem in XP admits an FPT algorithm: problems that are in XP but not in FPT belong to the W-hierarchy. It is widely believed that W[1] $\neq$ FPT, and so being W[1]-hard is strong evidence against being in FPT.

To prove that a problem is FPT, it suffices to provide an algorithm with running time $f(k) \cdot n^c$. One of the most useful techniques to design FPT algorithms is *kernelization*. Given an instance $(X, k)$ of a parameterized problem, a kernelization algorithm computes an equivalent instance $(X', k')$ whose size is bounded by a function of the original parameter. That is, given an instance $(X, k)$, we construct an instance $(X', k')$ such that:

- $(X', k')$ is a yes-instance if and only if $(X, k)$ is a yes-instance.
- $|X'| \leqslant f(k)$ for some computable function $f$ depending only on $k$.
- $k' \leqslant g(k)$ for some function $g$.

The constructed instance is then called a *kernel*. To design efficient algorithms, one generally seeks a kernel of small size, that is, we want the function $f(k)$ to be polynomial in $k$. If this holds, we say that $(X', k')$ is a *polynomial kernel*. It is well-known that a parameterized problem is in FPT if and only if it admits a kernelization algorithm [123]. However, not every problem admits a polynomial kernel under reasonable assumptions [20].

On the other hand, when a problem does not admit an FPT algorithm, we can attempt to prove that it is W[1]-hard. Just as NP-hardness is established through polynomial time reductions, to prove that a parameterized problem $P_2$ is W[1]-hard, one needs to provide an FPT-reduction from a known W[1]-hard problem $P_1$. That is, for every instance $(X_1, k_1)$ of $P_1$, one needs to construct an instance $(X_2, k_2)$ of $P_2$ such that:

- $(X_1, k_1)$ is a yes-instance for $P_1$ if and only if $(X_2, k_2)$ is a yes-instance for $P_2$.
- $k_2 = g(k_1)$ for some function $g$ that depends only on $k_1$.
- The construction of $(X_2, k_2)$ can be carried out in time $f(k_1) \cdot |X_1|^c$ for some function $f$ depending only on $k_1$ and some constant $c$.

Finally, problems which are already NP-hard for a constant value of $k$ are not in XP. We refer to [49] for more background on parameterized complexity.

# 3. State of the Art

All the results presented in the main body of this manuscript are related to structural and algorithmic questions on interval and multiple interval graphs, two subclasses of intersection graphs. In this chapter, we give an condensed overview of the landscape of known results related to these topics. We begin Section 3.1 with a general introduction to intersection graphs and one of its most important subclasses, the class of chordal graphs. Then, in Section 3.2, we introduce interval graphs, present the most relevant characterizations and structural properties of this class, and discuss some results related to their recognition. Finally, in Section 3.3, we define multiple interval graphs and other generalizations of interval graphs, and provide an overview of the known structural properties and recognition results.

## 3.1. Intersection graphs

Intersection graphs have been widely studied in the literature, as they are important both from a theoretical and a practical point of view. For an extended introduction to the topic, see [118].

**Definition 3.1.** *Given a family of sets $\mathcal{F} = \{S_1, \ldots, S_k\}$, we define the* intersection graph *of $\mathcal{F}$, denoted $\Omega(\mathcal{F})$, as the graph having $\mathcal{F}$ as vertex set and vertex $S_i$ adjacent to vertex $S_j$ if and only if $i \neq j$ and the intersection of the sets is non-empty, i.e., $S_i \cap S_j \neq \emptyset$. A graph is an intersection graph if there exists a family $\mathcal{F}$ such that $G = \Omega(\mathcal{F})$. The family $\mathcal{F}$ is then called a* set representation *or* model *of $G$.*

It is well known that every graph is an intersection graph [143].

The family of sets used to define intersection graphs can contain objects of very different types: intervals, arcs on a circle, trapezoids, unit disks, curves of a plane, axis-parallel boxes in dimension $k$, edges of a graph, intervals with varying tolerances... This diversity gives raise to many important subclasses of intersection graphs, such as interval graphs, circular arc graphs, trapezoid graphs, unit disk graphs, string graphs, graphs with boxicity $k$, line graphs, or tolerance graphs. Even some classes of graphs whose original definition is not given in terms of intersection graphs can also be defined in this way. That is the case of chordal graphs, which are intersection graphs of subtrees of a tree, or permutation graphs, which can be viewed as the intersection graphs of line segments whose endpoints lie on two parallel lines.

Before moving on to the study of intersection graphs of intervals, let us briefly present the class of chordal graphs (also called triangulated graphs or rigid circuit graphs), which properly contains the class of interval graphs, and possesses some interesting properties that are inherited by interval graphs.

**Definition 3.2.** *A graph $G$ is* chordal *if every cycle of length greater than three has a chord, that is, an edge connecting two non-consecutive vertices on the cycle.*

The previous definition implies that a graph is chordal if and only if it does not contain any induced cycles of length strictly greater than three. Alternatively, chordal graphs can be characterized as graphs that admit a *perfect* or *simplicial elimination ordering* [66]. A vertex $v$ of graph $G$ is *simplicial* if its neighborhood induces a complete subgraph of $G$. An ordering $\{v_1, \ldots, v_n\}$ of the vertices of $G$ is a simplicial elimination ordering if for $1 \leqslant i \leqslant n$, the vertex $v_i$ is simplicial in the graph induced by $\{v_i, \ldots, v_n\}$. Chordal graphs are also perfect.

Rose, Tarjan and Lueker proved that chordal graphs can be recognized in linear time by showing that a graph is chordal if and only if the ordering of the vertices produced by a lexicographic breadth first search algorithm (see Algorithm 1) is a perfect elimination ordering [135]. Furthermore, the structural properties of this class of graphs allows many optimization problems to be solved efficiently in chordal graphs, such us minimum coloring, maximum independent set or maximum clique [71].

**Algorithm 1** Lexicographic Breadth-First Search (LexBFS)

```
1: Assign an empty label to all vertices.
2: for i = n down to 1 do
3:     Select an unnumbered vertex v with lexicographically the largest label.
4:     Assign the number i to vertex v: α(i) := v.
5:     for each unnumbered vertex w adjacent to v do
6:         Add the current value of i to the label of w.
7:     end for
8: end for
9: return α  (the numbering of vertices obtained)
```

As we mentioned before, chordal graphs can also be characterized as the intersection graphs of a family of subtrees of a tree [30, 72, 147]. In fact, a graph $G$ is chordal if and only if it has a *clique tree*, that is, a tree on the set $M$ of maximal cliques of $G$ such that for every pair of distinct cliques $M_i, M_j \in M$, the set $M_i \cap M_j$ is contained in every clique on the path connecting them in the tree. Clique trees also satisfy the property that for every vertex $v$ of the graph, the set of cliques containing it induces a connected subtree of the tree.

**Subclasses of chordal graphs.** Two of the most important subclasses of chordal graphs are *split* graphs and interval graphs. A graph $G = (C \cup I, E)$ is a split graph if its vertex set can be partitioned into a set $C$ of pairwise adjacent vertices and set $I$ of pairwise nonadjacent vertices, while a graph $G$ is interval if it is the intersection graph of a family of intervals. The classes of split and interval graphs are incomparable: a path of length 5 is interval but not split, and a net (see Figure 3.6c) is split but not interval.

Within interval graphs, there exists an important subclass known as *trivially perfect graphs*. A graph $G = (V, E)$ is a trivially perfect graph or *quasi-threshold* graph if each

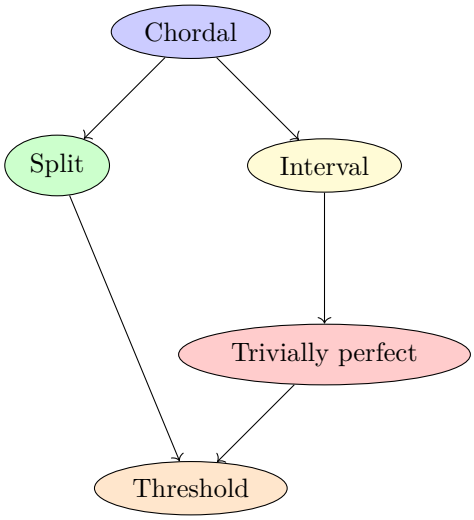



Figure 3.1.: Containment relationships between some subclasses of chordal graphs. An arrow from class $\mathcal{C}_1$ to a class $\mathcal{C}_2$ indicates that $\mathcal{C}_2 \subsetneq \mathcal{C}_1$.

of its connected components admits a rooted tree $T = (V, E(T))$ on the same vertex set, rooted at a vertex $r \in V$, such that $(u, v) \in E$ if and only if there is a path in $T$ starting in $r$ and containing both $u$ and $v$. The class of trivially perfect graphs is also incomparable with split graphs. However, both split and trivially perfect graphs properly contain a common subclass: the class of threshold graphs.

A graph $G$ is a *threshold* graph if it can be constructed from the empty graph by repeatedly adding either an isolated vertex (nonadjacent to every other vertex) or a dominating vertex (adjacent to every other vertex). Equivalently, it is a split graph in which any two independent vertices satisfy that the neighborhood of one is contained in the neighborhood of the other.

A summary of the containment relations between these subclasses of chordal graphs can be seen in Figure 3.1.

## 3.2. Interval graphs

Let us start by formally defining intervals and interval graphs.

**Definition 3.3.** *A* closed interval *is a set of real numbers of the form* $[a, b] := \{x \in \mathbb{R} \mid a \leqslant x \leqslant b\}$. *All the intervals considered throughout this manuscript are closed, unless indicated otherwise, so in the following, we will refer to them simply as* intervals. *Given an interval* $I = [a, b]$, *we denote its left endpoint* $a$ *by* $l(I)$, *and its right endpoint* $b$ *by* $r(I)$.

Note that many references do not actually specify whether the intervals considered for the intersection representation of interval graphs are open or closed, probably because both definitions lead to the same class of finite graphs [65], even for unit interval graphs.

Here, we define the intervals to be closed, but note that this equivalence implies that all of the results still hold if we let the intervals be open. However, if we allow the use of both open and closed intervals within one representation, then the class of unit interval graphs obtained is not the same as if we only allowed open or closed intervals within one representation [130]. Thus, the results of this thesis only hold if all the intervals within a representation are closed or all the intervals are open.

**Definition 3.4.** *A graph $G = (V, E)$ is an* interval graph *if there exists a bijection from the vertices of $G$ to a set of intervals, $f : V \to \mathcal{I}$, such that there exists an edge between two vertices $u$ and $v$ if and only if their corresponding intervals intersect, i.e., if $f(u) \cap f(v) \neq \emptyset$. The set $\mathcal{I}$ is called an* interval representation *of $G$ (or* interval model *in some references).*

The notion of interval graphs originated in the 50s. In 1957, Hajös defined the intersection graph of intervals on a straight line, and proposed studying the recognition of such graphs [87]. Independently, in 1959, Benzer suggested modeling the topology of the subelements of genes mathematically [15]. The modeling he proposed actually corresponds to the notion of interval graphs. Since then, this class of graphs has been extensively studied, mainly due to its numerous applications in scheduling, resource allocation and bioinformatics [12, 151, 39], but also because of its algorithmic advantages: many NP-hard problems become solvable in linear time when restricted to this class of graphs, for example, dominating set [23] or Hamiltonian cycle [100] (which are NP-complete even on chordal graphs). One of the few classical problems that remains NP-complete in this class of graphs is maximum cut [2].

### 3.2.1. Characterizations

From a structural point of view, there exist several characterizations of interval graphs. The first one dates back to 1962 and was given by Lekkerkerker and Boland [109]. It highlights a new forbidden structure which differentiates them from chordal graphs: asteroidal triples.

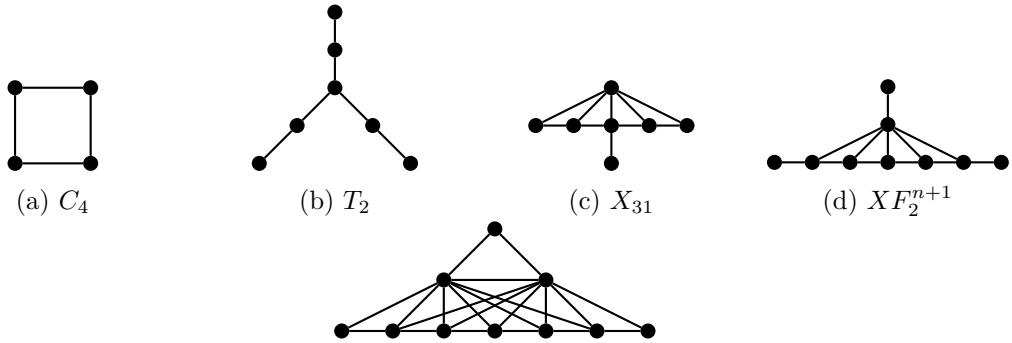
**Definition 3.5.** *A set of three vertices of a graph forms an* asteroidal triple *if for each pair of vertices, there exists a path containing them that does not pass through a neighbor of the third vertex.*

In the words of Golumbic, asteroidal triples represent a well-known law of business: "every shipment from a supplier to the consumer must pass by the middle man" [77].

**Theorem 3.1** ([109])**.** *An undirected graph $G$ is an interval graph if and only if the following two conditions are satisfied:*

1. *$G$ is a chordal graph.*
2. *$G$ does not contain any asteroidal triples.*

Lekkerkerker and Boland also characterized all the minimal graphs that contain an asteroidal triple, which allows us to reformulate the previous theorem to obtain a characterization of interval graphs in terms of forbidden induced subgraphs.



(a) $C_4$ (b) $T_2$ (c) $X_{31}$ (d) $XF_2^{n+1}$

(e) $XF_3^n$

Figure 3.2.: Forbidden induced subgraphs for interval graphs.

**Theorem 3.2** ([109])**.** *A graph $G$ is an interval graph if and only if it does not contain any of the graphs $C_{n+4}, T_2, X_{31}, XF_2^{n+1}$ or $XF_3^n$ as an induced subgraph (see Figure 3.2 for an illustration of the forbidden induced subgraphs).*

Later on, in 1964, Gilmore and Hoffman [75] gave a different characterization of interval graphs, which stems from their relationship to comparability graphs. Comparability graphs can be used to represent strict partial orders.

**Definition 3.6.** *A* strict partial order *is a binary relation $P$ on a set $X$ that satisfies the following conditions:*

1. *Irreflexivity: not $aPa$ for every $a \in X$*
2. *Asymmetry: for every pair of elements $a, b \in X$, if $aPb$ then not $bPa$.*
3. *Transitivity: for every triple of elements $a, b, c \in X$, if $aPb$ and $bPc$, then $aPc$.*

Two elements $a, b \in X$ are *comparable* in the ordered set $(X, P)$ if $xPy$ or $yPx$, and *incomparable* otherwise. Given an ordered set $(V, P)$, the *comparability graph* $G = (V, E)$ associated to it has edge set $E = \{(a, b) \mid aPb \text{ or } bPa\}$.

**Definition 3.7.** *A graph $G$ is a comparability graph if and only if there exists a strict partial order relation $P$ on the vertices of $G$ such that $G$ is the comparability graph of $(V, P)$.*

Equivalently, a graph is a comparability graph if and only if it has a transitive orientation, that is, we can replace every edge $(u, v)$ by a directed edge, i.e., $(u, v)$ or $(v, u)$, in such a way that if there exist two directed edges $(v_1, v_2)$ and $(v_2, v_3)$, then there also exists the directed edge $(v_1, v_3)$. We can also define the incomparability graph associated to $(V, P)$ as $\overline{G} = (V, \overline{E})$.

The characterization of interval graphs in terms of comparability graphs introduced by Gilmore and Hoffman is the following.

**Theorem 3.3** ([75])**.** *Let $G$ be an undirected graph. Then, the following are equivalent:*

1. *G is an interval graph.*
2. *G is chordal and its complement is a comparability graph.*
3. *The maximal cliques of G can be linearly ordered such that, for every vertex v of G, the maximal cliques of v occur consecutively.*

The third statement can be translated into a matrix formulation, as it implies some properties on the *clique matrix* of a graph, which is a matrix with a row for each maximal clique of the graph and a column for each vertex, and with an entry being 1 if and only if the vertex belongs to the clique and 0 otherwise. This new formulation was given by Fulkerson and Gross in 1965 [66].

**Theorem 3.4** ([66])**.** *A graph G is an interval graph if and only if its clique matrix M has the consecutive 1's property for columns, that is, its rows can be permuted in such a way that the 1's in each column occur consecutively.*

Equivalently, a graph is an interval graph if and only if it has a clique tree that is a path. In particular, much as chordal graphs can be viewed as intersection graphs of a family of subtrees of a tree, interval graphs can be seen as the intersection graphs of subpaths of a path.

The previous characterization led to the first linear-time algorithm to recognize interval graphs: Booth and Lueker proved in 1976 that testing for the consecutives 1's property can be done in linear time [24]. To do so, they introduced a novel data structure called PQ-trees, used to represent all permutations of a set $X$ which are consistent with a set of constraints of consecutivity. More details on PQ-trees are given in the next subsection.

The last characterization that we present is based on the observation that given an interval graph, there exists a linear order on its vertices that satisfies a specific property.

**Theorem 3.5** ([128, 125])**.** *A graph is an interval graph if and only if there exists a linear order $\prec$ on $V$ such that, for every choice of vertices $u, v, w$ with $u \prec v \prec w$, $(u, w) \in E$ implies $(u, v) \in E$.*

It is easy to see that given an interval representation of a graph, the ordering of the left (resp. right) endpoints of the intervals satisfies the previous condition. This characterization has been used to design efficient algorithms for problems like dominating set or coloring [128, 125].

### 3.2.2. Recognition algorithms

In this subsection, we discuss some of the most relevant recognition and isomorphism-testing algorithms for interval graphs. We start by presenting the PQ-tree structure introduced by Booth and Lueker in 1976, which gave the first linear-time recognition algorithm, and a simplification of this structure called modified PQ-trees. We finish with a brief overview of more recent recognition algorithms.

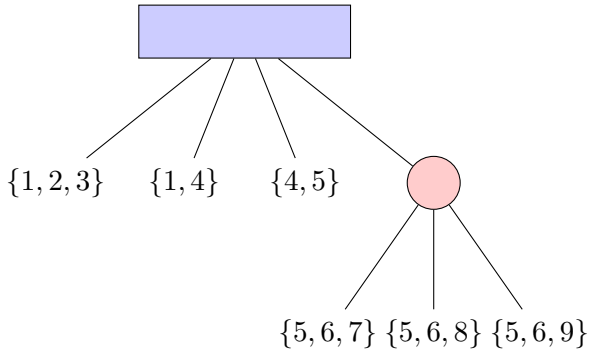



Figure 3.3.: PQ-tree of an interval graph. Q-nodes are represented by blue rectangles, and P-nodes are represented by red circles. Each maximal clique is contained in a leaf and represented by the set of vertices it contains.

**PQ-trees** Given a finite set $X$ and a collection $\mathcal{I}$ of subsets of $X$, PQ-trees serve to represent all the permutations of $X$ in which the members of each subset $I \in \mathcal{I}$ appear as a consecutive subsequence of the permutation. Formally, a PQ-tree $T$ is a rooted tree whose leaves are bijectively labeled by the elements of the set $X$ and whose internal nodes are of two types, each giving a different constraint on the ordering of its children:

- P-nodes: its children can occur in an arbitrary order.
- Q-nodes: its children must occur in the given order (up to reversal).

P-nodes are represented by a circle and Q-nodes by a square. The *frontier* of a tree $T$ is the permutation of $X$ obtained by reading the labels of its leaves from left to right. Two PQ-trees are equivalent if one can be obtained from the other by a sequence of two operations: arbitrarily permuting the children of a P-node, or reversing the children of a Q-node. In the words of Golumbic, "we obtain an equivalent tree by regarding $T$ as a mobile and exposing it to a gentle summer breeze" [78].

As we mentioned before, PQ-trees can be used to test the consecutive 1's property of the clique matrix of an interval graph. To do so, we represent an interval graph as a PQ-tree where the set $X$ is the set of maximal cliques (the rows of the matrix), and the family $\mathcal{I}$ is composed of sets of all the cliques that share a same vertex $v$ (the subsets of $X$ consisting of rows that contain a 1 in the same column), for every vertex $v \in V$. If there exists a PQ-tree satisfying the consecutivity constraints imposed by $\mathcal{I}$, then the graph is an interval graph. This yields an $O(n + m)$ algorithm to test whether a graph is interval, where $n$ and $m$ are the number of vertices and edges, respectively. An example of a PQ-tree is given in Figure 3.3, and the interval representation of the graph associated to it is given in Figure 3.4.

**Modified PQ-trees** Modified PQ-trees were introduced by Korte and Möhring in 1985 to give fast solutions to the seriation with order constraints problem [104] and applied later on to design a simpler algorithm for the recognition of interval graphs [105]. The

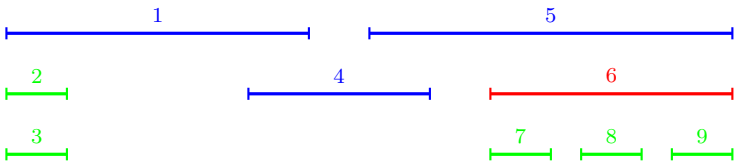



Figure 3.4.: Interval representation of the graph associated to the PQ-tree in Figure 3.3.

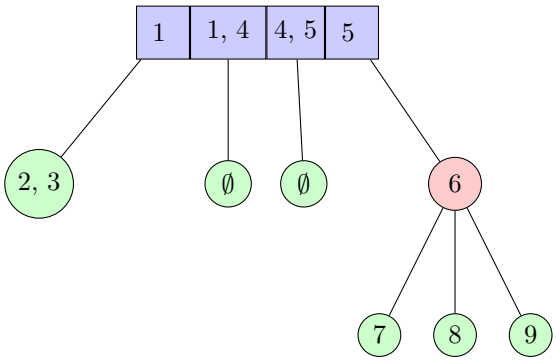



Figure 3.5.: MPQ-tree of the interval graph associated to the representation in Figure 3.4. Each section of a Q-node is represented by a blue rectangle, P-nodes are represented by red circles, and leaves are represented by green circles. Here, leaves contain vertices instead of maximal cliques, and inner nodes also contain a (possibly empty) set of vertices.

modified PQ-tree model (MPQ-tree) is a simplification of the standard PQ-tree model. It assigns a (possibly empty) set of vertices to each node of the PQ-tree: P-nodes are assigned one set while Q-nodes are assigned one set for each of its children (and each set is ordered from left to right according to the ordering of the children, which is unique up to reversal).

More concretely, to a P-node $P$, we assign the set of vertices of $G$ contained in all the maximal cliques represented by the subtree rooted at $P$ but which do not appear in any other cliques. On the other hand, to a Q-node $Q$ with children $Q_1, \ldots, Q_k$, (ordered from left to right), we assign a set $S_i$, called a section, for every $Q_i$. Section $S_i$ contains all the vertices that are contained in all maximal cliques of the subtree rooted at $Q_i$ and in at least one other subtree rooted at some $Q_j$ $(j \neq i)$, but which do not appear in any clique belonging to some other subtree that is not below $Q$. The MPQ-tree has the property that every vertex of $G$ appears exactly in one leaf, in one P-node or in consecutive sections of a Q-node. This data structure also yields an $O(n+m)$ recognition algorithm for interval graphs, but allows for a simpler algorithm and a more informative representation of interval graphs. An example of an MPQ-tree can be seen in Figure 3.5.

**Other recognition algorithms** Habib et al. gave the first linear-time recognition algorithm that did not make use of any complicated pre-processing steps (such as computing

PQ-trees or modular decompositions) [86]. To do so, they employed LexBFS and a clique partition refinement approach. This was later improved by Corneil, Olariu and Stewart, who proposed an algorithm that recognizes interval graphs by six iterations of LexBFS [45]. Li and Wu simplified it to only four sweeps [111]. Cao proposed a simpler self-contained proof of the algorithm [33].

### 3.2.3. Proper and unit interval graphs

Two of the most important subclasses of interval graphs are proper and unit interval graphs.

**Definition 3.8.** *An interval graph is* proper *if it has an interval representation where no interval is properly contained in another one, and it is* unit *if it has a representation where all the intervals have unit length (equivalently, the same length).*

Proper interval graphs were introduced by Roberts in 1969 as "indifference" graphs, motivated by the theory of preference and indifference in economics and psychology [133]. When trying to model preference between a set of elements, the most traditional model considers that given a pair of distinct elements, one can be preferred over the other or there can be indifference between them. That is, we equip the set of elements $X$ with two binary relations:

- An asymmetric preference relation $P$: $(a, b) \in P$ if "$a$ is preferred to $b$".
- A reflexive and symmetric indifference relation $I$: $(a, b) \in I$ if "$a$ and $b$ are indifferent".

The starting point of Roberts work was the characterization of semiorders given by Scott and Suppes [139], which states that a binary relation $P$ on a finite set $X$ is a semiorder if and only if there exists a real-valued function $f$ on $X$ such that for all $x, y \in X$,

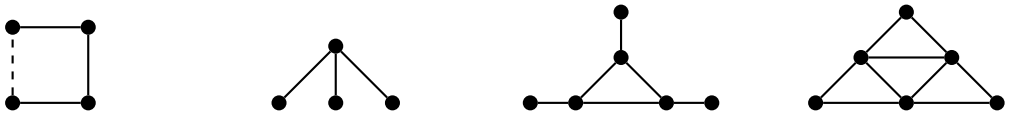
$$xPy \iff f(x) > f(y) + 1$$

Before going any further, let us give Scott-Suppes definition of semiorder, which simplifies the original definition given by Luce in 1956 [114].

**Definition 3.9.** *A partial order relation $I$ defined on a set $X$ is a* semiorder *if the following conditions hold:*

1. *$I$ is irreflexive (i.e., $(x, x) \notin I$).*
2. *$(x, y) \in I$ and $(z, w) \in I$ imply $(x, w) \in I$ or $(z, y) \in I$.*
3. *$(x, y) \in I$ and $(y, z) \in I$ imply that for every $w$, $(x, w) \in I$ or $(w, z) \in I$.*

Roberts extended the work of Scott and Suppes for indifference relations. That is, given a graph $(S, I)$, viewed as a finite set of points $S$ and a reflexive, symmetric binary



(a) $C_k$ with $k \geqslant 4$ (b) $K_{1,3}$ (c) Net (d) Tent

Figure 3.6.: Forbidden induced subgraphs for unit interval graphs.

relation $I$ on $S$ (which represents indifference), he addressed the question of whether there exists a real-valued function $f$ on $S$ such that for all $x, y \in S$,

$$xIy \iff |f(x) - f(y)| \leqslant 1$$

He proved that graphs which are representable in that way correspond exactly to unit interval graphs, along with other characterizations of this class of graphs. Below, we summarize these results.

**Theorem 3.6** ([131])**.** *Let $G = (V, E)$ be an undirected graph. Then, the following are equivalent:*

1. *There exists a function $f : V \longrightarrow \mathbb{R}$ such that $(u, v) \in E$ if and only if $|f(u) - f(v)| \leqslant 1$.*
2. *There exists a semiorder $I$ on the vertices $V$ such that $(u, v) \in I$ if and only in $(u, v) \notin E$.*
3. *$\overline{G}$ is a comparability graph and every transitive orientation of $\overline{G}$ is a semiorder.*
4. *$G$ is an interval graph with no induced $K_{1,3}$.*
5. *$G$ is a proper interval graph.*
6. *$G$ is a unit interval graph.*

Constructive proofs of the implications $K_{1,3}$-free $\implies$ proper $\implies$ unit have been then provided in [21] and [69], the later giving a linear time algorithm to construct a unit interval representation given a proper one, while the former is quadratic in the number of vertices.

Roberts also strove to find all the minimal not representable graphs (graphs that do not admit a proper interval representation), yielding the famous characterization of unit interval graphs in terms of a family of forbidden induced subgraphs.

**Theorem 3.7** ([131])**.** *Let $G$ be a unit interval graph. Then, $G$ does not contain the graphs $K_{1,3}$, the tent, the net or a $C_n$ with $n > 3$ as an induced subgraph (see Figure 3.6 for an illustration of the list of forbidden subgraphs).*

As for arbitrary interval graphs, we can also study the properties of the clique matrix of unit interval graphs. In fact, a connected unit interval graph has a unique clique path, up to reversal [126]. This can be expressed in terms of the clique matrix.

**Theorem 3.8** ([51]). *A graph $G$ is a proper interval graph if and only if its clique matrix has the consecutive ones property for both rows and columns.*

Finally, a graph is a unit interval graph if and only if it admits a specific ordering of its vertices called an *umbrella ordering*.

**Theorem 3.9** ([113]). *A graph $G$ is a unit interval graph if and only if there exists a linear order $\prec$ of $V$ such that for every choice of vertices $u, v, w$, if $u \prec v \prec w$ and $(u, w) \in E$, then $(u, v), (v, w) \in E$.*

Given a proper interval representation, the left endpoints of all vertices, from the smallest to the largest with ties broken arbitrarily, induce an umbrella ordering of the vertices.

Finally, with respect to the recognition, proper interval graphs can also be recognized in linear time [113, 44, 33].

**Generalizations of unit and proper interval graphs.** There are different graph classes that lie between interval and proper and unit interval graphs. The class of *$k$-lengths* interval graphs aims to generalize unit interval graphs by allowing $k$ different interval lengths within the representation [34, 108, 62]. The class of *mixed unit* interval graphs allows all the combinations of open and closed endpoints of the intervals within the representation [53]. The class of *$k$-proper* interval graphs generalizes proper interval graphs by allowing an interval to be properly contained in at most $k$ intervals [127]. On the other hand, *$k$-nested* interval graphs forbid chains of more than $k$ nested intervals [103]. Finally, *rigid* interval graphs are those that have a unique clique tree [110].

**Forbidden patterns.** To conclude this section, note that all the families of intersection graphs satisfy the hereditary property, so they can be characterized by the set of their minimal forbidden subgraphs. However, this set can be infinite, like in the case of chordal or interval graphs, where we exclude every $C_k$ for $k \geqslant 4$. To address this issue, characterizations of hereditary graph classes in terms of forbidden patterns in a vertex ordering or layout have also been studied [142, 50].

An *ordered graph* is a pair $(G, \prec_G)$ such that $\prec_G$ is a total ordering of $V(G)$. An ordered graph $(H, \prec_H)$ is a *pattern* on $(G, \prec_G)$ if $H$ is an induced subgraph of $G$ and for every pair of vertices $x$ and $y$ of $H$, $x \prec_G y$ if and only if $x \prec_H y$. Patterns are often denoted as a set of (ordered) edges and non-edges. A graph $G$ *excludes a pattern* $(H, \prec_H)$ if there exists a layout $\prec_G$ of $G$ such that $(H, \prec_H)$ is not a pattern of $(G, \prec_G)$. As an example, the class of chordal graphs can be characterized by excluding the pattern $\{(\overline{12}, 13, 23)\}$, while interval graphs and proper interval graphs can be characterized by excluding the sets of patterns $\{(\overline{12}, 13, 23), (\overline{12}, 13, \overline{2, 3})\}$ and $\{(12, 13, \overline{23}), (\overline{12}, 13, 23)\}$, respectively [50, 125].

## 3.3. Generalizations of interval graphs

We have seen that not every graph is an interval graph: already simple graphs as cycles of length four fail to belong to this class. Thus, a natural question arises: can we extend the idea of representing graphs by intersections of intervals to all graphs? One approach, introduced by Roberts, is to consider higher dimensional intervals. He defined a new graph-parameter, called *boxicity*, that measures the smallest integer $t$ such that a graph can be represented by the intersection of $t$-dimensional boxes which have their edges parallel to the coordinate axes [132]. Interval graphs are those graphs with boxicity at most one. Another approach, which is the one that we will study here, is allowing each vertex to be represented by the union of several intervals. This yields another natural graph parameter, the *interval number*, which is closely related to the class of multiple interval graphs.

### 3.3.1. Multiple interval graphs

Before giving an overview of the most important results concerning the interval number of a graph, let us give some preliminary definitions:

**Definition 3.10.** *For any natural number $d > 0$, a (disjoint) $d$-*interval *is the union of $d$ (disjoint) intervals on the real line.*

**Definition 3.11.** *For any natural number $d > 0$, a graph $G$ is a (disjoint) $d$-*interval graph *if there exists a bijection from the vertices of $G$ to a set of (disjoint) $d$-intervals, $f : V \to \mathcal{I}$, such that there exists an edge between two vertices if and only if their corresponding $d$-intervals intersect. The set $\mathcal{I}$ of $d$-intervals is called a $d$-*interval representation *of $G$, and the family of all intervals that compose the $d$-intervals in $\mathcal{I}$ is called the* underlying family of intervals *of $\mathcal{I}$.*

Note that in the literature, $d$-interval graphs have been defined both as the union of $d$ disjoint intervals [13, 31, 148], as the union of $d$ *not necessarily disjoint* intervals [145], and simply as the union of $d$ intervals, without specifying whether they are disjoint or not [58, 138]. When there are no length restrictions on the intervals, this ambiguity is not relevant, as both definitions lead to the same class of graphs, as one can simply stretch the intervals associated to a same vertex that intersect to make them disjoint without changing any of the other intersections.

**Observation 3.1.** *The classes of disjoint $d$-interval and $d$-interval graphs are equivalent.*

*Proof.* It is clear that the class of disjoint $d$-interval graphs is contained in the class of $d$-interval graphs. To see the other direction, it suffices to notice that if we have a $d$-interval representation, we can represent every pair of intersecting intervals $[a, b]$ and $[c, d]$ (with $a < c < b < d$), associated to the same vertex $v$, by a single interval $[a, d]$ to obtain an equivalent disjoint $d$-interval representation (note that since we have decreased the number of intervals associated to vertex $v$ by one, to obtain a $d$-interval representation, one would technically need to add a "dummy" interval associated to $v$ which does not intersect any other interval in the representation). □

Let us now define the depth of a multiple interval graph, which measures a property of its interval representations.

**Definition 3.12.** *The* depth *of a family of intervals is the maximum number of intervals that share a common point, and for every $d \geqslant 1$, the* representation depth *of a $d$-interval graph is the minimum depth of any $d$-interval representation of the graph. We use the term "depth-$r$" as an abbreviation of "representation depth at most $r$".*

Now, given the definition of $d$-interval graphs, we can define the interval number of a graph as follows.

**Definition 3.13.** *Given a graph $G$, the* interval number $i(G)$ *is the smallest integer $d$ such that $G$ is a $d$-interval graph.*

The interval number $i(G)$ of a graph $G$ is well-defined, as it is always smaller than the number of vertices of $G$, and a graph is an interval graph if and only if its interval number is one. This definition was first introduced by R. McGuigan in 1977 [117], and independently, by Trotter and Harary [145], who motivated its study with situations arising naturally in scheduling and allocation problems. The two seminal papers on the interval number of a graph, by Trotter and Harary [145] and Griggs and West [80], study different bounds on the interval number of some classes of graphs. We gather the most important results below.

**Theorem 3.10** ([145])**.** *The interval number of a tree is 1 if it is a caterpillar and 2 otherwise.*

**Theorem 3.11** ([145])**.** *The interval number of the complete bipartite graph $K_{m,n}$ is $\left\lceil \frac{mn+1}{m+n} \right\rceil$.*

**Theorem 3.12** ([80])**.** *Let $G$ be a simple graph on $n$ vertices and $m > 0$ edges which contains no $K_3$. Then $i(G) \geqslant \left\lceil \frac{m+1}{n} \right\rceil$.*

**Theorem 3.13** ([80])**.** *Let $G$ be a simple graph on $n$ vertices and $m > 0$ edges. Then, $i(G) \leqslant \lceil \sqrt{m} \rceil$.*

**Theorem 3.14** ([80])**.** *Let $G$ be a graph with maximum degree $\Delta$, then $i(G) \leqslant \left\lceil \frac{1}{2}(\Delta + 1) \right\rceil$.*

Both Trotter and Harary and, independently, Griggs and West, established the following upper bound on the interval number for general graphs: $i(G) \leqslant \left\lceil \frac{1}{3}n \right\rceil$, where $n = |V(G)|$. This was then improved by Griggs.

**Theorem 3.15** ([79])**.** *Let $G$ be a graph with $n > 1$ vertices. Then $i(G) \leqslant \left\lceil \frac{1}{4}(n+1) \right\rceil$. This bound is the best possible.*

Finally, both left as an open problem to find a minimal family of graphs that are not representable by $d$-intervals. However, in the 1980s, West and Shmoys proved that for any fixed value of $d$ with $d \geqslant 2$, determining whether the interval number of a

graph is smaller or equal to $d$ is NP-complete [148], which made the possibility of the existence of a "nice" characterization of graphs with interval number 2 (and greater than 2) unlikely. They proved that the recognition problem is NP-hard for $d = 2$, by reducing from HAMILTONIAN CYCLE in 3-regular triangle-free graphs, and then reduced from the case $d - 1$ to $d$. As a corollary, they obtained the following theorem.

**Theorem 3.16** ([148])**.** *For any $d \geqslant 2$ and any $r \geqslant 3$, determining whether a graph $G$ has a $d$-interval representation of depth at most $r$ is* NP*-complete.*

They also motivated the study of $d$-interval graphs with a new biological problem. They observed that new discoveries had shown that many genes are not represented as a single unbroken sequence, but rather as a collection of them [35], and so multiple interval graphs could be a useful tool for studying the structure of genes, since finding interval numbers could be used to test gene compositions if an upper bound could be given on the number of intervals used for each gene.

From the algorithmic point of view, the class of multiple interval graphs does not share the algorithmic advantages of interval graphs. Different optimization problems have been studied in multiple interval graphs, both from the classical complexity perspective [31, 64, 83] and the parameterized complexity one [59, 95]. It is known that MAXIMUM CLIQUE remains NP-complete in multiple interval graphs, even for unit 2-intervals [64], and so do other problems such as INDEPENDENT SET or DOMINATING SET [13, 31]. Some of the hardness results, for example, for vertex cover or coloring, follow because the problems are already NP-complete on graphs of maximum degree three or cubic graphs, which are properly contained in the class of 2-interval graphs. With respect to the parameterized complexity, Fellows et al. proved that the problems $k$-INDEPENDENT SET and $k$-DOMINATING SET are both W[1]-hard even in unit 2-interval graphs, whereas $k$-CLIQUE, which is W[1]-hard on general graphs, becomes FPT in multiple interval graphs [59].

**Subclasses of multiple interval graphs** As for classical interval graphs, the study of $d$-interval graphs soon drew attention to some of its subclasses, such as proper $d$-interval graphs and unit $d$-interval graphs, which as for interval graphs, are equivalent. Let us first give the formal definitions.

**Definition 3.14.** *A (disjoint) $d$-interval graph is* proper *if there exists a representation where no interval of the underlying family is properly contained in another one, and it is* unit *if there exists a (disjoint) $d$-interval representation where all the intervals of the underlying family have unit length.*

We have seen in Observation 3.1 that the classes of $d$-interval and disjoint $d$-interval graphs are equivalent. However, if there are length restrictions on the intervals (like in the case of unit $d$-interval graphs), this observation does not hold. For unit intervals, one cannot replace two intersecting intervals $[a, b]$ and $[c, d]$, with $a < c < b < d$, by $[a, d]$, as the resulting interval would not be of unit length, and stretching it to make it unit might disrupt the rest of the intersections. Thus, in this case, it cannot be inferred

that both definitions of $d$-intervals lead to the same class of graphs. In fact, throughout this thesis we will prove that they do not, so we will always distinguish between both definitions of $d$-intervals. Nevertheless, this distinction has not always been explicit in the literature.

The *unit interval number* was formally introduced by Andreae in 1985 [6].

**Definition 3.15.** *The* unit interval number *of a graph, denoted $i_u(G)$, is the minimum $d$ such that $G$ has a $d$-interval representation that is proper.*

However, this notion had already been studied before by Maas [115], who proved the following result.

**Theorem 3.17** ([115])**.** *A connected triangle-free graph of maximum degree $\Delta$ has a proper $d$-interval representation if and only if $d > \Delta/2$, and there is a vertex $u$ with degree smaller than $2d$.*

Andreae then studied the unit interval number on general graphs, and gave the following upper bound.

**Theorem 3.18** ([6])**.** *Let $G$ be an arbitrary graph on $n$ vertices. Then, $i_u(G) \leqslant \left\lceil \frac{1}{2}(n+1) \right\rceil$.*

The concrete applications of 2-intervals, namely in bioinformatics, motivated the study of another interesting restriction of 2-intervals: *balanced 2-intervals*. In [48], the authors introduced families of balanced 2-intervals to model RNA. This led, later on, to the study of balanced 2-interval graphs [67]. Let us first give the formal definitions:

**Definition 3.16.** *A (disjoint) $d$-interval graph is* balanced *if there exists a (disjoint) $d$-interval representation where the $d$ intervals of a same $d$-interval have the same length, but intervals of different $d$-intervals can differ in length.*

**Definition 3.17.** *A graph $G$ is a (disjoint) balanced $d$-interval graph if it is the intersection graph of a family of (disjoint) balanced $d$-intervals.*

Gambette and Vialette noticed that the proof of NP-completeness given by West and Shmoys for the recognition of 2-interval graphs could be adapted for the recognition of balanced 2-interval graphs. They also showed that the class of balanced 2-interval graphs is properly contained in the class of 2-interval graphs, and initiated the study of the complexity of recognizing unit 2-interval graphs [67].

The last important restriction of $d$-interval graphs that we will introduce here is the class of $(x_1, \ldots, x_d)$ $d$-interval graphs.

**Definition 3.18.** *A disjoint $d$-interval is a $(x_1, \ldots, x_d)$ $d$-interval if the $d$ disjoint intervals are open, have integer endpoints, and have lengths $x_1, \ldots, x_d$, respectively.*

Gambette and Vialette showed that the class of disjoint unit 2-interval graphs is equal to the union of the classes of $(x, x)$-interval graphs, for every $x \in \mathbb{N}$ [67]. However, they left the recognition of both (disjoint) unit 2-interval graphs and $(x, x)$-interval graphs

for $x \geqslant 2$ as an open question (notice that $(1,1)$-interval graphs are exactly line graphs, which can be recognized in linear time [136]).

These open questions were partially addressed by Jiang [96], who managed to prove that depth-two disjoint unit $d$-interval graphs can be recognized in linear time, and gave an approximation algorithm for recognizing depth-two unrestricted $d$-interval graphs with an additive error of one, that is, an algorithm that determines either that $G$ is not a depth-two $d$-interval graph, or that it is a depth-two $(d+1)$-interval graph. Nevertheless, whether arbitrary (disjoint) unit $d$-interval graphs or $(x,x)$-interval graphs can be recognized in polynomial time remained as the two main open questions.

### 3.3.2. Multiple track interval graphs

Multiple interval graphs are not the only generalization of interval graphs where every vertex is represented by more than one interval. We now present another closely related class of graphs, the class of *multiple track interval graphs*, where each of the intervals that represent a same vertex must be placed in a different *track*.

**Definition 3.19.** *For any natural number $d > 0$, a* $d$-track interval *is the union of $d$ intervals, each placed in a separate parallel line, called* track*, with each track being disjoint from the others.*

**Definition 3.20.** *A graph $G$ is a* $d$-track interval graph *if it is the intersection graph of a family of $d$-track intervals.*

**Definition 3.21.** *Given a graph $G$, we define the* multitrack interval number *or* track number *of a graph $G$, denoted $t(G)$, as the minimum positive integer $d$ such that $G$ is a $d$-track interval graph.*

The term multiple track interval was coined by Gyárfas and West [85], although the concept had already been proposed by Tibor Gallai in 1968, and studied by Gyárfás and Lehel in [84]. On the other hand, the study of multiple track interval *graphs* was initiated by Kumar and Deo in [107]. Kumar and Deo gave a lower bound on the track number of complete bipartite graphs, analogous to the bound given by Trotter and Harary for the interval number.

**Theorem 3.19** ([107])**.** *Let $K_{m,n}$ be a complete bipartite graph. Then $t(K_{m,n}) \geqslant \left\lceil \frac{mn}{(m+n-1)} \right\rceil$.*

They also stated that it was likely that the recognition of graphs with fixed track number could be solved in polynomial time. However, this was proven false by Gyárfás and West [85].

**Theorem 3.20** ([85])**.** *Recognizing 2-track interval graphs is* NP*-complete.*

Gyárfás and West adapted the proof given by West and Shmoys for 2-interval graphs to prove the previous theorem. Nevertheless, they did not manage to extend the hardness result for $d$-track interval graphs with $d > 2$, which was left as an open question, together

| | $d$-Interval Graphs | $d$-Track Interval Graphs |
|---|---|---|
| Unrestricted | NP-complete | NP-complete |
| Balanced | NP-complete ($d = 2$) | NP-complete |
| Unit | ? | NP-complete |
| $(2, \ldots, 2)$ | ? | NP-complete |
| Depth-two | ? (+1 approximation) | NP-complete ($d = 2$) |
| Depth-two, Unit | Linear-time | NP-complete ($d = 2$) |

Table 3.1.: Complexity of recognizing different subclasses of $d$-interval and $d$-track interval graphs before the results presented in this manuscript [67, 96].

with the recognition of depth-2 multitrack interval graphs. It took almost twenty years to close the gap between $d = 2$ and $d > 2$. In 2010, Jiang finally proved that recognizing $d$-track interval graphs is NP-complete for every $d \geqslant 2$, even for proper subclasses like balanced, unit, depth-two, depth-two unit and $(2, \ldots, 2)$ $d$-track interval graphs.

We finish this section with a summary of the complexities of recognizing the different subclasses of $d$-interval graphs (see Table 3.1), and a summary of the known containment relations between some of these subclasses (see Figure 3.7).

## 3.4. Structure of the thesis

The remaining of the manuscript is dedicated to the original results obtained during this thesis. Before presenting the core results, we discuss in Chapter 4 how to certify that a given graph belongs to the class of (disjoint) unit $d$-interval graphs. We give a characterization based on splitting vertices to obtain a unit interval graph and explain how to encode it in Answer Set Programming to solve the recognition problem efficiently in practice. Chapter 5 is dedicated to investigating whether we can generalize Roberts characterization of unit interval graphs to higher dimensions. After observing that this is not possible in the general case, we restrict ourselves to $d$-interval graphs that are also interval. We generalize Roberts characterization for (non-disjoint) unit $d$-interval graphs, and prove that it cannot be done for disjoint unit $d$-interval graphs, which implies that the two classes are not equivalent. We also study the relations between the classes of disjoint balanced $d$-interval graphs and (non-disjoint) balanced $d$-intervals. Then, in Chapter 6, we address the two main open questions regarding the complexity of recognizing subclasses of $d$-interval graphs: the complexity of recognizing unit and $(x, \ldots, x)$ $d$-interval graphs. We prove that it is NP-hard to recognize unit (disjoint) $d$-interval graphs and obtain, among other implications, that the recognition of $(x, \ldots, x)$ $d$-interval graphs is also NP-hard for every $x \geqslant 11$. Finally, in Chapter 7 we study the complexity of PIG-completion on interval graphs, an editing problem that consists on finding the minimum amount of edges that we need to add to a given interval graph to make it proper. We prove that the problem is polynomial-time solvable on graphs that contain a universal vertex and other specific settings. For the general case, we provide

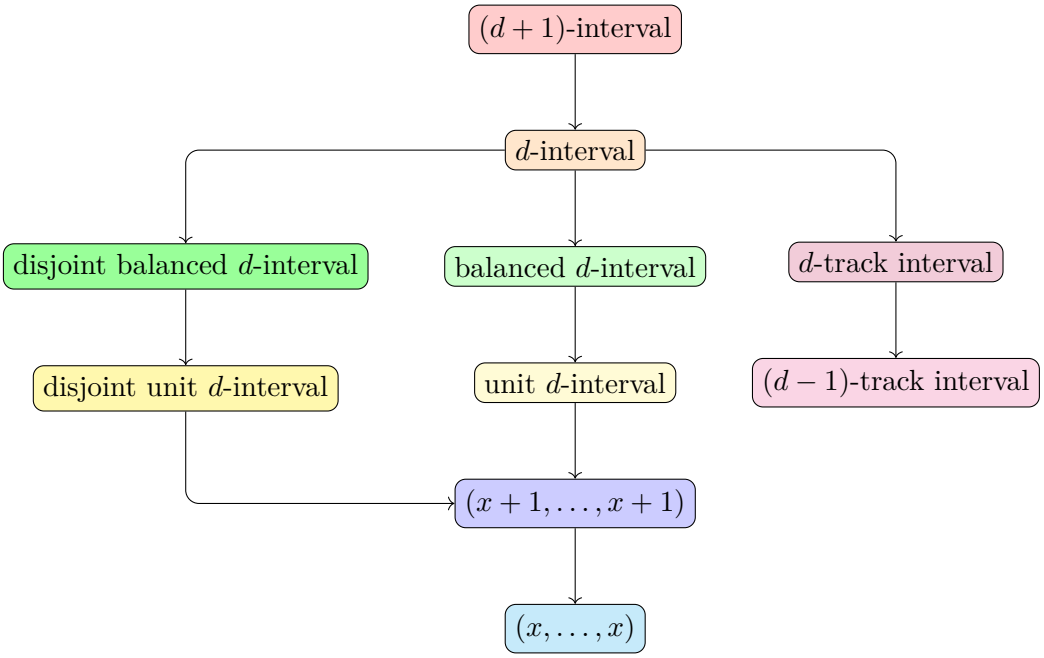



Figure 3.7.: Containment relations between some subclasses of $d$-interval graphs for any $d \geqslant 2$ (before the results presented in this thesis). An arrow from class $\mathcal{C}_1$ to a class $\mathcal{C}_2$ indicates that $\mathcal{C}_2 \subsetneq \mathcal{C}_1$.

an algorithm which is exponential on the number of non-disjoint maximal centers. We conclude with a summary and open questions in Chapter 8.

# 4. Certificates for being a (disjoint) unit 2-interval graph

In this chapter, we discuss the different certificates that we can use in order to establish that a graph is a (disjoint) unit 2-interval graph, which will be a fundamental task throughout the rest of the manuscript. We first highlight, in Section 4.1, the drawbacks of providing a (disjoint) unit 2-interval representation of the graph, and then present in Section 4.2 a combinatorial certificate based on splitting vertices in order to obtain a unit interval graph. Finally, in Section 4.3, we combine this approach with the semiorder characterization of unit interval graph to obtain a recognition algorithm and explain how to encode it in Answer Set Programming (a form of declarative programming oriented towards difficult search problems) to design an efficient software that checks whether a given graph belongs to the class of (disjoint) unit 2-interval graphs. All of the results generalize naturally for (disjoint) unit $d$-interval graphs with $d > 2$.

**Forbidden induced subgraphs.** First of all, let us remark that even though there does not exist a characterization of 2-interval graphs by forbidden induced subgraphs, there exist some graphs that are known to be forbidden induced subgraphs. Indeed, Trotter and Harary showed that a forbidden subgraph characterization of 2-interval graphs must include the complete bipartite graphs $K_{4,4}$ and $K_{3,6}$ [145]. Although such a characterization is not complete and seems hard to find if it exists, a partial characterization can be a useful tool to quickly determine that some graphs do not belong to this class.

For (disjoint) unit 2-interval graphs, it is straightforward to see that we have an additional forbidden induced subgraph: the complete bipartite graph $K_{1,5}$. Roberts showed that a unit interval graph cannot contain an induced $K_{1,3}$, as it is impossible for an interval of unit length to intersect three pairwise disjoint intervals of length one (see Figure 4.1). This observation can be generalized for (disjoint) unit 2-interval graphs. Indeed, if a graph contains an induced $K_{1,5}$, then it cannot be a (disjoint) unit 2-interval graph, as the two intervals associated to the center of the claw would have to intersect five pairwise disjoint intervals. Then, by the pigeonhole principle, one of the intervals associated to the center would have to intersect at least three pairwise disjoint intervals, which we already know to be impossible (see Figure 4.2). Naturally, this extends to (disjoint) unit $d$-interval graphs for any $d > 2$: if we try to give a (disjoint) unit $d$-interval representation of a graph that contains a $K_{1,2d+1}$ as an induced subgraph, then the $d$ intervals associated to the center must intersect $2d + 1$ pairwise disjoint intervals, which again implies that we must have an interval of length one intersecting three pairwise disjoint intervals of length one.

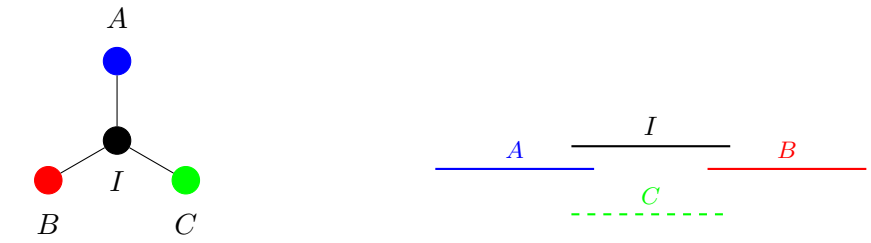



Figure 4.1.: A unit interval graph cannot contain an induced $K_{1,3}$.

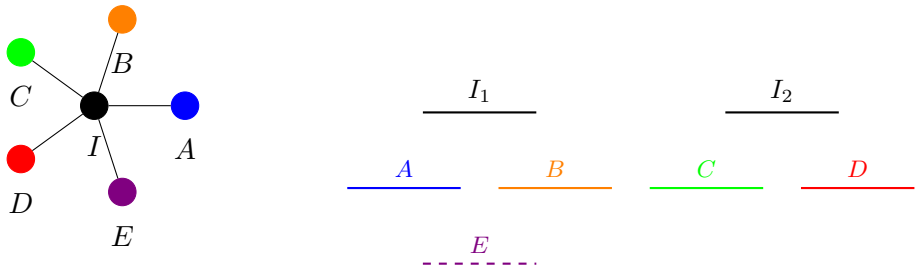



Figure 4.2.: A (disjoint) unit 2-interval graph cannot contain an induced $K_{1,5}$.

**Observation 4.1.** *Let $G$ be a (disjoint) unit $d$-interval graph for some natural number $d \geqslant 1$. Then, $G$ does not contain the graph $K_{1,2d+1}$ as an induced subgraph.*

*Proof.* Towards a contradiction, suppose that there exists a graph $G$ that is (disjoint) unit $d$-interval and contains a $K_{1,2d+1}$ as an induced subgraph. Then, there exists a (disjoint) unit $d$-interval representation of the induced $K_{1,2d+1}$, which means that the $d$-interval associated to the center must intersect $2d + 1$ pairwise disjoint $d$-intervals. Since the $d$ intervals associated to the center all have unit length, each of them can intersect at most two different $d$-intervals. That is, in total, the $d$-interval associated to the center can intersect at most $2d$ pairwise disjoint $d$-intervals, which contraadicts the assumption that $G$ is (disjoint) unit $d$-interval. □

## 4.1. Exhibiting a (disjoint) unit 2-interval representation

The most straightforward way to prove that a given graph $G = (V, E)$ is a (disjoint) unit 2-interval graph is to provide a (disjoint) unit 2-interval representation of the graph. That is, to every vertex $v \in V$, we assign two unit intervals $I_{v_1}$ and $I_{v_2}$, and for every edge $(u, v) \in E$, we require the interval $I_{u_i}$ to intersect the interval $I_{v_j}$, for some $i, j \in \{1, 2\}$. This can be manufactured easily for some small graphs, but if we want to determine whether an arbitrary graph is (disjoint) unit 2-interval, exploring by brute-force all the possible such representations of a graph is not an efficient approach. In fact, without

additional constraints, this is simply impossible: just by translating a single interval we can obtain an infinite number of representations.

**From $K_{1,3}$-free to unit.** In the case of small graphs for which it is easy to give a representation, the simplest procedure is to provide a (disjoint) 2-interval representation where no interval of the underlying family intersects three pairwise disjoint intervals. Then, if we regard the interval representation given by the underlying family, it will correspond to a representation of a $K_{1,3}$-free interval graph, or equivalently, as Roberts showed, to a unit interval graph. We can thus claim directly that the graph is (disjoint) unit 2-interval, or transform the representation into a unit one. Recall that there exists a constructive proof of Roberts characterization that provides an algorithm to convert an interval representation of a $K_{1,3}$-free interval graph into a proper one, and then to a unit one [21].

We describe the algorithm below for completeness. Given an interval representation of a $K_{1,3}$-free graph, we know that no interval intersects three pairwise disjoint intervals. Thus, if there is an interval $J = [a, b]$ properly contained in an interval $I = [c, d]$, with $c < a \leqslant b < d$, then one of the segments $[c, a]$ or $[b, d]$ will be free of endpoints of intervals that don't intersect $J$. Thus, we can extend $J$ past one of the endpoints of $I$ without altering the intersections. Doing so for every interval properly contained in another one yields a proper representation. The next step is to transform the obtained representation into a unit one. Note that, given a proper interval representation, the right endpoints of the intervals follow the same order as the left endpoints. Therefore, we can process the representation from left to right, and at each step, we consider the interval $I = [a, b]$ with the leftmost left endpoint that has not been converted to a unit interval and we make it unit. The goal is to stretch $I$ without altering the lengths of the already processed intervals, so we will only stretch the part of the interval that does not intersect any already processed intervals. Thus, suppose that $I$ contains the right endpoint of some other interval and let $\alpha$ be the largest such right endpoint. This interval must have been processed before, and so it is already of length one, which implies that $\alpha < \min\{a + 1, b\}$. If no such interval exists, we simply let $\alpha = a$. We can then adjust the part of the representation in $[\alpha, \infty)$ by shrinking or expanding $[\alpha, b]$ to $[\alpha, a + 1]$, and translating $[b, \infty)$ to $[a + 1, \infty)$. After we have processed every interval, we obtain a unit interval representation. The procedure described above is quadratic on the number of intervals.

Gardi proposed a more efficient procedure that yields a linear time algorithm, as it avoids translating all the remaining intervals at each step [69]. To do so, he exploited the fact that the clique-vertex incidence matrix of a proper interval graph has the consecutive 1's property for columns and rows. The algorithm starts with the vertex ordering given by the order of the left endpoint of the intervals. From the consecutive 1's property, we know that the vertices of a maximal clique appear consecutively in this ordering. For a clique $C_i$, we denote by $a(i)$ and $b(i)$ the indices of the first and last vertex on the ordering that belongs to the clique, respectively. The clique $C_i$ is then formed by the vertices $a(i), \ldots, b(i)$. We start by representing the vertices of clique $C_1$ by $b(1)$

pairwise intersecting unit intervals, whose left endpoints are ordered as in the vertex order $v_1, \ldots, v_{b(1)}$. Now, suppose we have assigned unit intervals to every vertex in the cliques $C_1$ until $C_{j-1}$ and consider clique $C_j$. To add the remaining intervals of $C_j - C_{j-1}$, for every vertex $v$ in the aforementioned set, we construct $I_v = [l(I_v), r(I_v)]$ as follows. We consider the order of the vertices of $C_j - C_{j-1}$ in the vertex ordering and put the left endpoints of the intervals associated to them in said ordering (that is, $l(I_{v_{b(j-1)+1}}), \ldots, l(I_{v_{b(j)}})$) between $r(I_{v_{a(j)-1}})$ and $r(I_{v_{a(j)}})$. In other words, all the intervals start after the last interval that is in $C_{j-1}$ but not in $C_j$, and before the first interval of $C_j$ finishes. This algorithm yields a unit interval representation of the graph.

In the remaining of the manuscript, we will often provide a proper 2-interval representation or a 2-interval representation that satisfies the property that no interval of the underlying family intersects three pairwise disjoint intervals as a certificate that a graph is unit 2-interval.

**Computer search.** For larger graphs, one can seek to find a (disjoint) unit 2-interval representation by computer search. As mentioned before, in this case the search space can explode, so additional constraints must be imposed to reduce it.

The first essential step is to discretize the real line by requiring all intervals to have integer endpoints. This will also likely force us to modify the length of the intervals: if the initial representation consisted of intervals of length one, discretizing the real line will have as consequence that an interval can only intersect another one if it starts/end at one of its endpoints. Thus, we will not be able to represent graphs that we could represent before, for example, a triangle with two of its vertices connected each to a private neighbor. Therefore, we need to force all the intervals to have the same length, but this length must be larger than one. In order to fix a length that maintains expressivity, i.e., that ensures that we can represent every graph that we could represent before discretizing the real line, we can choose as an upper bound $2n$, where $n$ is the number of vertices in the original graph $G$. Indeed, in a unit 2-interval representation of $G$, there will be exactly $2n$ intervals, and so in the worst case, an interval can intersect all the other intervals at different endpoints.

The other fundamental step to reduce the search space is to force all the intervals to start at the first available integer. This allows us to compress the size of the section of the real line that we need to explore. Since we consider every interval to have length $2n$, the length of a compact representation (a representation where every interval starts in the first available integer) is at most $2n \cdot (2n + 1)$ (we have $2n$ intervals of length $2n$, and since the intervals are closed, disjoint intervals need to start at the next integer, so we account for a section of the real line of length $2n + 1$ for each interval). That is, in total it suffices to consider a section of the real line of length $4n^2 + 2n$.

If we impose these two constraints, then we can reduce the problem to finding the starting point of every interval (which completely determines the representation since all the intervals have the same length) among $4n^2 + 2n$ possibilities (the number of integers in the section of the real line we are considering). A brute-force algorithm to solve this problem would require $2n$ variables, each indicating the starting point of an

interval and taking $4n^2 + 2n$ possible values. In total, that gives roughly $(4n^2 + 2n)^{2n}$ possible representations. Such a brute-force algorithm can be modeled as an ILP, but its running time already anticipates a limited efficiency.

## 4.2. Vertex splitting to make the graph unit interval

We have seen the drawbacks of providing an interval representation as a certificate to show that a graph is (disjoint) unit 2-interval. We now present a characterization of unit and disjoint unit 2-interval graphs that has the advantage of being a truly combinatorial certificate, and which will be an essential tool for proving that a graph belongs to these classes throughout the rest of the manuscript. On a high level, this characterization states that a graph is a (disjoint) unit 2-interval graph if and only if one can obtain a unit interval graph by splitting some (or all) of its vertices.

The *vertex splitting* operation replaces a vertex $v$ by two copies $v_1$ and $v_2$ such that the union of the neighborhoods of the copies is equal to the neighborhood of the original vertex. We will differentiate between *splits*, where the two copies can be adjacent, and *disjoint splits*, where the two copies must be non-adjacent.

Vertex splitting was first introduced in the context of circuit design and graph drawing [116, 56]. Since then, it has been studied with different purposes, most notably in the visualization of non-planar graphs [124, 57] and in graph modification problems [1, 11]. However, the notion of vertex splitting has also been applied to the recognition of trapezoid graphs [36, 119] and tolerance and bounded tolerance graphs [120]. In this context, the definition of this operation is more restrictive, as additional constraints are imposed in the neighborhoods of the two copies of a vertex so that it can be used as a tool to transform a trapezoid graph into a permutation graph with specific properties.

In a similar fashion, we apply the notion of vertex splitting to the recognition of (disjoint) unit $d$-interval graphs. We will first present the characterization of (disjoint) unit 2-interval graphs in terms of splits of a graph, and then generalize if for any $d \geqslant 2$. Let us thus formalize the notions of *split* and *disjoint split* that we use here.

**Definition 4.1.** *Given a graph $G$, a pair $(S, f)$ formed by a graph $S$ and a function $f : V(S) \mapsto V(G)$ is a* split *of $G$ if $f$ satisfies the following conditions:*

- $1 \leqslant |f^{-1}(v)| \leqslant 2$ *for every $v \in V(G)$.*
- *For every edge $(s, t)$ of $S$, $(f(s), f(t))$ is an edge of $G$.*
- *For every edge $(u, v)$ of $G$, there exist two vertices $s$ and $t$ in $f^{-1}(\{u, v\})$ such that $(s, t)$ is an edge of $S$.*

**Definition 4.2.** *Given a graph $G$, a pair $(S, f)$ formed by a graph $S$ and a function $f : V(S) \mapsto V(G)$ is a* disjoint split *of $G$ if $f$ satisfies the conditions in Definition 4.1 and the following extra condition:*

- *For every vertex $v$ of $G$, $f^{-1}(v)$ is an independent set in $S$.*

Given a split $(S, f)$ of $G$ and a vertex $v \in V(G)$, we call the set $f^{-1}(v)$ the set of *representatives* of the vertex $v$. Similarly, given an edge $(u, v) \in E(G)$, the edge $(s, t)$ of $S$ with $s$ and $t$ in $f^{-1}(\{u, v\})$ is called a *representative* of $(u, v)$. If $|f^{-1}(v)| = 1$, we say that the vertex $v$ has not been split.

As we anticipated before, among all the splits of a graph $G$, we are actually interested in those splits that yield a unit interval graph.

**Definition 4.3.** *The family of splits of $G$ that lead to a unit interval graph is $\mathcal{S}^*_{\mathcal{U}}(G) := \{(S, f) \mid (S, f)$ is a split of $G$ and $S$ is a unit interval graph$\}$.*

**Definition 4.4.** *The family of disjoint splits of $G$ that lead to a disjoint unit interval graph is $\mathcal{S}_{\mathcal{U}}(G) := \{(S, f) \mid (S, f)$ is a disjoint split of $G$ and $S$ is a unit interval graph$\}$.*

The next lemma shows how a (disjoint) split $(S, f)$ of a graph $G$ can be used to certify that $G$ is a (disjoint) unit 2-interval graph.

**Lemma 4.1.** *We can characterize (disjoint) unit 2-interval graphs as follows:*

1. *A graph $G$ is a unit 2-interval graph if and only if the family $\mathcal{S}^*_{\mathcal{U}}(G)$ is not empty.*
2. *A graph $G$ is a disjoint unit 2-interval graph if and only if the family $\mathcal{S}_{\mathcal{U}}(G)$ is not empty.*

*Proof.* 1. Suppose that $G$ is a unit 2-interval graph. Then, by assumption, there exists a collection of unit 2-intervals $\mathbf{D} = \{(I_1(v), I_2(v)) \mid v \in V\}$ such that $G \simeq \Omega(\mathbf{D})$. Let $\mathcal{F}$ be the family of intervals formed by the underlying family of $\mathbf{D}$. Let $S$ be the interval graph defined as the intersection graph of the family $\mathcal{F}$, i.e., $S \simeq \Omega(\mathcal{F})$. Consider the function $f : V(S) \mapsto V(G)$ such that:

- For every pair $(I_1(v), I_2(v)) \in \mathbf{D}$, $f(I_1(v)) = f(I_2(v)) = v$.

By construction, $f$ satisfies all the conditions in Definition 4.1. Indeed, the first two conditions follow directly by definition, while the last two conditions follow because if we have an edge $(I_j(u), I_k(v))$ in $S$, for some $j, k \in \{1, 2\}$, this is equivalent to the 2-intervals associated to vertices $u$ and $v$ of $G$ intersecting, so there is an edge $(u, v)$ in $G$. Therefore, $(S, f)$ is a split of $G$.

Conversely, suppose that there exists a split $(S, f)$ of $G$ that satisfies the property of being a unit interval graph. Then, there exists a collection of unit intervals $\mathbf{I} = \{I_1(s) \mid s \in V(S)\}$ such that $S \simeq \Omega(\mathbf{I})$. Since $(S, f)$ is a split of $G$, we know that there exists a map $f : V(S) \mapsto V(G)$ satisfying the conditions in Definition 4.1. We construct a unit 2-interval representation of $G$, i.e., a collection of unit 2-intervals $\mathbf{D} = \{(I_1(v), I_2(v)) \mid v \in V\}$ as follows:

- For every $v \in V(G)$, we let $I_1(v) = I_1(s)$ and $I_2(v) = I_1(t)$, where $\{s, t\} = f^{-1}(v)$.

By construction, this is a unit 2-interval representation of $G$, as the last two properties of $f$ ensure that we preserve the same edges.

2. Similarly, suppose that $G$ is a disjoint unit 2-interval graph and define the split $(S, f)$ as before. Since the family $\mathbf{D}$ is now a family of disjoint unit 2-intervals, the map $f$ satisfies the confition that $f^{-1}(v)$ is an independent set for every vertex $v$, and so $(S, f)$ is a disjoint split. Conversely, given a disjoint split $(S, f)$, defining the family of disjoint 2-intervals $\mathbf{D}$ as before, we obtain a disjoint unit 2-interval representation of $G$. □

Note that we could also ask the map $f$ of a (disjoint) split of a graph $G$ to satisfy $|f^{-1}(v)| = 2$ for every vertex $v$, as given a split $(S, f)$ such that $S$ is a unit interval graph, if $|f^{-1}(v)| = 1$, the split $(S', f)$ defined by adding an isolated representative of $v$ to the graph $S$ also belongs to $\mathcal{S}^*_{\mathcal{U}}(G)$ (resp., $\mathcal{S}_{\mathcal{U}}(G)$). However, we allow the possibility to discard isolated representatives for simplicity, as generally we will use this characterization to do case analysis on the possible splits, and reducing the number of total representatives reduces the search space.

**Vertex splitting to recognize (disjoint) unit $d$-interval graphs.** The previous characterizations can be generalized for higher dimensions, that is, for (disjoint) unit $d$-interval graphs with $d > 2$. To do so, it suffices to extend the notion of (disjoint) split of a graph as follows.

**Definition 4.5.** *Given a graph $G$ and a natural number $d > 0$, we say that a pair $(S, f)$ formed by a graph $S$ and a function $f : V(S) \mapsto V(G)$ is a* $d$-split *of $G$ if $f$ satisfies the following conditions:*

- $1 \leqslant |f^{-1}(v)| \leqslant d$ *for every* $v \in V(G)$.
- *For every edge $(s, t)$ of $S$, $(f(s), f(t))$ is an edge of $G$.*
- *For every edge $(u, v)$ of $G$, there exist two vertices $s$ and $t$ in $f^{-1}(\{u, v\})$ such that $(s, t)$ is an edge of $S$.*

*If furthermore, for every vertex $v$ of $G$, $f^{-1}(v)$ is an independent set in $S$, we say that $(S, f)$ is a* disjoint $d$-split.

Then, a graph $G$ is a (disjoint) unit $d$-interval graph if and only if there exists a (disjoint) $d$-split $(S, f)$ of $G$ such that $S$ is a unit interval graph. If we define the family of (disjoint) $d$-splits verifying that property as $\mathcal{S}^{d^*}_{\mathcal{U}}(G)$ (resp., $\mathcal{S}^{d}_{\mathcal{U}}(G)$), we can restate the characterization as follows.

**Lemma 4.2.** *A graph $G$ is a (disjoint) unit $d$-interval graph if and only if the family $\mathcal{S}^{d^*}_{\mathcal{U}}(G)$ (resp., $\mathcal{S}^{d}_{\mathcal{U}}(G)$) is not empty.*

As a remark, the reader can observe that we can also apply a similar characterization to verify whether a graph is a $d$-interval graph or not, by considering the family

$$\mathcal{S}^{d}_{\mathcal{I}}(G) := \{(S, f) \mid (S, f) \text{ is a } d\text{-split of } G \text{ and } S \text{ is an interval graph}\}$$

instead of the family $\mathcal{S}^*_{\mathcal{U}}(G)$.

**Brute-force recognition algorithm.** We conclude this section by pointing out that the previous characterizations can be used to obtain a brute-force recognition algorithm for (disjoint) unit $d$-interval graphs. Indeed, the brute-force algorithms based on the previous characterizations consist on considering all the possible $d$-splits or disjoint $d$-splits $(S, f)$ of the input graph and checking in linear time whether $S$ is a unit interval graph or not. We show that these algorithms are considerably more efficient that exploring all the possible (disjoint) unit $d$-interval representations.

**Theorem 4.1.** *DISJOINT UNIT $d$-INTERVAL RECOGNITION can be solved in time $O(2^{d^2m})$.*

*Proof.* We consider the brute-force algorithm that tests whether a given graph is a disjoint unit $d$-interval graph by considering all the possible disjoint $d$-splits of $G = (V, E)$. Let $(S, f)$ be a disjoint $d$-split of $G$. Let $S$ have vertex set $V' := \{u_1, \ldots, u_d | u \in V\}$. For every edge $(u, v) \in E$, $E(S)$ will contain at least one edge $(u_i, v_j)$, with $i, j \in \{1, \ldots, d\}$. There are $d^2$ such possibilities. However, these edge representatives are not mutually exclusive. Thus, in the worst case, we can have up to $2^{d^2}$ different sets of representatives for each edge of $G$. Since we have to guess the representatives for every edge in the original graph, this yields $2^{(d^2m)}$ possible disjoint $d$-splits. It then suffices to check in linear time whether each of the graphs is a unit interval graph, and return yes if at least one of the graphs is unit interval, and no otherwise. This yields an algorithm running in time $O(2^{d^2m})$. □

**Theorem 4.2.** *UNIT $d$-INTERVAL RECOGNITION can be solved in time $O(2^{d^2(m+n)})$.*

*Proof.* We can design a similar brute-force algorithm for UNIT $d$-INTERVAL RECOGNITION by considering all the possible $d$-splits of the input graph $G$. Again, for every edge, we have $2^{d^2}$ possible sets of representatives, but now we also need to consider all the possible edges between representatives of a same vertex. Since there are $d$ representatives, we have $d(d-1)/2$ possible edges for every vertex of $G$. This implies that the number of different $d$-splits of $G$ is upper-bounded by $2^{d^2(m+n)}$, and so the brute-force algorithm runs in time $O(2^{d^2(m+n)})$. □

## 4.3. Encoding the recognition problem in ASP

In this section, we exploit the semiorder characterization of unit interval graphs given in [131] to provide an efficient encoding of the problem of recognizing disjoint unit $d$-interval graphs. Recall that Roberts proved that an interval graph $G = (V, E)$ is unit if and only if there exists a semiorder $(V, P)$ on the vertex set $V$ such that $(u, v) \in P$ if and only if $(u, v) \notin E$. Building upon the previous characterization in terms of the disjoint $d$-splits of a graph, given a graph $G = (V, E)$, we want to check whether there exists a semiorder $(S, P)$ on the set $S := \{v_i | v \in V, i \in \{1, \ldots, d\}\}$ that satisfies the following properties:

- For every edge $(u, v) \in E$, the pairs $(u_i, v_j)$ do not belong to $P$ for any $i, j \in \{1, \ldots, d\}$.

- For every non-edge $(u, v) \in E$, at least one pair $(u_i, v_j)$ belongs to $P$ for some $i, j \in \{1, \ldots, d\}$.

We can use these constraints to guess a possible semiorder. That is, for every pair of vertices $u, v$ that is not connected by an edge, we guess whether $(u_i, v_j)$ belongs to $P$; $(v_i, u_j)$ belongs to $P$; or neither pair belongs to $P$, for every pair $u_i, v_j$ with $i, j \in \{1, \ldots, j\}$.

Here, we explain how to obtain an efficient solver to recognize disjoint unit 2-interval graphs by encoding this modelization of the problem in Answer Set Programming (ASP) language. Answer Set Programming is a form of declarative programming oriented towards difficult search problems, where problems are reduced to computing stable models (a set of statements that are true and don't contradict each other), and answer set solvers –programs for generating stable models– are used to perform the search [112].

Before explaining how to encode our modelization, let us give an overview of the different expressions that we can have in the ASP language `LPARSE` (front-end of the answer set solver `SMODELS`).

- Rules: a rule of the form `head:- body` means that `body` implies `head`, or equivalently if `body` is in the stable model then so is `head`. If the `body` is empty, the rule is called a fact.
- Constraints: they are rules without a head, written as `:- body`. They serve to eliminate stable models, for example, `:- s, not t` prohibits generating `s` if `t` is not generated (all the conditions in the body cannot happen at the same time).
- Choice rules, which can include numerical bounds. For example, `1 {s,t}:- p` means that if `p` is generated then at least one of `s` or `t` is generated.
- Variables: denoted by an uppercase letter, they are used to represent unknown values or to express general rules. For example, `p(X) :- q(X), r(X)`, is a rule that states that `p(X)` is true if both `q(X)` and `r(X)` are true, for any value of `X`.
- Literals: an atom or a negated atom, such as `not p(X)`.

**Encoding the semiorder constraints.** We can encode the constraints to check whether a relation $P$ on a set $S$ is a semiorder with the following constraints (where `semiorder(X,Y)` denotes that the pair $(X, Y)$ of elements of $S$ belongs to $P$):

- Transitivity: `:- semiorder(X,Y), semiorder(Y,Z), not semiorder(X,Z).`
- Irreflexivity: `:- semiorder(X,Y), semiorder(Y,X).`
- $(x, y) \in P$ and $(z, w) \in P$ imply $(x, w) \in P$ or $(z, y) \in P$:
  `:- semiorder(X,Y), semiorder(Z,W), not semiorder(X,W), notsemiorder(Z,Y).`
- $(x, y) \in P$ and $(y, z) \in P$ imply $(x, w) \in P$ or $(w, z) \in P$ for every $w$:
  `:- semiorder(X,Y), semiorder(Y,Z), element(W), not semiorder(X,W), not semiorder(W,Z).`

**Encoding a unit interval checker.** Recall that Roberts proved that an interval graph $G = (V, E)$ is unit if and only if there exists a semiorder $P$ on the set of vertices $V$ such that $(u, v) \in P$ if and only if $(u, v) \notin E$. To check whether a given graph is unit interval, we will see whether such a semiorder $P$ exists. First of all, we need to encode that for every pair of distinct elements $X$ and $Y$ (wich are vertices of the graph), the pairs $(X, Y)$ or $(Y, X)$ can be in $P$. We can do so with the following choice rule (where `element(X)` means that $X$ is an element of $V$, i.e., a vertex):

- `{ semiorder(X,Y) } :- element(X), element(Y), X != Y.`

Then, we add the constraint that ensures that if there is an edge between two elements $X$ and $Y$, then $(X, Y)$ is not in $P$ (where `theedge(X,Y)` means that there is an edge between $X$ and $Y$ in $G$):

- `:- element(X), element(Y), X!=Y, semiorder(X,Y), theedge(X,Y).`

Note that, as formulated here, to capture an undirected edge $(X, Y)$, we need to define `theedge(X,Y)` and `theedge(Y,X)`, but we will deal with this issue when we encode the graph from the input.

Finally, we add the constraint that if there is no edge between $X$ and $Y$, then at least one of $(X, Y)$ or $(Y, X)$ is in $P$:

- ```
  :- element(X), element(Y), X != Y, not semiorder(X,Y),
  not semiorder(Y,X), not theedge(X,Y).
  ```

The previous rules encode how to test whether a graph is a unit interval graph.

**Encoding the splits of a graph.** It only remains to encode the vertex splitting operation on the input graph $G$. We present the constraints required to verify that a graph is disjoint unit 2-interval.

For every vertex $V$ (`vertex(V)`), we need to generate two new elements of $S$, which we denote by `element(x(V))` and `element(y(V))`, and impose the constraint that every edge has a representative (the representative of an edge in the disjoint split are denoted by `theedge()`).

- `element(x(V)) :- vertex(V).`
- `element(y(V)) :- vertex(V).`
- `1 { theedge(x(U), x(V)); theedge(x(U), y(V)); theedge(y(U), x(V)); theedge(y(U), y(V))} :- edge(U,V), U < V.`
- `1 { theedge(x(U), x(V)); theedge(x(U), y(V)); theedge(y(U), x(V)); theedge(y(U), y(V))} :- edge(V,U), U < V.`

Finally, we impose that the two intervals associated to a same vertex are disjoint, and for every edge in the resulting graph, we generate both `theedge(V,U)` and `theedge(U,V)`, as we anticipated before.
  ```

- `semiorder(x(V), y(V)) :- vertex(V).`
- `theedge(V,U) :- theedge(U,V).`

To provide an instance for this solver, vertices must be indicated as `vertex(V)` and edges as `edge(U,V)` (only one pair per edge is needed). The implementation of this solver can be found in the following git repository: `https://github.com/AbdallahS/unit-graphs`. The reader can observe that this program can be adapted to test whether a graph belongs to the class of (disjoint) unit $d$-interval graphs or unit $d$-track interval graphs, for any $d \geqslant 2$.

The brute-force algorithm proposed in this chapter to recognize (disjoint) unit $d$-interval graphs and the characterization in terms of splits will be essential tools throughout the reminder of the manuscript. Furthermore, we also make use of the implementation in ASP to verify some results computationally and obtain counterexamples by computer search.

# 5. Generalizing Roberts' characterization of unit interval graphs

In this chapter, we study the generalization of Roberts characterization of unit interval graphs to higher dimensions. Recall that Roberts characterization states that the classes of proper and unit interval graphs coincide, and are exactly $K_{1,3}$-free interval graphs. Furthermore, there exist short constructive proofs of this result [21, 69]. This is a remarkable result as it gives a simple characterization of unit interval graphs.

As explained in Chapter 4, the necessary condition of being $K_{1,3}$-free extends naturally to multiple interval graphs: a (disjoint) unit 2-interval graph cannot contain a $K_{1,5}$ as an induced subgraph; and more generally, a (disjoint) unit $d$-interval graph cannot contain a $K_{1,2d+1}$ as an induced subgraph. Thus the following natural question arises: can we generalize Roberts characterization of unit interval graphs to multiple interval graphs? Perhaps the most straightforward generalization would be to characterize unit $d$-interval graphs as $K_{1,2d+1}$-free $d$-interval graphs, but this has already been proven false in [141]: there exists a graph which is 2-interval and $K_{1,5}$-free, but not unit 2-interval. But not all hope of generalizing Roberts characterization must be lost yet! What if we add some additional constraints?

Already in 2016, Durán et al. decided to focus on $d$-interval graphs which are also *interval* [55]. In a presentation at VII LAWCG, they claimed that if $G$ is an interval graph, then $G$ is a disjoint unit $d$-interval graph if and only if it is $K_{1,2d+1}$-free [1]. In this chapter, we show that the aforementioned statement is actually false, and that, perhaps surprisingly, Roberts characterization can only be generalized depending on the chosen definition of $d$-interval graphs! In particular, this implies that the classes of disjoint unit $d$-interval and unit $d$-interval graphs are not equivalent (see Figure 5.1 for a summary of the main results).

The outline of the chapter is the following. In Section 5.1, we generalize Roberts characterization for unit $d$-interval graphs. In Sections 5.2 and 5.3 we highlight the problems encountered when trying to generalize it for disjoint unit $d$-interval graphs and prove that it cannot be done. Then, in Section 5.4, we give an approximation algorithm with an additive error of one to recognize disjoint unit $d$-interval graphs which are also interval. Finally, in Section 5.5, we study the subclasses of graphs obtained under the two definitions of $d$-intervals in the balanced case, expanding the knowledge of the containment relations between the different subclasses of 2-interval graphs.

[1] Note that they refer to disjoint unit $d$-intervals simply as unit $d$-intervals, but they are explicitly defined beforehand as the union of $d$ disjoint intervals.

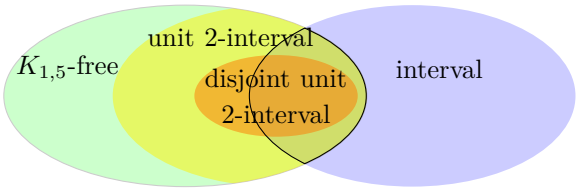



Figure 5.1.: $K_{1,5}$-free interval graphs are not contained in the class of disjoint unit 2-interval graphs. The class of unit 2-interval graphs is a superclass of disjoint unit 2-interval graphs, and spans the whole intersection of $K_{1,5}$-free and interval graphs.

## 5.1. Generalization for unit $d$-interval graphs

In this section, we generalize Roberts characterization of unit interval graphs for $d$-interval graphs. Recall that by $d$-interval graphs we refer to intersection graphs of $d$-intervals where the $d$ intervals are not necessarily disjoint.

**Theorem 5.1.** *Let $G$ be an interval graph. Then, for any natural number $d \geqslant 2$, $G$ is a unit $d$-interval graph if and only if $G$ does not contain a copy of a $K_{1,2d+1}$ as an induced subgraph. Furthermore, given a $K_{1,2d+1}$-free interval graph, a unit $d$-interval representation can be constructed in $\mathcal{O}(n+m)$ time, where $n$ and $m$ are the number of vertices and edges of the graph, respectively.*

We present a polynomial-time algorithm that, given an arbitrary interval representation $\mathcal{I}$ of a $K_{1,2d+1}$-free graph, returns a $d$-interval representation $\mathcal{I}'$ of the graph where no interval of the underlying family of $\mathcal{I}'$ intersects three or more pairwise disjoint intervals. This ensures that the underlying family of intervals returned corresponds to an interval representation of a $K_{1,3}$-free graph, so we can use the algorithm described in [21] to turn it into a proper representation (and then to a unit one in linear time [69]). Note that if an interval representation of the graph is not given, we can always compute it in linear time [45].

Before presenting the algorithm formally, let us give the idea behind it. The algorithm constructs a family $\mathcal{I}'$ of $d$-intervals in the following way: for every interval $I \in \mathcal{I}$ that intersects $m$ (and no more than $m$) pairwise disjoint intervals, we create a $t$-interval $I_1 \cup \ldots \cup I_t$, where $t = \lceil \frac{m}{2} \rceil$. Note that for every interval $I$ that intersect only two disjoint intervals, we have $t = 1$, and the interval $I_1$ added to $\mathcal{I}'$ will be exactly $I$. We will refer to such intervals as *original intervals*, as they are equal to the ones in $\mathcal{I}$. After creating the $t$-intervals described above, to obtain a $d$-interval representation of the graph, it suffices to add $d - t$ "dummy" intervals for each vertex that is represented by $t < d$ intervals (where by "dummy" intervals we mean that they do not intersect any other interval from the representation). Each $d$-interval $I_1 \cup \ldots \cup I_d$ introduced will preserve the same intersections as the interval $I \in \mathcal{I}$, and each $I_i$ will possess three key properties: it intersects at most two disjoint original intervals, it contains an original interval, and each of its endpoints coincides with an endpoint of an original interval. These properties ensure that the representation $\mathcal{I}'$ can be made unit.

**Algorithm.** Let the family of intervals $\mathcal{I}$ be an interval representation of $G$. For every interval $I \in \mathcal{I}$, let $l(I)$ and $r(I)$ stand for its left and right endpoint, respectively. Furthermore, define a partial order as follows: given two intervals $I, J \in \mathcal{I}$, let $I \prec J$ if and only if $r(I) < l(J)$ (i.e. interval $J$ is fully to the right of interval $I$). Two intervals are incomparable if they intersect.

**Step 1** Initialize a set of intervals $\mathcal{C}$ with all the intervals of $\mathcal{I}$, set $\mathcal{I}' := \emptyset$, and go to **Step 2**.

**Step 2** Pick an interval $I$ of $\mathcal{C}$, remove it from the set and define its neighborhood $\mathcal{N}(I) = \{J \in \mathcal{I} \colon J \cap I \neq \emptyset\}$. Let $m$ be the maximum number of pairwise disjoint intervals that $I$ intersects. If $m \leqslant 2$, go to **Step 3**; if $m = 3$, go to **Step 4**; and if $m > 3$, go to **Step 5**.

**Step 3** If $m \leqslant 2$, add the interval $I_1 = I$ to the family $\mathcal{I}'$ and call $I_1$ an original interval. Then go to **Step 6**.

**Step 4** If $m = 3$, define four auxiliary intervals:

$$A_1 = \underset{J \in \mathcal{N}(I)}{\arg\min}\{r(J)\} \qquad A_2 = \underset{\{J \in \mathcal{N}(I) \colon A_1 \prec J\}}{\arg\min}\{r(J)\}$$
$$A_4 = \underset{J \in \mathcal{N}(I)}{\arg\max}\{l(J)\} \qquad A_3 = \underset{\{J \in \mathcal{N}(I) \colon J \prec A_4\}}{\arg\max}\{l(J)\}$$

Then add to $\mathcal{I}'$ the 2-interval $I_1 \cup I_2$, with $I_1 = [l(I), r(A_2)]$ and $I_2 = [l(A_3), r(I)]$. Note that $A_2$ and $A_3$ necessarily intersect, as otherwise we would have $m \geqslant 4$, so $I_1 \cup I_2$ is not a disjoint 2-interval. After adding it to $\mathcal{I}'$, go to **Step 6**.

**Step 5** If $m > 3$, define two families of auxiliary intervals. The first family $\mathcal{A} := \{A_i \,|\, i \in \{1, \ldots, m\}\}$ forms a maximum set of pairwise disjoint intervals intersecting $I$, and it will ensure that all the intersections are preserved. It is defined as follows:

$$A_1 = \underset{J \in \mathcal{N}(I)}{\arg\min}\{r(J)\} \qquad A_i = \underset{\{J \in \mathcal{N}(I) \colon A_{i-1} \prec J\}}{\arg\min}\{r(J)\} \;,\; \forall\, i \in \{2, \ldots, m-2\}$$
$$A_m = \underset{J \in \mathcal{N}(I)}{\arg\max}\{l(J)\} \quad A_{m-1} = \underset{\{J \in \mathcal{N}(I) \colon J \prec A_m\}}{\arg\max}\{l(J)\}$$

The second family $\mathcal{B} := \{B_i \,|\, i \in \{1, \ldots, m\}\}$ is a tool to ensure that each new interval $I_i$ intersects only two disjoint intervals in $\mathcal{I}'$. Note that restricting each $I_i$ to intersect only two disjoint intervals from the family $\mathcal{A}$ is not enough: for example, in Figure 5.2, if $I_2$ were defined as $[l(A_3), r(A_4)]$, then it would intersect three pairwise disjoint intervals in $\mathcal{I}'$ (as all the intervals except $I$ are original intervals in this example), whereas if the left endpoint of $I_2$ were chosen as $r(A_{2i-1})$, then an original interval that was not the center of a claw in $\mathcal{I}$ might become the center of a new claw in $\mathcal{I}'$. Thus, for every $i \in \{1, \ldots, m\}$, $B_i$ is defined as follows:

$$B_i = \underset{J \in \mathcal{N}(A_i) \cup A_i}{\arg\max}\{l(J)\}$$

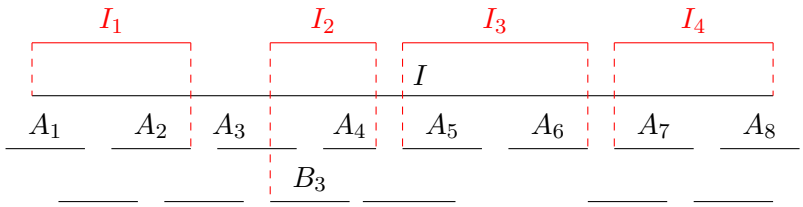



Figure 5.2.: Interval $I$ intersects 8 disjoint intervals. In red, the 4-interval returned by the algorithm. Note that if $l(I_2)$ were defined as $l(A_3)$ instead of $l(B_3)$, it would intersect three pairwise disjoint original intervals.

In other words, $B_i$ is the interval in the closed neighborhood of $A_i$ starting the latest. Note that if there does not exist any interval intersecting $A_i$ which starts after $A_i$, then $B_i = A_i$ since we are considering the closed neighborhood. Now, add to $\mathcal{I}'$ the $t$-interval $I_1 \cup ... \cup I_t$, defined as follows. We distinguish two slightly different cases:

a) If $m$ is even, i.e., $m = 2t$ for some $t > 1$, define $I_1 = [l(I), r(A_2)]$, $I_i = [l(B_{2i-1}), r(A_{2i})]$ for every $i \in \{2, \ldots, t-1\}$, and $I_t = [l(A_{2t-1}, r(I)]$.

b) If $m$ is odd, i.e., $m = 2t - 1$ for $t > 2$, define $I_{t-1}$ and $I_t$ differently, as $I_{t-1} = A_{2t-3}$ and $I_t = [l(A_{2t-2}, r(I)]$, and the rest of the intervals as before.

Notice that by definition, the intervals $I_i, ..., I_t$ are actually pairwise disjoint, so if $m > 3$, the $t$-interval added to $\mathcal{I}'$ is a disjoint $t$-interval. After adding the $t$-interval, go to **Step 6**.

**Step 6** If $\mathcal{C} = \emptyset$, return $\mathcal{I}'$, else go to **Step 2**.

Figure 5.2 illustrates the algorithm on a concrete interval which intersects eight pairwise disjoint intervals.

Before proceeding to the proof of correctness of the algorithm, we highlight the properties of the intervals constructed that will be useful to prove the next three lemmas. In the following, we say that an interval $I$ of $\mathcal{I}$ has been *transformed* into a $t$-interval $I_1 \cup \cdots \cup I_t$ by the algorithm after it has been processed.

**Observation 5.1.** *Let $I \in \mathcal{I}$ be an interval transformed into $I_1 \cup \cdots \cup I_t$ by the algorithm, for some $1 < t \leqslant d$. Then, for every $i \in \{1, \ldots, t\}$:*

1. *The left (resp., right) endpoint of every interval $I_i$ coincides with the left (resp., right) endpoint of an original interval.*

2. *There is an original interval contained in $I_i$.*

Now, to prove the correctness of the algorithm, we need to show that for every interval $I \in \mathcal{I}$, the $t$-interval $I_1 \cup ... \cup I_t \in \mathcal{I}'$ preserves the same intersections as $I$, and that no interval in the underlying family of $\mathcal{I}'$ intersects three pairwise disjoint intervals. In the next claim, we prove that intersections are preserved:

**Lemma 5.1.** *Let $I$ be an interval transformed into $I_1 \cup \cdots \cup I_t$ by the algorithm, for some $1 \leqslant t \leqslant d$. Then, the $t$-interval $I_1 \cup \ldots \cup I_t$ preserves the intersections of $I$.*

*Proof.* It is clear that no new intersections are created as $I_1 \cup \ldots \cup I_t \subseteq I$. To see that no intersection is lost, suppose that there exists an interval $L$ that intersects $I$ in the original representation $\mathcal{I}$, and after the algorithm finishes, $L$ is transformed into a $t_0$-interval $L_1 \cup \ldots \cup L_{t_0}$ (for some $1 \leqslant t_0 \leqslant d$, where if $t_0 = 1$, the interval remains as in the original representation) such that the $t$-interval $I_1 \cup \ldots \cup I_t$ does not intersect the $t_0$-interval $L_1 \cup \ldots \cup L_{t_0}$ in $\mathcal{I}'$. Since $l(I_1) = l(I)$ and $r(I_t) = r(I)$ (and the same holds for $L$), this means that there exists an $L_j$ (with $1 \leqslant j \leqslant t_0$) such that $I_i \prec L_j \prec I_{i+1}$ for some $1 \leqslant i \leqslant t-1$.

For $1 \leqslant t \leqslant 2$, this cannot occur because $I \subseteq I_1 \cup \ldots \cup I_t$. For $t > 3$, since the set of intervals $A_k$ used to defined the $t$-interval associated to $I$ forms a maximal set of pairwise disjoint intervals intersecting $I$, we cannot have that $A_k \prec L_j \prec A_{k+1}$ for any $1 \leqslant k \leqslant 2t-1$. Indeed, this would contradict maximality, as $L_j$ is either an original interval or it contains an original interval (by Observation 5.1). Thus, the only possible option is that there exists an $i$ such that $A_{2i} \prec L_j \prec B_{2i+1}$ (where $B_{2i+1}$ is different from $A_{2i+1}$). Then, since $B_{2i+1}$ intersects $A_{2i+1}$ and $L_j \prec B_{2i+1}$, we have that $r(L_j) < r(A_{2i+1})$. But this contradicts the choice of $A_{2i+1}$, which should have been $L_j$ or the original interval contained in $L_j$, as $A_{2i} \prec L_j$. □

The next two lemmas are dedicated to proving that no interval in the underlying family of $\mathcal{I}'$ intersects three or more pairwise disjoint intervals. We distinguish the cases when the center of the claw is an original interval and when it is not.

**Lemma 5.2.** *Let $I \in \mathcal{I}$ be an original interval (i.e., transformed to $I_1$ by the algorithm). Then, $I_1$ intersects at most two disjoint intervals in the underlying family of $\mathcal{I}'$.*

*Proof.* Suppose, towards a contradiction, that there exists an original interval $I_1$ that intersects three pairwise disjoint intervals $L_1, L_2$ and $L_3$ in the underlying family of $\mathcal{I}'$, with $L_1 \prec L_2 \prec L_3$. By Observation 5.1, there exists an original interval $L'_1$ with the same right endpoint as $L_1$, an original interval $L'_2$ contained in $L_2$, and an original interval $L'_3$ with the same left endpoint as $L_3$. Note that if any of the $L_i$ are original, then $L'_i = L_i$. But then, $L'_1 \prec L'_2 \prec L'_3$ are three pairwise disjoint original intervals that intersect $I_1$, which contradicts the fact that it is an original interval. Indeed, this implies that the interval $I$ intersects three pairwise disjoint intervals in $\mathcal{I}$, and so the algorithm would have transformed it into a $t$-interval with $t$ strictly greater than 1. □

**Lemma 5.3.** *Let $I \in \mathcal{I}$ be an interval transformed into the $t$-interval $I_1 \cup \cdots \cup I_t$ by the algorithm, for some $1 < t \leqslant d$. For every $1 \leqslant i \leqslant t$, $I_i$ intersects at most two disjoint intervals of the underlying family of $\mathcal{I}'$.*

*Proof.* We proceed by contradiction. Suppose that there exists an interval $I_i$, with $1 \leqslant i \leqslant t$ that intersects three pairwise disjoint intervals $L_1, L_2, L_3$, with $L_1 \prec L_2 \prec L_3$. By Observation 5.1, there exists an original interval $L'_1$ with the same right endpoint as

$L_1$, an original interval $L'_2$ contained in $L_2$, and an original interval $L'_3$ with the same left endpoint as $L_3$.

Assume first that $t = 2$. Then, if $i = 1$, this contradicts the choice of the interval $A_2$ (resp. $A_3$ if $i = 2$), which should have been $L'_2$.

Let us now study the general case for $t > 2$. Suppose first that $1 < t < d$ and $I_i$ is defined as $[B_{2i-1}, A_{2i}]$ with $B_{2i-1} \neq A_{2i-1}$. We distinguish two cases:

1. $r(L_1) > r(A_{2i-1})$. Then, since we are assuming that $L_1$ and $L_2$ are disjoint, $l(L_2) > r(A_{2i-1})$. Furthermore, as $L_3$ also intersects $I_i$, we need $r(L_2) < r(A_{2i})$. But this contradicts the choice of $A_{2i}$, which should have been $L'_2$.

2. $r(L_1) < r(A_{2i-1})$. If $l(L_2) > r(A_{2i-1})$, we are in the same case as before. Thus, $L_2$ and $A_{2i-1}$ must intersect. However, we have $l(L_2) > l(B_{2i-1})$ (since otherwise $I_i$ would not be able to intersect $L_1$ on its left extreme). This contradicts the choice of $B_{2i-1}$ if $L'_2$ intersects $A_{2i-1}$, or the choice of $A_{2i}$ otherwise.

On the other hand, if $B_{2i-1} = A_{2i-1}$, then by construction, since we take the two disjoint intervals that finish first, we cannot have three pairwise disjoint intervals intersecting $I_i$. This is also the case for $I_1$ and $I_t$ (although in the latter case, we take the two disjoint intervals starting last). Finally, for odd claws, it is also clear that $I_{t-1}$ intersects at most two disjoint intervals, as it is equal to an original interval. □

Combining Lemmas 5.1, 5.2 and 5.3, plus the fact that we can trivially transform a $t$-interval with $t < d$ into a $d$-interval, we obtain that the algorithm returns a $d$-interval representation of the input graph where no interval of the underlying family intersects more than two disjoint intervals, which as explained before can be converted into a unit representation. The last part of Theorem 5.1 follows because an efficient implementation of the algorithm described above requires $\mathcal{O}(1 + deg(v))$ operations for each vertex $v$ (where $deg(v)$ denotes the degree of vertex $v$), as it suffices to iterate over the neighborhood of a given interval to transform it into the corresponding $d$-interval. Finally, the obtained representation can be converted to a unit representation in linear time, which yields the stated runtime $\mathcal{O}(n+m)$. This concludes the proof of Theorem 5.1.

We have proven that the algorithm constructs a unit $d$-interval representation, but it is not a disjoint one. Indeed, as mentioned before, in the case of maximal $K_{1,3}$'s, the constructed intervals $I_1$ and $I_2$ intersect each other. However, in the case of maximal $K_{1,m}$'s with $m > 3$, the $t$ intervals of the $t$-interval created are actually pairwise disjoint. Thus, we can infer the following direct corollary.

**Corollary 5.1.** *Let $G$ be a $K_{1,2d+1}$-free interval graph that does not contain any maximal $K_{1,3}$. Then $G$ is a disjoint unit $d$-interval graph.*

*Proof.* If the interval graph given as input of the algorithm does not contain any maximal $K_{1,3}$, by construction, all the $d$-intervals returned are formed by $d$ disjoint intervals. Thus, we obtain a unit $d$-interval representation which is also a disjoint unit $d$-interval representation. □

However, with a more careful analysis, we can infer an even stronger corollary, which instead of requiring the absence of maximal 3-claws altogether, only forbids a subset of them. We refer to these forbidden claws, which are exactly those maximal 3-claws contained in an induced $E$ graph, as $E$-claws. Recall that an $E$ graph (or $\text{star}_{1,2,2}$) is a graph on six vertices which has as edge set a path $v_1, v_2, v_3, v_4, v_5$ and an additional edge $(v_3, v_6)$.

**Definition 5.1.** *A maximal 3-claw is an $E$-claw if it is contained in an induced $E$ graph.*

**Theorem 5.2.** *Let $G$ be a $K_{1,2d+1}$-free graph that does not contain any $E$-claws. Then, $G$ is a disjoint unit $d$-interval graph.*

*Proof.* To prove the theorem, we modify **Step 4** of the previous algorithm so that it produces a disjoint 2-interval.

**Step 4'** Let $I$ be an interval and let $m = 3$ be the maximum number of pairwise disjoint intervals that it intersects. By assumption, the vertex associated to $I$ is a center of a maximal claw which is not an $E$-claw. We define

$$A_1 = \arg\min_{J \in \mathcal{N}(I)} \{r(J)\}$$
$$A_2 = \arg\min_{\{J \in \mathcal{N}(I) \colon A_1 \prec J\}} \{r(J)\}$$
$$A_4 = \arg\max_{J \in \mathcal{N}(I)} \{l(J)\}$$
$$A_3 = \arg\max_{\{J \in \mathcal{N}(I) \colon J \prec A_4\}} \{l(J)\}$$

Note that $A_2$ and $A_3$ necessarily intersect (or are the same interval). Now, since the vertex associated to $I$ is a center of a claw that is not an $E$-claw, this means that at least one of $A_1$ or $A_4$ does not intersect an interval which is disjoint from $I$. Thus, we can modify the representation so that $A_1$ (resp. $A_4$) is properly contained in $I$ without loosing any intersections, by simply stretching them. Then, if $A_1$ is properly contained in $I$, we define $I_1 = A_1$ and $I_2 = [l(A_3), r(I)]$. On the other hand, if $A_4$ is properly contained in $I$ instead, we define $I_1 = [l(I), r(A_2)]$ and $I_2 = A_4$. If both of them are properly contained in $I$, we can define $I_1$ and $I_2$ either way. After adding $I_1$ and $I_2$ to $\mathcal{I}'$, go to **Step 6**.

Notice that the 2-intervals introduced in this step have the same properties as in Observation 5.1, so the proof of correctness of the previous algorithm can be directly adapted for this extension. □

## 5.2. Limits of the algorithm

In this section, we explore the limits of the algorithm. We have just seen that if an interval graph $G$ which is $K_{1,2d+1}$-free does not contain any induced $E$-claws, we can extend

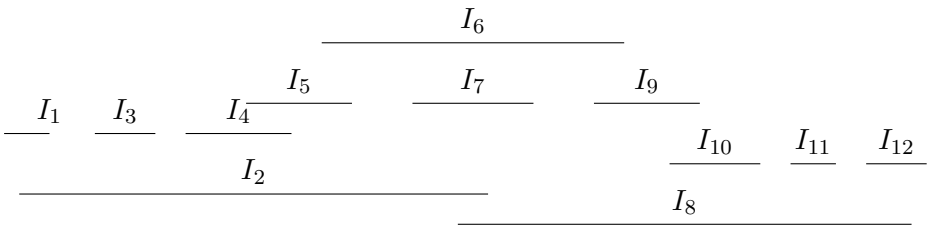



Figure 5.3.: Interval representation of the graph $G$ for which the algorithm does not return a disjoint unit 2-interval representation.

the algorithm in Section 5.1 so that it returns a disjoint unit $d$-interval representation of $G$. But what happens when the graph contains an $E$-claw? A natural question to pose is whether the extension of the algorithm returns a solution for every graph that *is* a disjoint unit $d$-interval graph.

In this section, we show that this is not the case. We provide a $K_{1,5}$-free interval graph that contains an $E$-claw and is a disjoint unit 2-interval graph (see Figure 5.3), for which the extension of the algorithm does not work. Furthermore, we observe with another example (see Figure 5.5) that it is not always straightforward to obtain a disjoint unit 2-interval representation even when one exists.

The algorithm and the extension of the algorithm for graphs without $E$-claws both process the intervals in the same manner: "cutting" them. That is, given an interval $I$, they find two points $x$ and $y$ in $I$ and define two intervals $[l(I), x]$ and $[y, r(I)]$ which comprise the 2-interval associated to $I$ in the output representation. We first show that there exists a $K_{1,5}$-free interval graph that contains an induced $E$-claw, and is disjoint unit 2-interval, for which the extension of the algorithm does not work and cannot be modified naturally, in the sense that no matter the points $x$ and $y$ that we choose in the interval $I$ associated to the center, no intervals $[l(I), x]$, $[y, r(I)]$ will lead to a disjoint unit representation. In other words, given an interval representation of the graph, we cannot "cut" the interval $I$ associated center of the claw in order to obtain a representation that can be transformed into a unit one.

Consider an interval graph $G$ which has the interval representation $\mathcal{M} = \{I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9, I_{10}, I_{11}, I_{12}\}$ displayed in Figure 5.3. The reader can observe that the graph is almost rigid, as the order of its maximal cliques is unique up to reversal and up to permuting the cliques that contain $I_1$ and $I_3$ and the cliques that contain $I_{11}$ and $I_{12}$ (the representation is unique if we ignore labels). In particular, this implies that no matter what interval representation of the graph we start with, the same situation will arise.

The graph does not contain any induced $K_{1,5}$'s, so it is a unit 2-interval graph. However, it contains an induced $E$-claw, so we cannot apply the previous algorithm to obtain a *disjoint* unit 2-interval representation. Suppose that we want to extend the algorithm in the most natural way, that is, we want to find points $x, y \in [l(I_6), r(I_6)]$ such that defining $I_6^1 = [l(I_6), x]$ and $I_6^2 = [y, r(I_6)]$ and processing every other interval as in the previous algorithm results in a disjoint unit 2-interval representation of the graph.

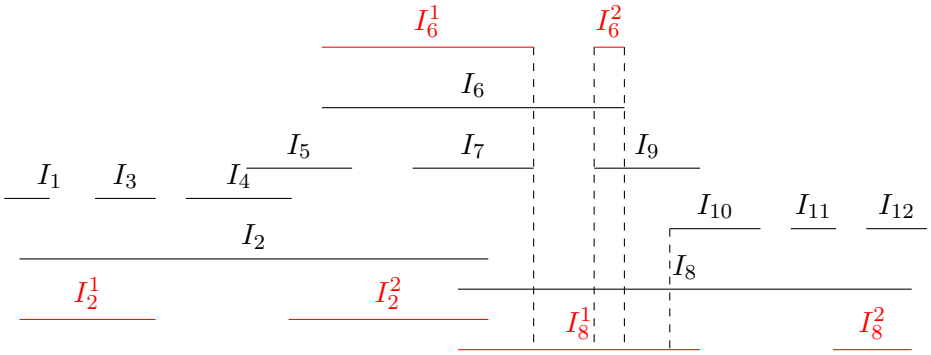



Figure 5.4.: Interval representation of the graph $G$ with the 2-intervals representing non-original intervals in red. No way of "cutting" $I_6$ leads to a disjoint unit 2-interval representation.

We prove here that points $x$ and $y$ do not exist. Towards a contradiction, suppose that they exist. Then, if $x < l(I_8)$, there would be three disjoint intervals intersecting $I_2^2$: $I_4, I_6^1$ and $I_8^1$. On the other hand, if $x \geqslant l(I_8)$, since $I_6^1$ and $I_6^2$ cannot intersect (as we aim to obtain a disjoint unit 2-interval representation), then there would be three pairwise disjoint intervals intersecting $I_8^1$: $I_6^1$, $I_6^2$ and $I_{10}$ (see Figure 5.4). Thus, we cannot obtain a disjoint unit 2-interval representation of $G$ in this manner. Note that in both cases, the problem arises because either $I_6^1$ or $I_6^2$ do not properly contain an original interval of the representation. This highlights the importance of this property of the $d$-intervals defined in Observation 5.1.

However, the graph $G$ *is* a disjoint unit 2-interval graph. One can simply define $I_6^1 = [l(I_6), r(I_7)]$, $I_9^1 = I_9$ and let $I_6^2$ and $I_9^2$ be two intersecting intervals that do not intersect any other interval in the representation. This yields a 2-interval representation of the graph where no interval intersects more than two disjoint interval, which can then be transformed into a unit one.

On the other hand, note that if we consider the graph $G'$ obtained from $G$ by suppressing edge $(I_2, I_8)$, then $G'$ still contains an $E$-claw, but now we can define $I_6^1 = [l(I_6), r(I_2)]$ and $I_6^2 = [l(I_8), r(I_6)]$, and the other 2-intervals as before, and this would yield a representation that can be transformed into a disjoint unit 2-interval one, that is, we can "cut" $I$ and obtain a representation with the desired properties.

The two previous examples demonstrate that the presence of $E$-claws by themselves does not necessarily pose a problem, as there exist graphs with $E$-claws that are disjoint unit 2-interval graphs and for which such a representation can be easily found. Thus, the next natural question is whether we can shed more light into the forbidden structures that cause difficulty in finding a disjoint unit representation, and whether we can always overcome this difficulty in a simple manner.

These questions remain open. However, we show with another example that it is not always as straightforward to find a disjoint unit 2-interval representation, even when one exists. We provide a $K_{1,5}$-free interval graph $G$ that contains an $E$-claw and has a

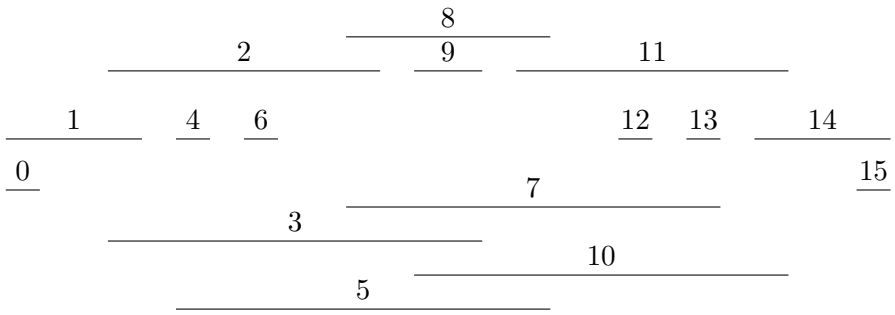



Figure 5.5.: Interval representation of a $K_{1,5}$-free interval graph for which the algorithm does not return a disjoint unit 2-interval representation. In this particular case, it is not straightforward to find such a representation, even though it exists.

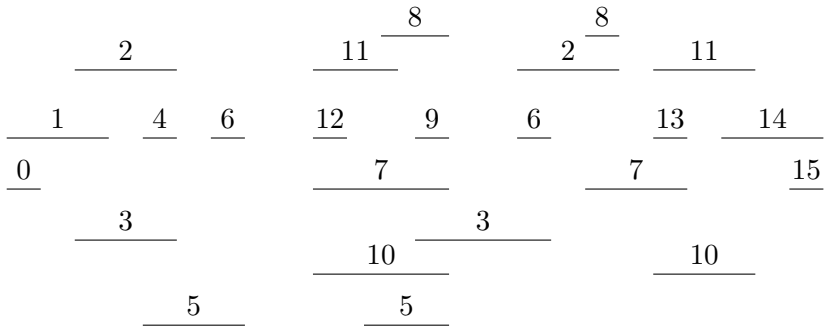



Figure 5.6.: Disjoint 2-interval representation of the graph in Figure 5.5. The fact that no interval intersects three pairwise disjoint intervals implies that the input graph is disjoint unit 2-interval.

disjoint unit 2-interval representation, but this representation can only be obtained by permuting the order of the intervals (as opposed to before, where we only "cut" intervals and place new copies in a separate part of the line). The interval representation of the graph (again, unique if we ignore labels) is depicted in Figure 5.5, and a disjoint 2-interval representation of the graph where no interval intersects three pairwise disjoint intervals is given in Figure 5.6. The reader can observe that in the original representation, the maximal cliques that contain vertex 7 must appear in the following order: $\{8, 1, 7, 3, 5\}, \{8, 9, 7, 3, 10, 5\}$, $\{8, 7, 11, 10, 5\}$ and, up to reversal, $\{11, 7, 12, 10\}$ and $\{7, 11, 10, 13\}$. However, in the disjoint unit 2-interval representation, the order of the cliques changes: the cliques containing vertices $\{1, 7\}$ and vertices $\{7, 11, 13\}$ are adjacent in the clique path, which is not possible in an interval representation. Although the representation shown here might not be unique, we conjecture that there does not exist a disjoint unit 2-interval representation where the cliques appear in the same order as the maximal cliques of the interval representation. This highlights new difficulties for determining whether a disjoint unit 2-interval representation exists.

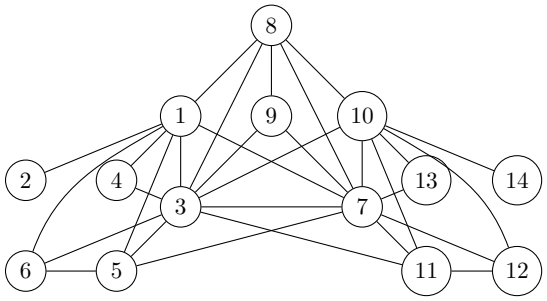



Figure 5.7.: One of the 6 graphs with 14 vertices (the one with the fewest edges) which is an interval graph (see Figure 5.8) and $K_{1,5}$-free, but not disjoint unit 2-interval.

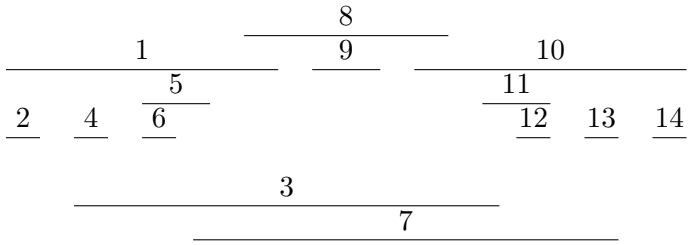



Figure 5.8.: An interval representation of the graph in Figure 5.7.

## 5.3. Counterexample for disjoint unit $d$-interval graphs

We have seen that there are some $K_{1,2d+1}$-free interval graphs that contain $E$-claws that are still disjoint unit $d$-interval graphs, even if the algorithm proposed in Section 5.1 does not return a unit representation. In this section, we prove that this is not always the case, that is, Theorem 5.1 cannot be generalized for disjoint unit $d$-interval graphs. In particular, we prove the following theorem.

**Theorem 5.3.** *There exists a $K_{1,5}$-free interval graph that is not a disjoint unit 2-interval graph.*

To prove Theorem 5.3, we offer the graph $G$ in Figure 5.7 as a certificate. The reader can check that $G$ has no induced $K_{1,5}$, and an interval representation of $G$ is provided in Figure 5.8. The proof that $G$ is not a disjoint unit 2-interval graph is the challenging part. Indeed, checking whether a graph is disjoint unit 2-interval is a computationally expensive task, as discussed in Section 4.1, and even with the aid of computer search, a naive ILP implementation already takes too much time to consider an exhaustive search. Needless to say, checking manually by brute force leads to a very long branching process.

The proof presented here uses the characterization of disjoint unit 2-interval graphs in Lemma 4.1 and is based on a careful analysis of the graph. We also verify the proof computationally, using the encoding in answer set programming based on the semiorder characterization of unit interval graphs proposed in Section 4.3, which proves to be way

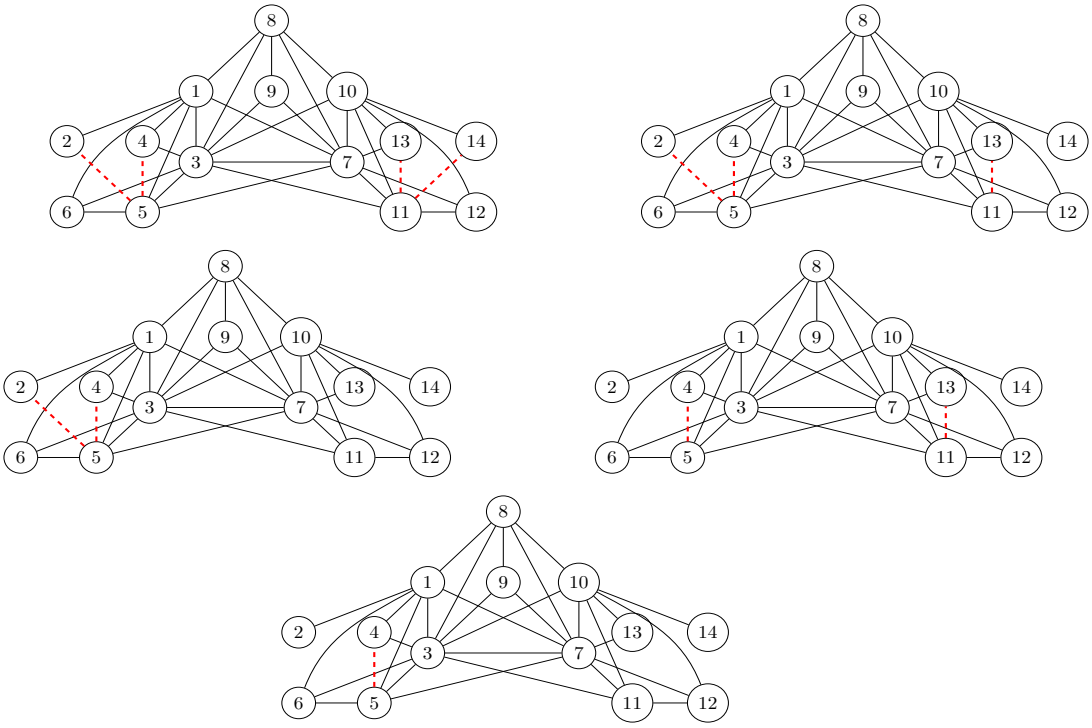



Figure 5.9.: The five other graphs on 14 vertices which are interval and $K_{1,5}$-free but not disjoint unit 2-interval. In dashed red, the edges that differ from the graph that we analyze here.

more efficient than an ILP encoding in practise (note that the theoretical analysis of the running time of both algorithms already anticipated this). Furthermore, we provide 5 other $K_{1,5}$-free interval graphs on the same number of vertices, and with a very similar structure, that are not disjoint unit 2-interval (see Figure 5.9). The proof that they are not disjoint unit 2-interval is omitted, but it is analogous to the one presented here. These six graphs are the only such graphs on 14 vertices, and there does not exist a graph satisfying the conditions of Theorem 5.3 with fewer vertices. These assertions were verified by computer search over all interval graphs of a given size without induced $K_{1,5}$'s [149]. Our code and experimental setting can be found on our git repository [2].

Theorem 5.3 follows directly from the next lemma.

**Lemma 5.4.** *The graph $G$ in Figure 5.7 is not a disjoint unit 2-interval graph.*

*Proof.* By Lemma 4.1, we know that a graph $G$ is a disjoint unit 2-interval graph if and only if the family of disjoint splits of $G$ that lead to a unit interval graph, $\mathcal{S}_{\mathcal{U}}(G)$, is not empty. Thus, to prove that $G$ is not a disjoint unit 2-interval graph, we need to show that there is no disjoint split of $G$ in $\mathcal{S}_{\mathcal{U}}(G)$, that is, there is no way to obtain a unit interval graph $G'$ from $G$ by splitting some or all of the vertices $v$ of $G$ into two

[2] `https://github.com/AbdallahS/unit-graphs`

non-adjacent vertices $v_1$ and $v_2$ so that $(u, v) \in E(G)$ if and only if $(u_i, v_j) \in E(G')$ for some $i, j \in \{1, 2\}$. Recall that $v_1$ and $v_2$ comprise the set $f^{-1}(v)$ and they are called the *representatives* of $v$, while the edges $(u_i, v_j)$ in $E(G')$ are called the *representatives* of edge $(u, v) \in E(G)$. Furthermore, given a disjoint split $(G', f)$ that belongs to $\mathcal{S}_{\mathcal{U}}(G)$, we will refer to the graph $G'$ as a *solution*. In particular, note that if a representative of a vertex that has been split in $G'$ is a center of a 3-claw and has as leaves vertices that are not adjacent to the other representative, then we cannot obtain a solution by splitting the rest of the vertices of $G'$. Likewise, if any vertex in $G'$ is a center of a 5-claw, we cannot obtain a solution from $G'$. We say that a solution is *canonical* if for every vertex $v$ that has been split into $v_1$ and $v_2$, none of the following hold:

- $v_i$ is isolated in $G'$ (if this happened, we could remove $v_i$ from $G'$ and end up with another solution with fewer split vertices).
- $v_i$ is adjacent to $u_1$ or $u_2$ for every $(u, v) \in E(G)$ (in this case, we could remove vertex $v_{3-i}$ from $G'$ and end up with another solution with fewer split vertices).

Note that if there exists a solution, then a canonical solution also exists. We now list some properties that a solution $G'$ must satisfy:

**Observation 5.2.** *The number of representatives of an edge must satisfy the following conditions:*

- *No edge $(u, v) \in E(G)$ can have 4 representatives in $G'$. Otherwise, the subgraph induced by $(u_1, v_1, u_2, v_2)$ would form a $C_4$ in $G'$, contradicting the fact that $G'$ is an interval graph.*
- *Every edge $(u, v) \in E(G)$ that is part of a 4-claw of $G$ must have precisely one representative in $G'$, since $G'$ cannot contain any 3-claws.*
- *Every edge $(u, v) \in E(G)$ that is part of a 3-claw of $G$ has at most two representatives.*

After these preliminary observations, we proceed to the detailed analysis of the graph $G$ presented in Figure 5.7 and we show that it does not have a canonical solution. We begin with three observations that follow directly from the structure of $G$.

**Observation 5.3.** *The following statements hold:*

- *No leaf vertex (vertex of degree 1) is split by a canonical solution. Hence, vertices 2 and 14 are not split in a canonical solution.*
- *The centers of an induced 3-claw in $G$ must be necessarily split in every solution. Therefore, if there exists a solution for $G$, then the vertices $1, 3, 7, 8$ and $10$ must all be split.*
- *The graph $G$ has a unique non-trivial automorphism $\pi$, which is an involution and described by Table 1 (and it corresponds to the symmetry with respect to the y-axis in Figure 5.7).*

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\pi(i)$ | 10 | 14 | 7 | 13 | 11 | 12 | 3 | 8 | 9 | 1 | 5 | 6 | 4 | 2 |

Table 5.1.: The unique non-trivial automorphism of $G$.

Next, we observe that the 4-claws offer particularly strong constraints. Indeed, if a vertex $v$ is the center of a 4-claw $[v; a, b, c, d]$ and it is split into two vertices $v_1$ and $v_2$ (that is, $f^{-1}(v) = \{v_1, v_2\}$), then, in any solution $G'$, there must be precisely two vertices in $a, b, c, d$, say $a$ and $b$, such that $(v_1, a)$ and $(v_1, b)$ are the edges representing $(v, a)$ and $(v, b)$, whereas $(v_2, c)$ and $(v_2, d)$ are the edges representing $(v, c)$ and $(v, d)$. In the following claims we examine and list some of the constraints on the possible disjoint splits of $G$.

**Claim 5.1.** *Vertex 1 imposes the following constraints:*

1. *There exist three indices $i, j, k$ ($i, j, k \in \{1, 2\}$) such that:*
   - $(1_i, 7_j)$ *is the unique representative of edge* $(1, 7)$,
   - $(1_i, 8_k)$ *is the unique representative of edge* $(1, 8)$ *and*
   - $(7_j, 8_k)$ *is the unique representative of edge* $(7, 8)$.
2. *There exist three indices $i, j, k$ ($i, j, k \in \{1, 2\}$) such that:*
   - $(1_i, 5_j)$ *is the unique representative of edge* $(1, 5)$
   - $(1_i, 6_k)$ *is the unique representative of edge* $(1, 6)$ *and*
   - $(5_j, 6_k)$ *is a representative of edge* $(5, 6)$.

Proof. Vertex 1 is the center of three different 4-claws: $A = [1; 2, 4, 6, 7], B = [1; 2, 4, 6, 8]$ and $C = [1; 2, 4, 5, 8]$. The first point of the claim follows from comparing claws $A$ and $B$. Towards a contradiction, assume without loss of generality that the representatives of $(1, 7)$ and $(1, 8)$ are incident to vertices $1_1$ and $1_2$, respectively. Then, by the pigeonhole principle, it is impossible to place the remaining representatives of the edges $(1, 2)$, $(1, 4)$ and $(1, 6)$ without creating a 3-claw with center either $1_1$ or $1_2$ (we can have at most one of them incident to each of the vertices, and we have three edges and two vertices). Furthermore, if they were both incident to $1_1$ (w.l.o.g.) but there was no representative of edge $(7, 8)$ incident to $1_1$, then it would still be impossible to place the remaining edges without creating a claw. The proof of the second point of the claim follows similarly by comparing claws $B$ and $C$, but now the representative of $(5, 6)$ is not necessarily unique because neither vertex is the center of a 4-claw. ◁

**Claim 5.2.** *Vertex 10 imposes the symmetric constraints of vertex 1.*

Proof. The proof is analogous to the previous one, comparing the symmetric 4-claws. ◁

**Claim 5.3.** *Vertex 3 imposes the following constraints:*

1. *There exist three indices $i, j, k$ ($i, j, k \in \{1, 2\}$) such that:*
   - $(3_i, 5_j)$ *is the unique representative of edge* $(3, 5)$,
   - $(3_i, 6_k)$ *is the unique representative of edge* $(3, 6)$ *and*
   - $(5_j, 6_k)$ *is a representative of edge* $(5, 6)$.
2. *There exist three indices $i, j, k$ ($i, j, k \in \{1, 2\}$) such that:*
   - $(3_i, 10_j)$ *is the unique representative of edge* $(3, 10)$,
   - $(3_i, 11_k)$ *is the unique representative of edge* $(3, 11)$, *and*
   - $(10_j, 11_k)$ *is the unique representative of edge* $(10, 11)$.
3. *There exist three indices $i, j, k$ ($i, j, k \in \{1, 2\}$) such that:*
   - $(3_i, 8_j)$ *is the unique representative of edge* $(3, 8)$,
   - $(3_i, 9_k)$ *is the unique representative of edge* $(3, 9)$, *and*
   - $(8_j, 9_k)$ *is a representative of edge* $(8, 9)$.

Proof. Vertex 3 is the center of four different 4-claws: $A = [3; 4, 5, 9, 10]$, $B = [3; 4, 6, 9, 10]$, $C = [3; 4, 6, 9, 11]$ and $D = [3; 4, 6, 8, 11]$. The first point follows by comparing claws $A$ and $B$; the second one, by comparing claws $B$ and $C$; and the third one, by comparing claws $C$ and $D$. ◁

**Claim 5.4.** *Vertex 7 imposes the symmetric constraints of vertex 3.*

Proof. The proof is analogous to the proofs of the previous claims. ◁

We now use the previous claims to deduce that the way to split vertex 8 is actually unique up to symmetry. First, it is clear that vertex 8 must be split because of the 3-claw $[8; 1, 9, 10]$. Assume vertex 8 is the first vertex of $G$ to be split.

**Claim 5.5.** *Let $G'$ be a solution. Then, the neighborhoods of the two representatives of vertex 8 are uniquely determined, up to symmetry.*

Proof. Let $8_1$ be the representative of 8 adjacent to 10. Then $8_1$ is also adjacent to 3 (by Claim 5.2, as otherwise there would be an induced 5-claw: $[10; 8_1, 3, 14, 13, 12]$). Moreover, $8_1$ is also adjacent to 9 (by Claim 5.3, as otherwise there would be an induced 5-claw: $[3; 8_1, 9, 4, 6, 11]$). As such, vertex 1 cannot be adjacent to $8_1$ (because it would create a 3-claw) and has to be adjacent to $8_2$ instead. We have now the symmetric deductions. Let $8_2$ be the representative of 8 adjacent to 1, then $8_2$ is also adjacent to 7 (by Claim 5.1, otherwise we have a 5-claw $[1; 8_2, 7, 2, 4, 6]$). Moreover, $8_2$ is also adjacent to 9 (by Claim 5.4, otherwise we get a 5-claw $[7; 8_2, 9, 13, 12, 5]$). Note that vertex 10 cannot be made adjacent to $8_2$. This shows that the neighborhoods of the representatives of vertex 8 are fully determined before splitting the other vertices: one representative needs to be adjacent to $10, 3$ and 9, while the other needs to be adjacent to the vertices $1, 7$ and 9. The graph resulting from splitting vertex 8 can be seen in Figure 5.10. ◁

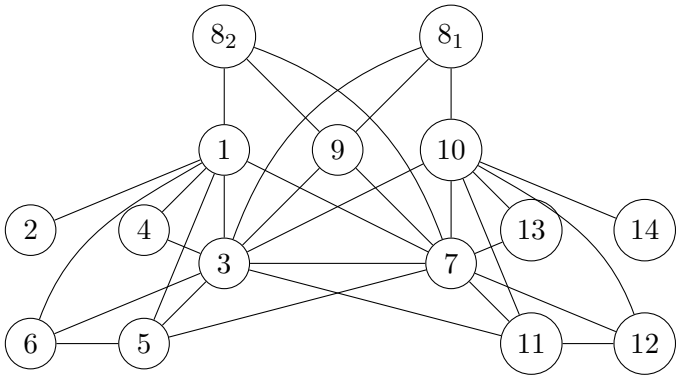



Figure 5.10.: The graph $G$ after the split of vertex 8

If we consider the graph obtained by splitting vertex 8 in the only possible way, we observe that there is a new 4-claw centered at vertex 3 (and, symmetrically, at vertex 7). These 4-claws introduce the following new constraints.

**Claim 5.6.** *Vertex 3 imposes now the following additional constraints:*

1. *There exist three indices $i, j, k$ ($i, j, k \in \{1, 2\}$) such that:*
   - $(3_i, 7_j)$ *is the unique representative of edge* $(3, 7)$*,*
   - $(3_i, 11_k)$ *is the unique representative of edge* $(3, 6)$ *and*
   - $(7_j, 11_k)$ *is the unique representative of edge* $(7, 11)$*.*

*Adding this to the previous constraints, we have that the unique representatives of the edges* $(7, 11)$ *and* $(10, 11)$ *should both be adjacent to either* $3_1$ *or* $3_2$*.*

Proof. This follows from comparing the 4-claws $D = [3; 4, 6, 8, 11]$ and $E = [3; 8_1, 7, 4, 6]$. ◁

At this step, the constraints on vertex 3 allow us to derive the following result.

**Claim 5.7.** *Let $G'$ be a solution. Then, the neighborhoods of the two representatives of vertex 3 are uniquely determined, up to symmetry.*

Proof. By the above, we already know:

1. The representatives of $(3, 7)$ and $(3, 11)$ are incident to each other (because of the 4-claws $[3; 4, 6, 8_1, 7]$ and $[3; 4, 6, 8_1, 11]$).

2. The representatives of $(3, 11)$ and $(3, 10)$ are incident to each other (because of the 4-claws $[3; 4, 6, 9, 11]$ and $[3; 4, 6, 9, 10]$).

3. The representatives of $(10, 3)$ and $(10, 8_1)$ are incident to each other (because of the 4-claws $[10; 14, 13, 12, 3]$ and $[10; 14, 13, 12, 8_1]$) and to a representative of $(3, 8)$.

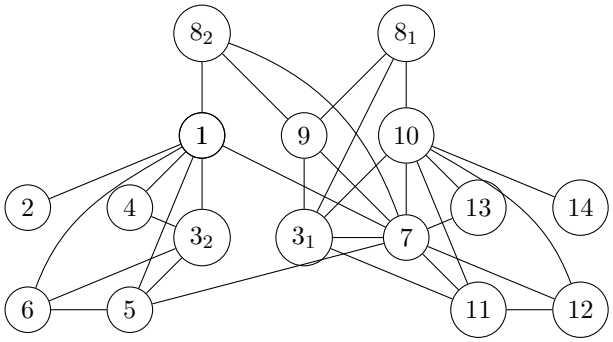



Figure 5.11.: Graph resulting from the split of vertex 3

4. The representatives of $(3, 8_1)$ and $(3, 9)$ are incident to each other (because of the 4-claws $[3; 4, 6, 11, 8_1]$ and $[3; 4, 6, 11, 9]$).

5. There is an unique pair $i, j$ such that $3_i$ and $10_j$ are adjacent (since the edge belongs to a 4-claw $[10; 14, 13, 12, 3]$).

Points 1 and 2 imply that one of the vertices representing 3, w.l.o.g., $3_1$, is adjacent to vertices 7, 11 and 10. Let $10_1$ be the vertex representing 10 and adjacent to $3_1$ (well defined by point 5). Then $10_1$ is adjacent to $8_1$ and it must also be the case that $3_1$ and $8_1$ are adjacent. Otherwise, we would obtain the 5-claw $[10; 14, 12, 13, 3_1, 8_1]$ in the original certificate graph. By point 3, vertex $3_1$ is adjacent to $8_1$, so point 4 implies that it must also be adjacent to 9. Now, $3_1$ cannot have any more neighbors, as otherwise there would be an induced $K_{1,3}$. Therefore, the neighborhoods of the two representatives of vertex 3 are uniquely determined up to symmetry: one of the representatives of 3 is adjacent to 7, 11, 10, 8 and 9; and the other one, to the rest of its neighbors. The graph resulting from splitting vertex 3 can be seen in Figure 5.11. ◁

Finally, we consider vertex 7 and we state that it cannot be split without creating a $K_{1,3}$.

**Claim 5.8.** *There is no way to split vertex 7 without creating a $K_{1,3}$ with center either $7_1$ or $7_2$.*

Proof. If we apply the symmetry of the graph before splitting vertex 3, we would obtain that one of the representatives of vertex 7, say $7_1$ has to be adjacent to 1, 8, 9, 3 and 5. However, since vertices 3 and 8 have already been split, these five edges now create the 3-claw $[7; 8_2, 3_1, 5]$. Furthermore, note that $3_1$ cannot be made adjacent to the other representative of vertex 7, as it also creates a 3-claw ($[7; 3_1, 12, 13]$). Thus, it is not possible to split vertex 7 maintaining $G'$ as a unit interval graph. ◁

The last claim concludes the proof that the $K_{1,5}$-free interval graph presented is not disjoint unit 2-interval. □

To conclude this section, we show that Theorem 5.3 can actually be generalized for disjoint unit $d$-interval graphs for any $d > 2$.

**Corollary 5.2.** *There exists a $K_{1,2d+1}$-free interval graph that is not a disjoint unit $d$-interval graph.*

*Proof.* One can extend the example for $d = 2$ (displayed in Figure 5.7) by adding $2d-4$ common neighbors to vertices 1 and 3 and $2d-4$ common neighbors to vertices 7 and 10. The proof that this graph is not disjoint unit $d$-interval is analogous to the proof for $d = 2$. □

## 5.4. Approximation algorithm

Since we know that given a $K_{1,2d+1}$-free interval graph $G$, we cannot easily determine whether it is *disjoint* unit $d$-interval or not, we now try to give a bound on the value $t$ such that $G$ is always a disjoint unit $t$-interval graph. We prove that for $t = d + 1$, $G$ is always disjoint unit $t$-interval. To do so, we modify the algorithm of Section 5.1 so that given an arbitrary interval representation of graph $G$, it constructs a disjoint unit $(d + 1)$-interval representation $\mathcal{I}'$. First, we let the set of intervals $\mathcal{C}$ initialized to $\mathcal{I}$ be a queue of the intervals in $\mathcal{I}$ ordered by their left endpoints. Then, we replace **Step 4** by the following steps.

**Step 4.1** If $m = 3$ and $I$ is not the center of an $E$-claw, go to **Step 4.2**. If $I$ is the center of an $E$-claw, go to **Step 4.3**.

**Step 4.2** In this case, the step is defined as in **Step 4'** in the proof of Theorem 5.2.

**Step 4.3** If $I$ is an interval associated to a center of an $E$-claw, we define:

$$A_1 = \arg\min_{J \in \mathcal{N}(I)} \{r(J)\} \qquad A_2 = \arg\min_{\{J \in \mathcal{N}(I):\ A_1 \prec J\}} \{r(J)\}$$

and we add the interval $I_1 = [l(I), r(A_2)]$ to $\mathcal{I}'$. Then, we define the set $E(I) := \{J \in \mathcal{I} \mid r(A_2) < l(J) \leqslant r(I)\}$. If $E(I)$ is not empty, all the intersections between $I$ and intervals from $E(I)$ are still not present in the target representation $\mathcal{I}'$. To display them, we will do so in a separate part of the representation, that we will refer to as *auxiliary part of the representation*, and starts at point $M$, where $M-1$ is the length of the original representation $\mathcal{I}$ (equivalently, $M - 1 = \max_{J \in \mathcal{I}}\{r(J)\}$).

In the auxiliary part, we add the interval $I_2 = [r(A_2) + M, r(I) + M]$, associated to $I$, and then we distinguish two different cases:

- No interval from $E(I)$ is already represented by an auxiliary interval in $\mathcal{I}'$. In this case, for every $J \in E(I)$, we define $J_a = [l(J) + M, \min\{r(I) + M, r(J) + M\}]$. Note that every $J_a$ is contained in $I_2$.
- Otherwise, let $E_l(I)$ be the set of intervals of $E(I)$ which are already represented by an auxiliary interval. That is, $J \in E_l(I)$ if there is an interval

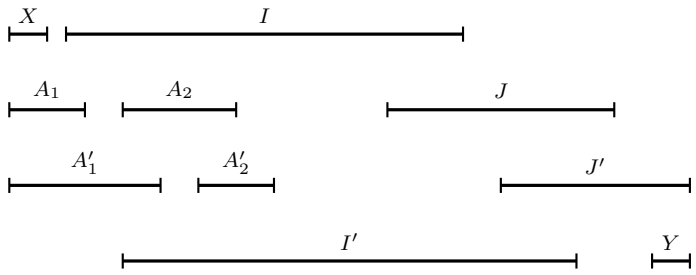



Figure 5.12.: Interval representation of a graph with two vertices that are centers of an $E$-claw, represented by intervals $I$ and $I'$.

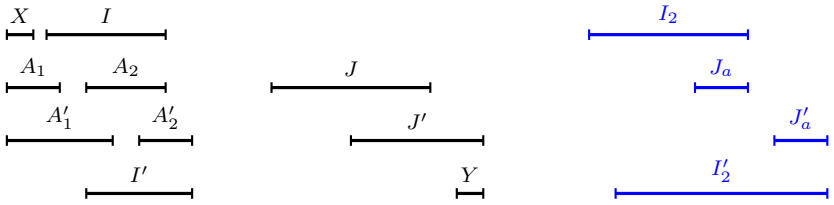



Figure 5.13.: The representation $\mathcal{I}'$ returned by the algorithm for the input representation in Figure 5.12. In blue, the intervals that belong to the auxiliary part of $\mathcal{I}'$. The interval $J_a$, which is in $E_l(I')$ and was added while processing interval $I$, intersects $I'_2$.

$J_a$ which was already added to the auxiliary part while processing a center $I'$ with $J \in E(I')$. For every interval $J \in E(I) \setminus E_l(I)$, we define $J_a = [l(J) + M, \min\{r(I) + M, r(J) + M\}]$ as before. On the other hand, the intersections between intervals of $E_l(I)$ and $I$ are already present in $\mathcal{I}'$, since the auxiliary intervals associated to intervals of $E_l(I)$ that were already present intersect $I_2$, because the left endpoints of the intervals follow the same order as in $\mathcal{I}$. This step is illustrated in Figure 5.13, which shows the output of the algorithm when the input is the interval representation in Figure 5.12.

Note that we do not check whether there is an interval associated to the center already in the auxiliary part, which could happen if the interval $I$ belonged to $E(I')$ for some interval $I'$ processed before, because in that case we add another auxiliary interval associated to the center. Note that when we process an interval $I$ as a center of an $E$-claw, we denote the corresponding auxiliary interval by $I_2$, while when we process it as an inteval of $E(I')$, we denote the auxiliary interval added by $I_a$, so we will not have two intervals with the same label.

After adding all the intervals to $\mathcal{I}'$, go to **Step 6**.

It is then clear, by construction, that the intersections of $\mathcal{I}$ are preserved. We now prove that the representation $\mathcal{I}'$ constructed in this modified algorithm is indeed a disjoint unit $(d+1)$-interval representation. We start with two simple observations.

**Observation 5.4.** *Let $I$ be a center of an $E$-claw. There do not exist two intervals $J$ and $J'$ in $E(I)$ that are disjoint.*

Proof. Suppose $J$ and $J'$ are two intervals of $E(I)$ that are disjoint. Without loss of generality, we can assume that $J \prec J'$. Then, $A_1, A_2, J$ and $J'$ would be four pairwise disjoint intervals that intersect $I$, contradicting the fact that $I$ was the center of a maximal 3-claw. ◁

**Observation 5.5.** *Let $J$ be an interval contained in the sets $E(I)$ and $E(I')$ for two centers $I$ and $I'$. Then, $I$ and $I'$ intersect.*

Proof. Let $A_1, A_2$ be the intervals defined in **Step 4.3** when processing $I$, and $A'_1, A'_2$ the intervals defined when processing $I'$. By definition, $r(A_2) < l(J) \leqslant r(I)$ and $r(A'_2) < l(J) \leqslant r(I')$. Without loss of generality, suppose that $r(A_2) \leqslant r(A'_2)$. Then, $l(I') \leqslant r(A_2)$, as otherwise the intervals $A_1, A_2, A'_2$ and $J$ would form a set of four pairwise disjoint intervals intersecting $I$, which contradicts the fact that it is a maximal 3-claw. ◁

The previous observation implies that given a set of centers $\{I_i \mid i \in \{1, \ldots, n\}\}$ (for some $n \in \mathbb{N}$) such that $J$ belongs to $E(I_i)$ for every $i$, all the centers are pairwise intersecting (i.e., they form a clique).

**Lemma 5.5.** *No interval in the auxiliary part of $\mathcal{I}'$ intersects three pairwise disjoint intervals.*

*Proof.* Recall that in **Step 4.3**, an interval $J$ processed as an interval of $E(I)$ for some center $I$ which is added to the auxiliary part of $\mathcal{I}'$ is denoted by $J_a$, while an interval $I$ processed as a center of an $E$-claw added to the auxiliary part is denoted by $I_2$. Since every interval $J_a$ is contained in an interval $I_2$ for some $I \in \mathcal{I}$, it suffices to prove that for every center $I \in \mathcal{I}$, the auxiliary interval $I_2$ (if it exists) does not intersect three pairwise disjoint intervals. To prove this, notice that when the interval $I_2$ is added to $\mathcal{I}'$, by Observation 5.4, it cannot intersect two disjoint intervals. In a later iteration, the only intersections that can arise are with centers of $E$-claws. However, by Observation 5.5, the set of these intervals cannot contain two disjoint intervals. Thus, $I_2$ can intersect at most two disjoint intervals. □

**Lemma 5.6.** *No interval of $\mathcal{I}'$ intersects three pairwise disjoint intervals.*

*Proof.* Every interval which is not in the auxiliary part satisfies the properties of Observation 5.1, so we already know that this part of the representation does not contain an interval intersecting three pairwise disjoint intervals. On the other hand, in the auxiliary part of the representation, the result follows by Lemma 5.5. □

Since we have proven that the representation can be transformed into a disjoint unit multiple interval representation, it only remains to show that every vertex is represented by at most $d + 1$ intervals.

**Lemma 5.7.** *For every vertex $v$, the multiple interval associated to $v$ has at most two auxiliary intervals. In particular, if it has two auxiliary intervals, then $v$ is the center of an $E$-claw.*

*Proof.* Fix a vertex $v$ and let $I$ be the interval representing it in $\mathcal{I}$. We will distinguish two different cases:

- Let $v$ be a center of an $E$-claw that has been processed in an iteration of the algorithm. Then, an auxiliary interval $I_2$ has potentially been added to $\mathcal{I}'$. However, no other auxiliary interval can be added, since $I$ cannot belong to a set $E(J)$ for any interval $J$ not processed yet, as $l(I) \leqslant l(J)$. Thus, the statement holds.
- Let $I \in E(J)$ for some $J$. If $I$ is in $E(J')$ for some other $J'$, then $J$ and $J'$ intersect, and the same auxiliary interval $I_a$ is used to display the intersections with $J$ and $J'$. The only way a new auxiliary interval of $I$ can be added is if it is processed in a later step as center of an $E$-claw. In that case, a new auxiliary interval $I_2$ will be potentially added to display the intersections with the intervals in $E(I)$, but then we would be in the first case, so at most two auxiliary intervals will be added in total.

□

The correctness of the previous algorithm implies the following result.

**Theorem 5.4.** *Given a $K_{1,2d+1}$-free interval graph, there exists a polynomial-time algorithm that either determines that the graph is disjoint unit $d$-interval or returns a disjoint unit $(d+1)$-interval representation.*

*Proof.* The algorithm first checks the existence of $E$-claws. If there are no $E$-claws, it determines that the graph is disjoint unit $d$-interval by Theorem 5.2. Otherwise, using the previous algorithm, it constructs a disjoint unit $(d+1)$-interval representation (it is clear that every vertex is represented by at most $d+1$ intervals since every vertex that is not a center of an $E$-claw has a single auxiliary interval, while centers, which have a single non-auxiliary interval, can have at most two auxiliary intervals no matter the value of $d$). □

As a final remark, notice that even if the previous algorithm returns a $(d+1)$-interval representation, the input graph could be a disjoint unit $d$-interval graph. Thus, the algorithm is actually an approximation algorithm with an additive error of one.

## 5.5. Containment relations between different subclasses of multiple interval graphs

In this section, we analyze the relations between different subclasses of multiple interval graphs. We have already seen that 2-interval graphs and disjoint 2-interval graphs are equivalent. Furthermore, the results from the previous sections imply that the class of

disjoint unit 2-interval graphs is properly contained in the class of unit 2-interval graphs. In the following, we summarize the containment relations between unit 2-interval graphs, disjoint unit 2-interval graphs, balanced 2-interval graphs and disjoint balanced 2-interval graphs (see Figure 5.14 for a graphical illustration).

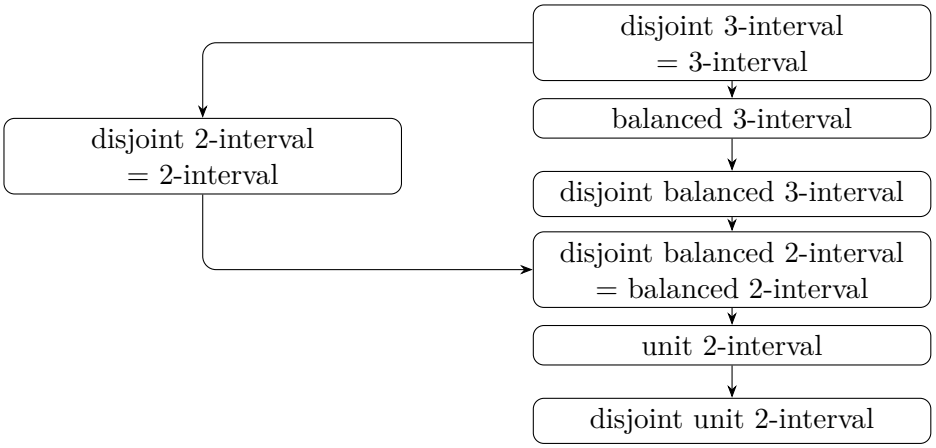



Figure 5.14.: Landscape of graph subclasses of 3-interval graphs. An arrow from a graph class $\mathcal{C}$ to a $\mathcal{C}'$ indicates that $\mathcal{C}' \subset \mathcal{C}$.

**Theorem 5.5.** *There exist the following containment relations between the subclasses of 2-interval graphs:*

1. *The classes of* 2-interval *and* disjoint 2-interval *graphs are* equivalent.

2. *The classes of* balanced 2-interval *and* disjoint balanced 2-interval *graphs are* equivalent.

3. *The class of* unit 2-interval graphs *is* properly contained *in the class of* disjoint balanced 2-interval *graphs.*

4. *The class of* disjoint unit 2-interval *graphs is* properly contained *in the class of* unit 2-interval *graphs.*

*Proof.* 1. See Observation 3.1.

2. To see that the classes of balanced and disjoint balanced 2-intervals are equivalent, it suffices to observe that if we have a balanced 2-interval representation, then for every pair of intersecting intervals $[a, b]$ and $[c, d]$ (with $a \leqslant c \leqslant b \leqslant d$) associated to the same vertex, we can stretch both intervals until the middle point of their intersection, i.e., we can replace both intervals by $[a, a + \frac{b-c}{2} - \epsilon]$ and $[a + \frac{b-c}{2} + \epsilon, d]$, respectively, for some $\epsilon$ sufficiently small. This procedure yields a disjoint balanced 2-interval representation.

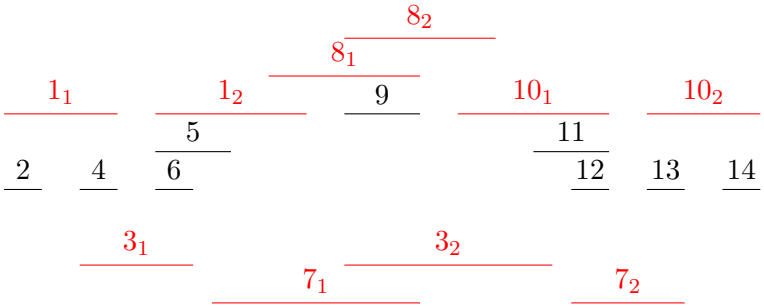



Figure 5.15.: Unit 2-interval representation of the graph in Figure 5.7.

3. To see that unit 2-interval graphs have a disjoint balanced 2-interval representation, it suffices to apply the same procedure as in 2. Now the two obtained intervals won't have unit length any longer, but the length of both intervals will be the same. The inclusion is proper because $K_{5,3}$ is disjoint balanced 2-interval (see [68, Fig. 3]) but not unit 2-interval (the latter follows from the fact that every vertex of degree 3 is adjacent to 5 independent vertices, so one of the intervals must intersect more than two disjoint intervals).

4. As shown before, the graph in Figure 5.7 is not disjoint unit 2-interval, but applying the algorithm presented in Section 5.1, one can obtain a unit 2-interval representation (see Figure 5.15) of it. This proves that the class of disjoint unit 2-interval graphs is properly contained in the class of unit 2-interval graphs.

□

We finish by showing that the previous theorem cannot be completely generalized for the subclasses of $d$-interval graphs, as the class of balanced $d$-interval graphs is not equivalent to the class of disjoint balanced $d$-interval graphs for $d > 2$. We first construct a graph that is balanced 3-interval but not disjoint balanced 3-interval and then show how to generalize this construction for every $d > 3$.

**Theorem 5.6.** *The class of disjoint balanced 3-interval graphs is properly contained in the class of balanced 3-interval graphs.*

*Proof.* We construct a graph $G$ which is balanced 3-interval but not disjoint balanced 3-interval. The high-level idea of the construction is that for a particular vertex, one of its intervals is forced to a given length, while the other two are forced to be placed somewhere where there is not enough space for both of them, and thus they cannot be disjoint (note that the difference with the case $d = 2$ is that now, if we stretch two of the intervals so that they do not intersect, we also have to modify the length of the third interval, and as we show here, this is not always possible). To enforce these constraints, we use the complete bipartite graph $K_{11,4}$ as a gadget and exploit the fact that any 3-interval representation of this gadget must be continuous (i.e., the union of the intervals in its underlying family is an interval) [148, Lemma 2] (see also [68, Fig. 3] for the idea of its representation).

We construct $G$ as follows: we connect in a chain five $K_{11,4}$'s, to which we add six vertices $v_1$, $v_2$, $v_3$, $v_4$, $v_5$, $v_6$ (Figure 5.16 shows how to link $v_1$, $v_2$, $v_3$, $v_4$ to the chain, while vertices $v_5$ and $v_6$ mimic the behavior of $v_3$ and $v_4$ with a different set of neighbors, namely, $v_5$ is connected to the corresponding vertices of the first two $K_{11,4}$'s, and $v_6$ is connected to the corresponding vertices of the second and the third $K_{11,4}$'s). More precisely, let $C_i$, with $i \in \{1, \ldots, 5\}$, be the five $K_{11,4}$'s forming the chain, enumerated from left to right. Moreover, for every $C_i$, let $f_i^j$ with $j \in \{1, \ldots, 11\}$ be the eleven vertices of one side of the bipartition, and $t_i^k$, with $k \in \{1, 2, 3, 4\}$, the four vertices of the other side of the bipartition. We assume that the chain is connected such that $f_i^{11}$ is linked to $f_{i+1}^1$. Then, $v_1$ is connected to all the vertices of $C_2$ and $C_4$, and to $f_3^{11}$ and $f_5^1$, plus another independent vertex. Similarly, $v_2$ is connected to all the vertices of $C_2$ and $C_4$, to $v_1$ and to $f_1^{11}$ and $f_3^1$. On the other hand, $v_3$ is connected to $f_3^{11}$, $t_3^4$, $t_4^1$ and $f_4^j$ for $j \in \{1, \ldots, 9\}$; while $v_4$ is connected to $f_5^1$, $t_4^4$, $t_5^1$ and $f_4^j$ for $j \in \{3, \ldots, 11\}$. Finally, $v_5$ is connected to $f_1^{11}$, $t_1^3$, $t_2^1$ and $f_2^j$ for $j \in \{1, \ldots, 7\}$, as well as $f_4^8$ and $f_4^9$, whereas $v_6$ is connected to $f_3^1$, $t_2^4$, $t_3^1$ and $f_2^j$ for $j \in \{5, \ldots, 11\}$, as well as $f_4^8$ and $f_4^9$. The vertices $v_1$ and $v_2$ are both connected to $v_3, v_4, v_5$ and $v_6$.

Now, as any 3-interval representation of a $K_{11,4}$ is continuous, any realization of $G$ groups the five $K_{11,4}$'s in a block [148]. For $j \in \{1, 2, 3\}$, let $I_j$ be the intervals associated to $v_1$, $J_j$ the intervals associated to $v_2$, and $K_j$ the intervals associated to $v_3$. First, it is clear that we need three different intervals to cover the neighbors of $v_1$ (and these three intervals must be disjoint). Instead, the neighbors of $v_2$ could be covered only with two intervals. However, we will see that the two segments of the real line that need to be covered cannot have the same length (assuming that the 3-interval associated to $v_1$ is balanced). We will show that we need two intersecting intervals to cover the first segment.

Suppose that only two intervals are needed to represent the adjacencies of $v_2$, and let $J$ be the interval displaying the edges between $C_2$ and $v_2$, and $J_3$ the interval displaying the edges between $C_4$ and $v_2$. Similarly, let $I_1$ be the interval associated to $v_1$ used to represent the edges with $C_2$, let $I_2$ the interval used to represent the edges with $C_4$, and $I_3$ the interval displaying the edge with the isolated vertex. One can easily see that $J_3$ is properly contained in $I_2$ (since $I_2$ must also intersect an interval associated to $f_3^{11}$ on its left and an interval associated to $f_5^1$ on its right), while $I_1$ is properly contained in $J$ (by an analogous argument). Thus, $len(J_3) < len(I_2) = len(I_1) < len(J)$. In order for the representation to be balanced, the segment of the real line covered by $J$ needs to be covered by two different intervals, say $J_1$ and $J_2$. To prove that $G$ is balanced 3-interval but not disjoint balanced 3-interval, we need to bound $len(J) - len(J_3)$. In particular, we need $len(J) - len(J_3) < len(J_3)$. Vertices $v_3$ and $v_4$ will allow us to find constants $a$ and $a'$ to bound $len(I_2) - len(J_3) \leqslant a + a'$, while vertices $v_5$ and $v_6$ will serve to find constants $b$ and $b'$ to bound $len(J) - len(I_1) \leqslant b + b'$. By showing that we can force the constants such that $a + a' + b + b' < len(J_3)$, we have the result. This will follow since we will have eight pairwise disjoint intervals properly contained in $J_3$: two of length $a$, two of length $a'$, two of length $b$ and two of length $b'$.

Indeed, let $a$ and $a'$ be the lengths of the intervals associated to $v_3$ and to $v_4$, re-

spectively. The next claim implies that there are two disjoint intervals associated to $v_3$ properly contained in $J_3$, and another disjoint interval that properly contains the segment between $l(I_2)$ and $l(J_3)$, and so $l(J_3) - l(I_2) < a$.

**Claim 5.9.** *Let $G$ be a graph formed by the union of a $K_{11,4}$ and a vertex $v$ which is adjacent to nine vertices in $S_{11}$, where $S_{11}$ denotes the side of the bipartition with eleven vertices. Then, vertex $v$ must be represented by three pairwise disjoint intervals, two of which are each properly contained in an interval representing a vertex of $S_{11}$.*

Proof. Let $S_{11}$ and $S_4$ denote the two sides of the bipartition of the $K_{11,4}$, and let $f^j$ with $j \in \{1, \ldots, 11\}$ denote the vertices of $S_{11}$ and $t^j$ with $j \in \{1, \ldots, 4\}$ denote the vertices of $S_4$. Since every interval of $S_{11}$ intersects four pairwise disjoint intervals, every $f^j$ must be represented by three intervals: one that intersects two intervals $t^i, t^j$, with $i, j \in \{1, \ldots, 4\}$, and two intervals contained each in one of the remaining $t^i$'s. Indeed, if an interval associated to $f_j$ intersected three intervals $t_i$ with $i \in \{1, \ldots, 4\}$, then there would only be nine gaps between intervals of the form $t_i$. That is, only nine intervals of the form $f_j$ would be able to intersect two intervals $t_i$ at the same time, which implies that one vertex of $S_{11}$ needs to intersect four pairwise disjoint intervals, but no interval associated to that vertex can intersect two of those intervals at the same time. Since we only have three intervals associated to each vertex, this leads to a contradiction. Thus, for any given $t^i$, there are at least four vertices $f^j$ such that no interval associated to them is contained in an interval of $t^i$. Since there are eleven vertices in $S_{11}$ and nine are adjacent to $v$, at least two of these nine vertices will not be contained in $t^1$. In particular, they will also not intersect the first interval of $t^1$. Thus, one of the intervals of $v$ must intersect $t^1$ and seven of the $f_4^j$, while the other two intervals of $v$ are contained each in one of the intervals of the remaining $f^j$'s, w.l.o.g., $f^8$ and $f^9$ (recall that the two remaining $f^j$'s fill the gaps in between intervals of the from $t^i$, and so it is possible to display these intersections without creating any new ones).

Finally, the reader can observe that this gadget admits a balanced 3-interval representation, as one can dilate the two holes that are covered by $f^8$ and $f^9$ as needed (and since no interval of these vertices is contained in an interval of $t^1$, modifying their length will have no effect on the length of the first interval of $v$). More precisely, consider a representation where an interval of $t^1$ contains intervals of $f^1, f^2, f^3, f^4, f^5, f^6$ and intersects $f^7$; an interval of $t^4$ contains $f^3, f^4, f^5, f^7, f^{10}, f^{11}$ and intersects $f^6$; and an interval of $v$ contains $t_1$. Then, $t_2$ and $t_3$ must contain an interval of $f^8$ and $f^9$, which must have length at least the length of $v$ plus one (because another interval associated with them contains an interval of $v$). On the other hand, $t_2$ also contains $f^1, f^2$, and $f^6$, while $t_3$ contains $f^7, f^{10}$, and $f^{11}$. Thus, let all $f^i$ have length three except $f^8$ and $f^9$. We can then take $t^1$ and $t^2$ to have length $6(3+1)+1 = 25$. Therefore, $l(v) \geqslant 25$, and it suffices to take $l(f^8) = l(f^9) = l(v)+4$. Finally, we can take $l(t_2) = l(t_3) = 2(l(v)+5)+3(4)+3$. In fact, note that we can take $l(f^8)$ and $l(f^9)$ arbitrarily large, and so, in the complete construction, an interval associated to $f_4^8$ and an interval associated to $f_4^9$ can each contain intervals associated to $v_4, v_5$ and $v_6$, while still maintaining the balanced property. ◁

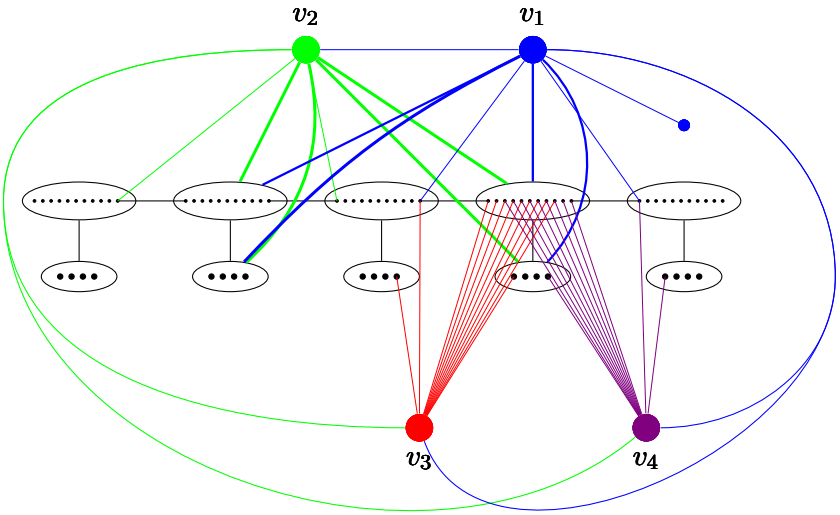



Figure 5.16.: $G$ is balanced 3-interval but not disjoint balanced 3-interval. $K_{11,4}$ graphs are drawn abstractly and are chained. A thick edge stopping at the border of the ellipse means that the vertex is connected to every vertex in the corresponding part of the $K_{11,4}$. Vertices $v_5$ and $v_6$ are omitted for readability purposes.

Similarly, the segment between $r(I_2)$ and $r(J_3)$ is also contained in an interval associated to $v_4$, which has the same properties as $v_3$ and does not intersect any interval associated to $v_3$. This proves that there are two intervals of length $a$ and two intervals of length $a'$ (all pairwise disjoint) contained in $J_3$. Doing the same to bound $l(I_1)-l(J)$ and $r(J)-l(I_1)$, we get the result. Thus, to represent $v_2$, we need two intervals associated to $v_2$ to intersect. If we do not allow intersection, the length of these two intervals will be smaller than the length of the third interval associated to $v_2$, contradicting the fact that they are balanced. □

**Corollary 5.3.** *The class of disjoint balanced $d$-interval graphs is properly contained in the class of balanced $d$-interval graphs for every natural number $d \geqslant 3$.*

*Proof.* The construction used in the proof of Theorem 5.6 can be modified to yield a balanced $d$-interval graph by adding new isolated neighbors to $v_1$ and $v_2$ and replacing the chain of $K_{11,4}$'s by a chain of $K_{d^2+d-1,d+1}$'s, as any $d$-interval representation of this gadget must be continuous [148]. Furthermore, we modify the adjacencies of vertices $v_3, v_4, v_5$ and $v_6$: vertex $v_3$ is now connected to $f_3^{11}$, $t_3^4$, $t_4^1$ and $f_4^j$ for $j \in \{1, \ldots, d^2-d+1-2d+4\}$; while $v_4$ is connected to $f_5^1$, $t_4^4$, $t_5^1$ and $f_4^j$ for $j \in \{2d-3, \ldots, d^2+d-1\}$, and the adjacencies of $v_5$ and $v_6$ are also modified accordingly. The same arguments used in Theorem 5.6 prove that this constructed graph is not a disjoint balanced $d$-interval graph. □

The results of this chapter highlight the importance of expliciting whether we define $d$-intervals as the union of $d$ disjoint or $d$ not necessarily disjoint intervals when we impose length restrictions on them. Indeed, both definitions lead to different classes of graphs. We have completely characterized unit $d$-interval graphs that are also interval as $K_{1,2d+1}$-free interval graphs, but it is still unclear how to recognize disjoint unit $d$-interval graphs that are also interval. It remains as an open question whether they can be characterized by a list of forbidden induced subgraphs, or even if they can be recognized in polynomial time. Finally, we have obtained a relatively complete landscape of the containment relations between different subclasses of 2-interval graphs, that cannot be fully generalized for $d > 2$. In particular, for $d > 2$, it is still unknown whether the class of unit $d$-interval graphs is contained in the class of disjoint balanced $d$-interval graphs.

# 6. Complexity of Recognition

In this chapter, we study the complexity of recognizing unit (disjoint) $d$-interval graphs, which was the major open question in [96]. We start with a small discussion in Section 6.1 on why the unit case is so different from the rest of subclasses of multiple interval graphs, and why the existing hardness proofs cannot be easily adapted. We then move on to the main result of the chapter, presented in Section 6.2: the NP-hardness of DISJOINT UNIT 2-INTERVAL RECOGNITION, which is obtained with a completely different approach from the one used for the rest of the subclasses of multiple interval graphs. We extend this hardness result to disjoint unit $d$-interval graphs for any $d \geqslant 2$ (recall that this does not follow directly in graph recognition problems, as discussed in Section 2.2) and obtain two important corollaries, namely that recognizing $(x, \ldots, x)$ $d$-interval graphs and depth $r$ disjoint unit 2-interval graphs is NP-complete for every $x \geqslant 11$ and every $r \geqslant 4$. Then, in Section 6.3, we show that the construction used in the NP-hardness proof of DISJOINT UNIT 2-INTERVAL RECOGNITION can also be adapted for the non-disjoint case. Finally, in Section 6.4, we observe that recognizing (disjoint) balanced $d$-interval graphs is NP-hard for every $d \geqslant 2$ (while only the case $d = 2$ had been studied).

## 6.1. Why is the unit case different?

To prove the NP-hardness of recognizing disjoint $d$-interval graphs, West and Shmoys gave a reduction from the problem Hamiltonian circuit in cubic triangle-free graphs [148]. This proof was then adapted by Gyárfás and West to prove the NP-hardness of recognizing 2-track interval graphs [85], and by Gambette and Vialette to extend the hardness result of recognizing disjoint 2-interval graphs to balanced 2-interval graphs [68]. Later on, Jiang also used the problem Hamiltonian circuit in cubic triangle-free as a starting point to extend the hardness of recognizing 2-track interval graphs to $d$-track interval graphs [95]. All these reductions are based on the same simple observation: if a cubic graph $G$ has a Hamiltonian path between two vertices of some edge $(u, v)$, then $(u, v)$ closes the Hamiltonian path into a Hamiltonian circuit, and the edges not in the circuit form a perfect matching. The challenging part of the reduction is to separate the circuit from the matching.

In [148], [68] and [85], the separation is achieved by constructing a supergraph of the original graph $G$ with a special vertex $z$ which is adjacent to the $n$ vertices of $G$ and to some additional gadgets. In the words of Jiang: "the edges of the circuit are enclosed in a "cage" formed by $z$ and some additional gadgets in the supergraph, and the edges of the matching stay outside" [95]. Since the special vertex $z$ is adjacent to $n$ different vertices of a triangle-free graph, it has to be represented by a long interval, and so these

reductions cannot be easily adapted to the case where the intervals need to have unit length.

Instead, in the hardness proof of recognizing $d$-track interval graphs, Jiang managed to avoid the use of such special vertex by restricting the edges of the circuit and of the matching to two separate tracks. This also worked in the case of balanced and unit $d$-track interval graphs. However, once again, this technique cannot be adapted for (disjoint) unit $d$-interval graphs, as we cannot use different tracks for the separation.

Therefore, given the difficulty of achieving the separation of the matching and the circuit in a (disjoint) unit $d$-interval representation, we approach the problem from a different perspective and we reduce directly from SATISFIABILITY.

## 6.2. Hardness of recognizing disjoint unit multiple interval graphs

In this section, we show that recognizing disjoint unit 2-interval graphs is NP-complete, and use this to prove the hardness of recognizing unit $d$-intervals for every $d \geqslant 2$. The result for $d = 2$ is obtained in two steps. First, we define a more general version of the problem, which we call COLORED DISJOINT UNIT 2-INTERVAL REPRESENTATION, and prove that it is NP-complete. Then, we reduce this new problem to DISJOINT UNIT 2-INTERVAL RECOGNITION.

Let us introduce the more general problem, COLORED DISJOINT UNIT 2-INTERVAL REPRESENTATION.

COLORED DISJOINT UNIT 2-INTERVAL RECOGNITION

***Instance:*** A graph $G = (V, E)$ and a coloring $\gamma : V \to \{\texttt{white}, \texttt{black}\}$.
***Goal:*** Decide whether $G$ has a representation where:

- each white vertex is represented by a disjoint unit 2-interval,
- each black vertex is represented by a single unit interval.

We refer to this representation as a *colored disjoint unit 2-interval representation.*

### 6.2.1. Hardness of Colored Unit 2-Interval Recognition

To prove that COLORED DISJOINT UNIT 2-INTERVAL RECOGNITION is NP-complete, we reduce from a variant of SATISFIABILITY. Let us first introduce this variant of SATISFIABILITY. In the following, we use the term "$j$-clause" to refer to a clause that contains exactly $j$ literals.

**Lemma 6.1** ([60])**.** *SATISFIABILITY is* NP*-complete even when restricted to CNF-formulae such that:*

1. *Every clause contains either* 3 *literals (*3*-clause) or* 2 *literals (*2*-clause).*

2. *Each variable appears in exactly one* 3*-clause.*

3. *Each* 3*-clause is positive monotone, i.e., is comprised of three positive literals.*

4. *Each variable occurs exactly in three clauses, once negated and twice positive.*

*Proof.* This lemma is proven in [60, Lemma 2.1]. Note that condition (2) is not explicitly stated in the Lemma's original statement. However, upon close examination of the proof of Lemma 2.1 given in [60], one can see that condition (2) holds for all the instances of SATISFIABILITY produced by the proposed reduction if we reduce from an instance of 3-SAT. Specifically, in the proof, each occurrence of a variable in the original formula is replaced by a new variable, and each new variable (which corresponds to an occurrence of an original variable) also appears in two new 2-clauses. Since the new variable occurs only in these three clauses, it follows that there is exactly one occurrence in a 3-clause if the original instance is an instance of 3-SAT. □

We can now proceed to the proof of hardness of COLORED DISJOINT UNIT 2-INTERVAL RECOGNITION.

**Theorem 6.1.** *COLORED DISJOINT UNIT 2-INTERVAL RECOGNITION is NP-complete, even for graphs where the white vertices have degree at most 6 and the black vertices have degree at most 5.*

The rest of the subsection is dedicated to the proof of Theorem 6.1. We first describe the construction used for the reduction and then prove its correctness.

**Construction** Let $\Psi$ be an instance of the variant of SAT described in Lemma 6.1, formed by a set of Boolean variables $x_1, \ldots, x_n$ and a set of clauses $C_1, \ldots, C_m$. We construct an equivalent instance $(G_\Psi, \gamma_\Psi)$ of COLORED DISJOINT UNIT 2-INTERVAL RECOGNITION as follows.

For every variable $x_i$, we introduce the variable gadget $\hat{V}_i$ (truth setting component), which is the vertex-colored graph on three black vertices $A_i$, $B_i$, $C_i$ and three white vertices $x_i^1$, $x_i^2$ and $x_i^N$, with all edges between a black vertex and a white vertex, plus the edges $(x_i^1, x_i^2)$, $(C_i, A_i)$ and $(C_i, B_i)$. We anticipate that the white vertices of $\hat{V}_i$ will be adjacent also to vertices outside $\hat{V}_i$; in order to underline this distinction, these three vertices are called *public*, and the black vertices are called *private*.

The variable gadget $\hat{V}_i$ is illustrated in Figure 6.1. Notice that the three white vertices $x_i^1, x_i^2, x_i^N$ correspond each to precisely one of the occurrences of the represented variable $x_i$: vertex $x_i^1$ represents the positive occurrence in a 3-clause, vertex $x_i^2$ represent the positive occurrence in a 2-clause and vertex $x_i^N$ represents the negated occurrence of $x_i$. Therefore, we refer to them as *literal vertices*. Furthermore, note that a vertex of $\hat{V}_i$ is adjacent to $A_i$ if and only if it is adjacent to $B_i$; and being private, these two vertices will remain false twins also in $G$. We will exploit this symmetry to simplify the case analysis.

To conclude the construction, we show how to encode each clause $C_\alpha$, for $\alpha = 1, \ldots, m$. If $C_\alpha$ is a 3-clause, then it is monotone positive, i.e., $C_\alpha = (x_i \vee x_j \vee x_k)$ for some

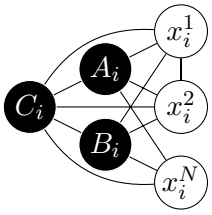



Figure 6.1.: Variable gadget $\hat{V}_i$ corresponding to a variable $x_i$. Black vertices are displayed with a black background.

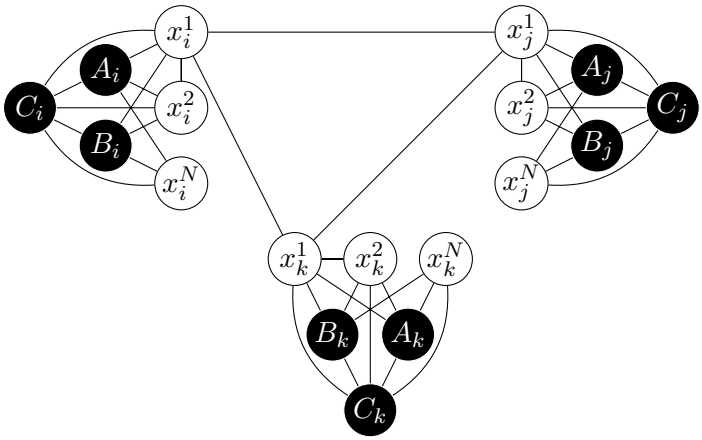



Figure 6.2.: Clause gadget $\hat{C}_\alpha$ associated to a 3-clause $C_\alpha = (x_i \vee x_j \vee x_k)$. Note that in the final graph, each vertex $x_i^m, x_j^m, x_k^m$, for every $m \in \{1, 2, N\}$, will be incident to exactly 2 edges linking them to vertices outside their variable gadget.

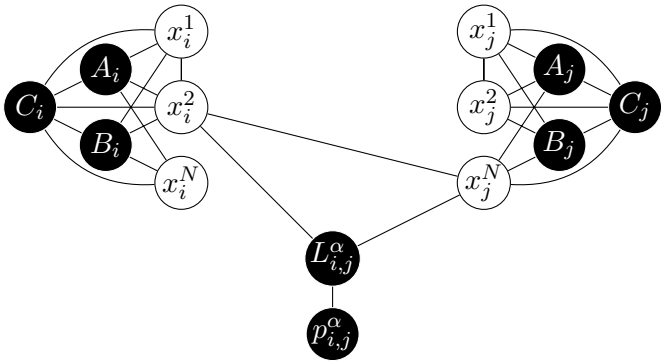



Figure 6.3.: Gadget for a 2-clause $\hat{C}_\alpha$ of the form $C_\alpha = (x_i \vee \overline{x}_j)$.

$i, j, k \in \{1, \ldots, n\}$. In this case, we introduce the three edges $(x_i^1, x_j^1)$, $(x_j^1, x_k^1)$, $(x_k^1, x_i^1)$, which comprise the clause gadget (see Figure 6.2).

If $C_\alpha$ is a 2-clause, say $C_\alpha = (x_i^r \vee x_j^s)$ with $i, j \in \{1, \ldots, n\}$ and $r, s \in \{2, N\}$, then we introduce a public black vertex $L_{i,j}^{\alpha}$ with a private black neighbor $p_{i,j}^{\alpha}$ and we add the four edges $(x_i^r, x_j^s)$, $(x_i^r, L_{i,j}^{\alpha})$, $(x_j^s, L_{i,j}^{\alpha})$ and $(L_{i,j}^{\alpha}, p_{i,j}^{\alpha})$. These four edges together with the two vertices added comprise the clause gadget (see Figure 6.3).

The description of the reduction is now complete. Clearly, $G_\Psi$ has at most $6n + 2m$ vertices and at most $12n + 4m$ edges. Next, we introduce a few notions to ease the proof that $G_\Psi$ is a colored disjoint unit 2-interval graph if and only if $\Psi$ is satisfiable. In particular, we extend the notion of *split* given in Definition 4.2 to colored graphs.

**Definition 6.1.** *Given a colored graph $(G, \gamma)$, we say that a pair $(S, f)$ formed by a graph $S$ and a function $f : V(S) \mapsto V(G)$ is a disjoint split of $(G, \gamma)$ if $f$ satisfies the following conditions:*

- $|f^{-1}(v)| = 1$ *for every* $v \in V(G)$ *with* $\gamma(v) =$ `black`.
- $|f^{-1}(v)| = 2$ *for every* $v \in V(G)$ *with* $\gamma(v) =$ `white`.
- *For every vertex $v$ of $G$, $f^{-1}(v)$ is an independent set in $S$.*
- *For every edge $(s, t)$ of $S$, $(f(s), f(t))$ is an edge of $G$.*
- *For every edge $(u, v)$ of $G$, there exist two vertices $s$ and $t$ in $f^{-1}(\{u, v\})$ such that $(s, t)$ is an edge of $S$.*

**Definition 6.2.** *We define the family of disjoint splits of a colored graph $G$ that lead to a unit interval graph as $\mathcal{S}_{\mathcal{U}}(G) := \{(S, f) \mid (S, f)$ is a split of $G$ and $S$ is a unit interval graph$\}$.*

Note that the assumption that the 2-interval used to represent a white vertex is composed of two disjoint intervals is enforced by the third condition of Definition 6.1.

The next lemma shows how a disjoint split $(S, f)$ of a colored graph $G$ can be used to certify that $G$ is a colored disjoint unit 2-interval graph.

**Lemma 6.2.** *A colored graph $(G, \gamma)$ is a colored disjoint unit 2-interval graph if and only if the family $\mathcal{S}_{\mathcal{U}}(G)$ is not empty.*

*Proof.* Suppose that $G$ is a colored disjoint unit 2-interval graph with $V = V_{\texttt{white}} \cup V_{\texttt{black}}$. Then, by assumption, there exists a collection of disjoint unit 2-intervals $\mathbf{D}_{\texttt{white}} = \{(I_1(v), I_2(v)) \mid v \in V_{\texttt{white}}\}$ and a collection of unit intervals $\mathbf{I}_{\texttt{black}} = \{I_1(v) \mid v \in V_{\texttt{black}}\}$ such that $G \simeq \Omega(\mathbf{D}_{\texttt{white}} \cup \mathbf{I}_{\texttt{black}})$. Let $\mathcal{F}$ be the family of intervals formed by the underlying family of $\mathbf{D}_{\texttt{white}} \cup \mathbf{I}_{\texttt{black}}$. Let $S$ be the interval graph defined as the intersection graph of the family $\mathcal{F}$, i.e., $S \simeq \Omega(\mathcal{F})$. Consider the function $f : V(S) \mapsto V(G)$ such that:

- For every $I_1(v) \in \mathbf{I}_{\texttt{black}}$, $f(I_1(v)) = v$.
- For every pair $(I_1(v), I_2(v)) \in \mathbf{D}_{\texttt{white}}$, $f(I_1(v)) = f(I_2(v)) = v$.

By construction, $f$ satisfies all the conditions in Definition 6.1. Indeed, the first three conditions follow directly by definition, while the last two conditions follow because if we have an edge $(I_j(u), I_k(v))$ in $S$, for some $j, k \in \{1, 2\}$, this is equivalent to the 2-intervals associated to vertices $u$ and $v$ of $G$ intersecting, so there is an edge $(u, v)$ in $G$. Therefore, $(S, f)$ is a split of $(G, \gamma)$.

Conversely, suppose that there exists a disjoint split $(S, f)$ of $(G, \gamma)$ that satisfies the property of being a unit interval graph. Then, there exists a collection of unit intervals $\mathbf{I} = \{I_1(s) \mid s \in V(S)\}$ such that $S \simeq \Omega(\mathbf{I})$. Since $(S, f)$ is a split of $(G, \gamma)$, we know that there exists a map $f : V(S) \mapsto V(G)$ satisfying the conditions in Definition 6.1. We construct a colored disjoint unit 2-interval representation of $G$, i.e., a collection of disjoint unit 2-intervals $\mathbf{D}_{\texttt{white}} = \{(I_1(v), I_2(v)) \mid v \in V_{\texttt{white}}\}$ and a collection of unit intervals $\mathbf{I}_{\texttt{black}} = \{I_1(v) \mid v \in V_{\texttt{black}}\}$, as follows:

- For every $v \in V(G)$ with $\gamma(v) = \texttt{black}$, we let $I_1(v) = I_1(s)$, where $s = f^{-1}(v)$.
- For every $v \in V(G)$ with $\gamma(v) = \texttt{white}$, we let $I_1(v) = I_1(s)$ and $I_2(v) = I_1(t)$, where $\{s, t\} = f^{-1}(v)$.

By construction, this is a colored disjoint unit 2-interval representation of $G$, as the last two conditions of $f$ ensure that we preserve the same edges. □

We can now proceed to study the shape of the possible splits $(S, f) \in \mathcal{S}_{\mathcal{U}}(G_\Psi)$. Since all the splits considered in this section are disjoint, in the remaining we will refer to them simply as splits. Let $(S, f)$ be a split of a colored graph $G$. For every vertex $v \in V(G)$, we call each element of the set $f^{-1}(v)$ a representative of $v$. In particular, if $v$ is a white vertex, we denote its two representatives in $V(S)$ by $f_1^{-1}(v)$ and $f_2^{-1}(v)$. For simplicity, when we refer to an arbitrary representative of a vertex or to the unique representative of a black vertex, we abuse notation and denote it by its label in $V(G)$. Likewise, given an edge $(u, v) \in G$, we call the edge $(s, t) \in S$, a representative of $(u, v)$ if $s \in f^{-1}(u)$ and $t \in f^{-1}(v)$. Furthermore, given a split $(S, f)$ of the graph $G_\Psi$, we denote by $S[\hat{V}_i]$ the subgraph of $S$ induced by the vertices of the variable gadget $\hat{V}_i$ (i.e., vertices $A_i, B_i, C_i$, $f_1^{-1}(x_i^N)$, $f_1^{-1}(x_i^1)$, $f_1^{-1}(x_i^2)$, $f_2^{-1}(x_i^N)$, $f_2^{-1}(x_i^1)$ and $f_2^{-1}(x_i^2)$). Finally,

we say that a representative of a literal vertex is an *isolated vertex* if it is not adjacent to any of the private vertices of its variable gadget (i.e., it is not adjacent to $A_i, B_i$ or $C_i$).

**Lemma 6.3.** *Let $(S, f)$ be an arbitrary graph in $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Then, none of the black vertices of $S[\hat{V}_i]$ can be adjacent to both representatives of a literal vertex. Furthermore, if a black vertex is adjacent to a representative of $x_i^1$ and to a representative of $x_i^2$, these two representatives must be adjacent to each other.*

*Proof.* Suppose that the two representatives of a literal vertex are adjacent to the same black vertex. If the literal vertex is $x_i^1$ or $x_i^2$, the black vertex would be a center of a $K_{1,3}$ with these two representatives plus a representative of the vertex $x_i^N$ as leaves. If the literal vertex is $x_i^N$, the black vertex would be a center of a $K_{1,3}$ with the two representatives of $x_i^N$ and one of $x_i^1$ or $x_i^2$ as leaves. Since the graph $K_{1,3}$ is a forbidden induced subgraph for unit interval graphs, this contradicts the fact that $S$ belongs to $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Finally, if a black vertex is adjacent to a representative of $x_i^1$ and to a representative of $x_i^2$ which are not adjacent, the black vertex would be a center of a $K_{1,3}$ with these two representatives plus a representative of $x_i^N$ as leaves. □

From now on, since we know that $A_i$ is adjacent to a single representative of a literal vertex, we will denote by $f_1^{-1}(x_i^s)$ the representative that is adjacent to $A_i$, for $s \in \{1, 2, N\}$.

**Lemma 6.4.** *Let $(S, f)$ be an arbitrary split in $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Then, for every variable $x_i$ with $i \in \{1, \ldots, n\}$, the subgraph $S[\hat{V}_i]$ satisfies at least one of the following two conditions:*

1. *The vertex $f_1^{-1}(x_i^N)$ is adjacent to $A_i$ and the vertex $f_2^{-1}(x_i^N)$ is adjacent to $B_i$.*
2. *The vertices $f_1^{-1}(x_i^1)$ and $f_1^{-1}(x_i^2)$ are adjacent to each other and to $A_i$, and the vertices $f_2^{-1}(x_i^1)$ and $f_2^{-1}(x_i^2)$ are adjacent to each other and to $B_i$.*

*Proof.* By the properties of $f$, for every edge $(u, v) \in G_\Psi$, there exist elements $s, t \in V(S)$ with $f^{-1}(u) = s$ and $f^{-1}(v) = t$ such that $(s, t)$ is an edge in $S$.

Suppose condition 1 does not hold, i.e., $f_1^{-1}(x_i^N)$ is adjacent to both $A_i$ and $B_i$. We will show that if condition 2 does not hold either, $S$ cannot be a unit interval graph. Assume that one of the representatives of $x_i^1$ or $x_i^2$, $f_1^{-1}(x_i^1)$ (resp. $f_1^{-1}(x_i^2)$), is adjacent to both $A_i$ and $B_i$. Then, $S$ contains an induced cycle of length four: $\left(f_1^{-1}(x_i^N), B_i, f_1^{-1}(x_i^1), A_i\right)$ (resp. $\left(f_1^{-1}(x_i^N), B_i, f_1^{-1}(x_i^2), A_i\right)$). This is a forbidden induced subgraph for unit interval graphs, so it contradicts the hypothesis. Thus, it follows that vertices $f_1^{-1}(x_i^1)$ and $f_1^{-1}(x_i^2)$ need to be adjacent to $A_i$, and vertices $f_2^{-1}(x_i^1)$ and $f_2^{-1}(x_i^2)$, to $B_i$. Finally, by Lemma 6.3, $f_1^{-1}(x_i^1)$ and $f_1^{-1}(x_i^2)$ need to be adjacent to each other, so condition 2 must hold.

□

The previous lemma implies that there are four possible configuration of $S[\hat{V}_i]$ such that it does not contain any induced cycles of length greater or equal to 4.

**Lemma 6.5.** *Let $(S, f)$ be a split of $G_\Psi$ such that $S[\hat{V}_i]$ does not contain any induced cycles of length greater or equal to 4. Then, $S$ satisfies one of the following conditions:*

1. *The vertex $f_1^{-1}(x_i^N)$ is adjacent to $A_i$ and the vertex $f_2^{-1}(x_i^N)$ is adjacent to $B_i$, while for the rest of the literal vertices, there exists an element in the image via $f^{-1}$ that is an isolated vertex.*

2. *The vertices $f_1^{-1}(x_i^1)$ and $f_1^{-1}(x_i^2)$ are adjacent to each other and to $A_i$, and the vertices $f_2^{-1}(x_i^1)$ and $f_2^{-1}(x_i^2)$ are adjacent to each other and to $B_i$, while $f^{-1}(x_i^N)$ contains an isolated vertex.*

3. *The images of $x_i^1$ and $x_i^2$ via $f^{-1}$ are as in Case 2 and $f^{-1}(x_i^N)$ is as in Case 1 (see the graph in Figure 6.4).*

4. *Either the image of $x_i^1$ or the image of $x_i^2$ via $f^{-1}$ is as in Case 2 (w.l.o.g., assume it is $f^{-1}(x_i^1)$) so that both representatives of $x_i^1$ are adjacent to the non-isolated representative of $x_i^2$; and $f^{-1}(x_i^N)$ is as in Case 1.*

*Proof.* We have already shown that one of the conditions of Lemma 6.4 must hold. If condition 1 holds, then we have three possible configurations of $f^{-1}(x_i^1)$ and $f^{-1}(x_i^2)$: either both literal vertices have a representative that is isolated (Case 1), only one of them has a representative that is isolated (Case 4), or none of them has an isolated representative (Case 3). On the other hand, if condition 2 holds, then we only have two possible configurations of $f^{-1}(x_i^N)$: one representative of $x_i^N$ is isolated (Case 2), or none of them is (Case 3). Finally, note that in Case 4, both representatives of $x_i^1$ need to be adjacent to the non-isolated representative of $x_i^2$ by Lemma 6.3. □

The next two lemmas are devoted to proving that if $(S, f)$ is a split of $(G_\Psi, \gamma)$ contained in the family $\mathcal{S}_\mathcal{U}(G_\Psi)$, then Cases 3 and 4 of Lemma 6.5 are not possible. To do so, observe that by construction, since every variable appears exactly in three clauses (twice positive and once negated), we know that in $G_\Psi$, the vertices $x_i^1$, $x_i^2$ and $x_i^N$ all have two incident edges linking them with vertices outside of the variable gadget, called *external edges* in the following. The neighbors outside of the variable gadget are called *external vertices*, and they constitute private neighbors of the vertices of the variable gadget, as it is not possible for two different vertices of the variable gadget to be incident to the same external neighbor. We will see that if $S$ is as in Case 3 or Case 4, then the vertices of $S[\hat{V}_i]$ create an induced net with the external neighbors. Since nets are a forbidden induced subgraph for (unit) interval graphs, then $S$ cannot be a unit interval graph.

**Lemma 6.6.** *Let $S$ be an arbitrary graph in $\mathcal{S}_\mathcal{U}(G_\Psi)$. Then, for every variable $x_i$ with $i \in \{1, \ldots, n\}$, the subgraph $S[\hat{V}_i]$ cannot be as in Case 3 of Lemma 6.5.*

*Proof.* Suppose that $S[\hat{V}_i]$ is as in Case 3 of Lemma 6.5, i.e., as in Figure 6.4 (where $C_i$ could be in the neighborhood of the other representatives of the vertices, but thanks to the symmetry, these cases are equivalent). We distinguish two cases:

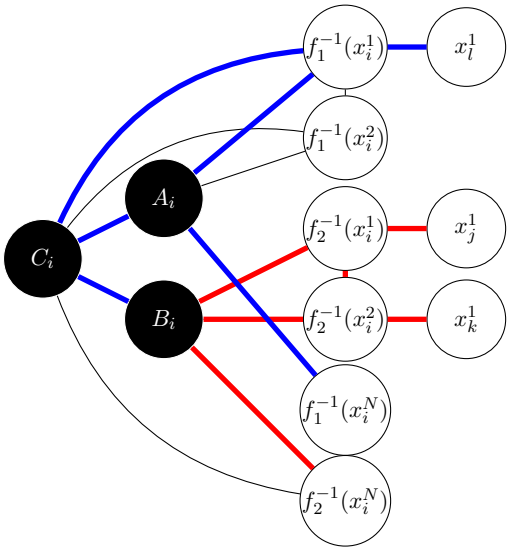



Figure 6.4.: Configuration of $S[\hat{V}_i]$ described in Case 3 of Lemma 6.5. In red, the net created if both $f_2^{-1}(x_i^1)$ and $f_2^{-1}(x_i^2)$ have an external neighbor. In blue, the net created if $f_1^{-1}(x_i^1)$ has an external neighbor.

- At least one of $f_1^{-1}(x_i^1)$ or $f_1^{-1}(x_i^2)$ is incident to an external edge. Then, $C_i, A_i, f_1^{-1}(x_i^1)$ or $C_i, A_i, f_1^{-1}(x_i^2)$ will create a net together with $B_i, f_1^{-1}(x_i^N)$, and the corresponding external neighbor of $f_1^{-1}(x_i^1)$ or $f_1^{-1}(x_i^2)$, respectively (see the blue net in fig. 6.4).
- Otherwise, the two external edges incident to $x_i^1$ and $x_i^2$ are incident to $f_2^{-1}(x_i^1)$ and $f_2^{-1}(x_i^2)$, respectively. Then, $f_2^{-1}(x_i^N), B_i, f_2^{-1}(x_i^1), f_2^{-1}(x_i^2)$, a private neighbor of $f_2^{-1}(x_i^1)$ and a private neighbor of $f_2^{-1}(x_i^2)$ form a net (see the red net in fig. 6.4).

In both cases, we have a forbidden induced subgraph for (unit) interval graphs, contradicting the hypothesis that $S$ is a unit interval graph. □

**Lemma 6.7.** *Let $(S, f)$ be an arbitrary split in $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Then, for every variable $x_i$ with $i \in \{1, \ldots, n\}$, the subgraph $S[\hat{V}_i]$ cannot be as in Case 4 of Lemma 6.5.*

*Proof.* Suppose that $S[\hat{V}_i]$ is as in Case 4 of Lemma 6.5, i.e., as in Figure 6.5. By construction, $x_i^2$ and at least one of $f_1^{-1}(x_i^1)$ or $f_2^{-1}(x_i^1)$ have an external neighbor. We distinguish two cases:

- The vertex $f_1^{-1}(x_i^1)$ has an external neighbor. Then, vertices $A_i, f_1^{-1}(x_i^1), x_i^2, f_1^{-1}(x_i^N)$, and the external neighbors of $x_i^2$ and $f_1^{-1}(x_i^1)$ form a net (see the red net in Figure 6.5).
- The vertex $f_2^{-1}(x_i^1)$ has an external neighbor. Then, vertices $B_i, f_2^{-1}(x_i^1), x_i^2, f_2^{-1}(x_i^N)$, and the external neighbors of $x_i^2$ and $f_2^{-1}(x_i^1)$ form a net (see the blue net in Figure 6.5).

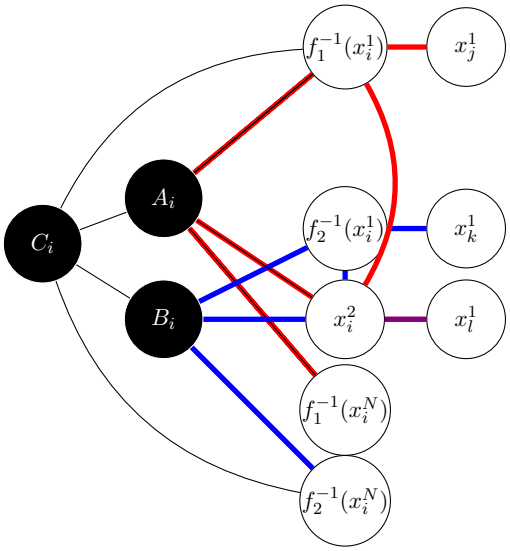



Figure 6.5.: Configuration of $S[\hat{V}_i]$ described in Case 4 of Lemma 6.5. In red, the net created if $f_1^{-1}(x_i^1)$ has an external neighbor, and in blue, the net created if $f_2^{-1}(x_i^1)$ has an external neighbor (edge $(x_i^2, x_l^1)$ is part of both nets and is depicted in purple).

In both cases, we have a forbidden induced subgraph for (unit) interval graphs, contradicting the hypothesis that $S$ is a unit interval graph. □

We have shown that only Cases 1 and 2 of Lemma 6.5 are possible. Recall that in Case 1, one of the representatives of $x_i^1$ and one of the representatives of $x_i^2$ are isolated; and in Case 2, one of the representatives of $x_i^N$ is isolated. Therefore, we obtain the following result.

**Lemma 6.8.** *Let $(S, f)$ be an arbitrary split in the family $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Then, for every variable $x_i$ with $i \in \{1, \ldots, n\}$, the subgraph $S[\hat{V}_i]$ satisfies exactly one of the following two conditions:*

1. *There is a representative of $x_i^1$ and a representative of $x_i^2$ that are isolated vertices (they are either two non-adjacent vertices or they form a $K_2$).*
2. *One of the representatives of $x_i^N$ is an isolated vertex.*

*Proof.* Combining Lemma 6.5 with Lemmas 6.6 and 6.7, it follows that $S[\hat{V}_i]$ is either as in Case 1 or as in Case 2 of Lemma 6.5, which means that either one representative of each of $x_i^1$ and $x_i^2$ is isolated, or that one representative of $x_i^N$ is isolated, respectively. These options correspond to the interval representations in Figure 6.6a and Figure 6.6b, respectively. The reader can check the previous assertion observing the figures, and verify that the external edges incident to each of the vertices $x_i^1, x_i^2$ and $x_i^N$ can be added in both representations, as we always have either a whole free interval (not depicted in the figures) or one extreme of the interval free for each of the vertices. □


(a) (b)

Figure 6.6.: Representation of the variable gadget associated to the true value (left, 6.6a) or false value (right, 6.6b).

The correctness of the reduction now follows from the two lemmas below.

**Lemma 6.9.** *If $\Psi$ is satisfiable, then the constructed graph $G_\Psi = (V, E)$, $V = V_{white} \cup V_{black}$, admits a colored disjoint unit 2-interval representation.*

*Proof.* Given a satisfying assignment $\Phi$ of $\Psi$, we explain how to construct a colored disjoint unit 2-interval representation of $G_\Psi$, i.e., a collection of disjoint unit 2-intervals $\mathbf{D}_{\texttt{white}} = \{(I_1(v), I_2(v)) \mid v \in V_{\texttt{white}}\}$ and a collection of unit intervals $\mathbf{I}_{\texttt{black}} = \{I_1(v) \mid v \in V_{\texttt{black}}\}$ such that $G \simeq \Omega(\mathbf{D}_{\texttt{white}} \cup \mathbf{I}_{\texttt{black}})$. To do so, we show how to construct a colored *proper* 2-interval representation, that is, a representation where no interval of the underlying family is properly contained in another one, which can then be transformed into a unit representation using the algorithm described in [21] Note that by Lemma 4.1, if $G_\Psi$ is a colored disjoint unit 2-interval graph, then there exists a split $(S, f)$ in the family $\mathcal{S}_{\mathcal{U}}(G_\Psi)$, and we know how to construct a colored disjoint unit 2-interval representation of $G_\Psi$ given a unit interval representation of $S$ by defining the 2-interval associated to a white vertex $v \in V_{\texttt{white}}$ as the union of the interval associated to $f_1^{-1}(v)$ and the interval associated to $f_2^{-1}(v)$; and the interval associated to a black vertex $v \in V_{\texttt{black}}$ as the interval associated to the single vertex $f^{-1}(v)$.

For each variable $x_i$ with $i \in \{1, \ldots, n\}$, if $\Phi(x_i) = \mathit{true}$, we represent the variable gadget $\hat{V}_i$ as shown in Figure 6.6a, which corresponds exactly to Case 1 of Lemma 6.8. On the other hand, if $\Phi(x_i) = \mathit{false}$, we represent $\hat{V}_i$ as in Figure 6.6b, which corresponds to Case 2 of Lemma 6.8. Notice that in both representations, the literals that are true have an isolated representative, i.e., one of the intervals associated to them is unused in the representation of $\hat{V}_i$ and remains completely free to display intersections with external neighbors.

After this, it only remains to explain the connections introduced by the clauses.

**Claim 6.1.** *Given a 3-clause $(x_i \vee x_j \vee x_k)$, there exists a unit interval representation of the subgraph of $G_\Psi$ induced by the vertices of the variable gadgets $\hat{V}_i, \hat{V}_j$ and $\hat{V}_k$.*

Proof. Each of the variable gadgets can be represented as in Figure 6.6a or Figure 6.6b. To represent the edges associated to the 3-clauses, we first notice that, since the 3-clauses are positive monotone, true literals correspond to true variables. As we are assuming that we have a satisfying assignment, we only have three cases (up to symmetry), which

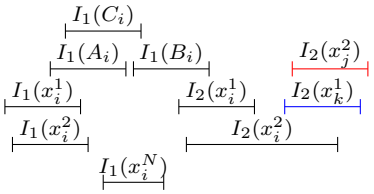



Figure 6.7.: Representation of a 3-clause $(x_i \vee x_j \vee x_k)$, where $x_i$ is set to false while $x_j, x_k$ are set to true.

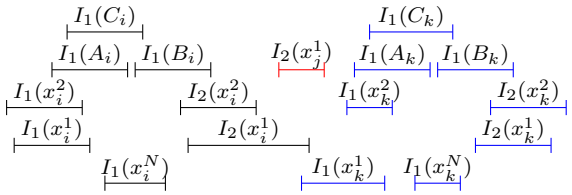



Figure 6.8.: Representation of a 3-clause $(x_i \vee x_j \vee x_k)$, where $x_i$ and $x_k$ are set to false and $x_j$ is set to true.

correspond to the three variables being true; exactly two variables being true; and only one variable being true. The literals that are true have a whole free interval to display the intersection, whereas the literals that are false only have the extreme of an interval (while the other extreme is glued to the rest of the representation of the gadget, see Figure 6.6b). Let $(x_i \vee x_j \vee x_k)$ be a 3-clause, with $i, j, k \in \{1, \ldots, n\}$. If the three variables are true, we can easily represent the clause by making the three free intervals of the variables – w.l.o.g. $I_2(x_i^1), I_2(x_j^1), I_2(x_k^1)$ – intersect at the same time. On the other hand, if only one variable – say $x_i$ – is false, we can add the two free intervals $-I_2(x_j^1), I_2(x_k^1)$ – to the corresponding extreme of the gadget of the false variable, as in Figure 6.7. Finally, if two variables are false – say $x_i, x_k$ –, then we need to merge the two interval representations associated to their gadgets and add the free interval – $I_2(x_j^1)$ – in the middle, as shown in Figure 6.8. Note that, as pointed out before, the interval representations given in the figures are not unit, but they are proper. $\lhd$

After representing all the 3-clauses, we can assume that the representations of some of the variable gadgets have been merged two by two (we will never have to merge a gadget more than once since a variable occurs in exactly one 3-clause in $\Psi$) and we can fix them all in the real line separated from one another. The separation between them can be arbitrarily large, and needs to be at least greater than the space needed to place the remaining intervals. The variable gadgets that have not been merged can also be fixed in the real line, while the unused free intervals (corresponding to true literals), the intervals $I_1(L_{i,j}^{\alpha})$, and the intervals $I_1(p_{i,j}^{\alpha})$ remain unplaced.

Now, to display the 2-clauses, we distinguish two cases. First, if both literals are true, then there exists a free interval for each, and we can represent the clause in a separate

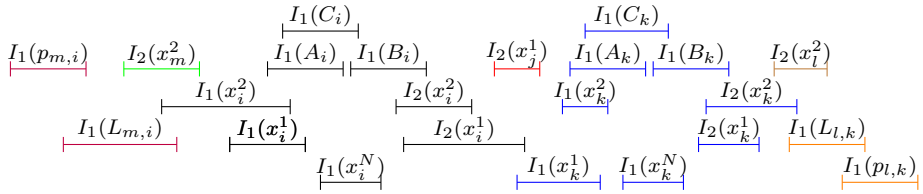



Figure 6.9.: Representation of a longest contiguous block of intervals, where each color represents the intervals associated to a different variable. A longest contiguous block occurs when there is a clause $(x_i \vee x_j \vee x_k)$, where $x_i$ and $x_k$ are set to false and both of them also appear as positive literals in a 2-clause.

part of the real line (there is one $L_{i,j}^{\alpha}$ and one $p_{i,j}^{\alpha}$ per clause, so these intervals will never cause a problem). Secondly, if one of the literals is false, then the free interval associated to the true literal needs to be glued to the extreme of the representation of the variable gadget of the false one. Note that there is always one free extreme because the 3-clauses use at most one extreme per variable gadget (and we can extend $I_j(x_i^2)$ to allow the intersection while keeping the representation proper). Note also that we will never need more than two extremes to obtain a representation because, since each variable occurs twice positive and once negated, we can have at most two false literals (when the variable is set to false).

Since we have constructed a proper interval representation, we can now use the algorithm described in [21] to turn the representation into a unit one, as explained before. □

Let us now prove the converse implication.

**Lemma 6.10.** *If the constructed graph $G_\Psi = (V, E)$, $V = V_{\texttt{white}} \cup V_{\texttt{black}}$, admits a colored disjoint unit 2-interval representation, then the original formula $\Psi$ is satisfiable.*

*Proof.* Assume that the constructed graph $G_\Psi$ admits a colored disjoint unit 2-interval representation where black vertices are represented by unit intervals and white vertices are represented by disjoint unit 2-intervals. As in Lemma 6.8, we study the splits $(S, f) \in \mathcal{S}_{\mathcal{U}}(G_\Psi)$.

We have already seen in Lemma 6.8 that there are only two possible configurations for $S[\hat{V}_i]$, up to symmetry. Let us assign a truth value to each of the configurations. If $S[\hat{V}_i]$ satisfies condition 1 of Lemma 6.8, we set $\Phi(x_i) = true$. Otherwise, if it satisfies condition 2 of Claim 6.8, then we set $\Phi(x_i) = false$. Recall that this implies that there is a representative of the vertices representing true literals which remains isolated from its variable gadget.

The following claims restrict the structure of a representable clause gadget. Both use similar arguments, so only the first proof is included here. Given a clause gadget $\hat{C}_\alpha$ in $G$, we define the clause gadget $S[\hat{C}_\alpha]$ in $S$ as the set of representatives of the edges and vertices of $\hat{C}_\alpha$.

**Claim 6.2.** *Let $(S, f)$ be an arbitrary split in $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Then, for every 3-clause, at least one of the representatives of the literal vertices incident to the clause gadget in $S$ must be an isolated vertex.*

Proof. Towards a contradiction, we assume that there exists a 3-clause gadget in $S$ such that none of the representatives of the literal vertices adjacent to the clause gadget are isolated. Let $C_\alpha = x_i \vee x_j \vee x_k$, with $i, j, k \in \{1, \ldots, n\}$ be a (monotone positive) 3-clause. Each of the literal vertices has two external neighbors. In $S$, either the two external neighbors are incident to the same representative of the literal vertices (and thus only one representative is incident to the clause gadget), or each of them is incident to a different representative. We distinguish two cases, depending on whether only one representative of each literal vertex is incident to the clause gadget, or whether there is at least one literal vertex such that both of its representatives are incident to the clause gadget:

- If only one representative of each literal vertex is incident to the clause gadget in $S$, then w.l.o.g., the clause gadget is formed by edges $\{(f_1^{-1}(x_i^1), f_1^{-1}(x_j^1)), (f_1^{-1}(x_j^1), f_1^{-1}(x_k^1)), (f_1^{-1}(x_k^1), f_1^{-1}(x_i^1))\}$. By assumption, none of the vertices incident to the clause gadget in $S$ are isolated, so they are all connected to at least one black vertex of their variable gadget. Thus, without loss of generality, $\{f_1^{-1}(x_i^1), f_1^{-1}(x_j^1), f_1^{-1}(x_k^1), A_i, A_j, A_k\}$ form a net (the readers can convince themselves looking at Figure A.1). Note that when we say without loss of generality, we are using the symmetry between $A_i$ and $B_i$.

- If there is at least one literal vertex such that both of its representatives are incident to the clause gadget, then w.l.o.g., the clause gadget in $S$ contains edges $\{(f_1^{-1}(x_i^1), f_1^{-1}(x_j^1)), (f_1^{-1}(x_k^1), f_2^{-1}(x_i^1))\}$ (and eventually, edges between representatives of $x_j^1$ and $x_k^1$). Then, since one of the representative of $x_i^2$ also has a private neighbor outside of the variable gadget, either the subgraph induced by $\{A_i, f_1^{-1}(x_i^1), f_1^{-1}(x_i^2)\}$ or the subgraph induced by $\{B_i, f_2^{-1}(x_i^1), f_2^{-1}(x_i^2)\}$ (and one private neighbor of each of the three vertices, where the private neighbor of $A_i$ and $B_i$ is $x_i^N$) is a net. This situation is depicted in Figure 6.10.

In both cases, the resulting graph $S$ would not be a unit interval graph, contradicting the hypothesis. ◁

**Claim 6.3.** *Let $(S, f)$ be an arbitrary split in $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Then, for every 2-clause, at least one of the representatives of the literal vertices incident to the clause gadget in $S$ must be an isolated vertex.*

Proof. Towards a contradiction, we assume that there exists a 2-clause gadget in $S$ such that none of the representatives of the literal vertices adjacent to the clause gadget are isolated. Let $C_\alpha = x_i^r \vee x_j^s$, with $i, j \in \{1, \ldots, n\}$, be a 2-clause, where the indices $r, s \in \{2, N\}$ indicate which occurrence of the variable appears in the clause. Again, there are two options:

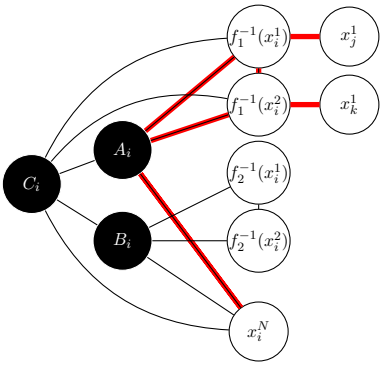



Figure 6.10.: In red, the net created if both representatives of $x_i^1$ are incident to the clause gadget in $S$ and $f_1^{-1}(x_i^2)$ is incident to an external edge.

- The clause gadget in $S$ forms a triangle, i.e., the two edges that are incident to a literal of a variable gadget are incident to the same representative. In this case, w.l.o.g., the clause gadget in $S$ comprises edges $\{(f_1^{-1}(x_i^r), f_1^{-1}(x_j^s)), (f_1^{-1}(x_j^s), L_{i,j}^\alpha), (L_{i,j}^\alpha, f_1^{-1}(x_i^r))\}$. Then, without loss of generality, we will have a net induced by $\{x_i^r, x_j^s, L_{i,j}^\alpha, A_i, A_j, p_{i,j}^\alpha\}$ (since $L_{i,j}^\alpha$ is black and $f^{-1}(L_{i,j}^\alpha)$ consists of a single element, this unique representative will be incident to the clause gadget and adjacent to $p_{i,j}^\alpha$ at the same time). The readers can convince themselves looking at Figure 6.3.
- Otherwise, there is a literal vertex such that both representatives are incident to an edge of the clause gadget. W.l.o.g., the clause gadget in $S$ contains edges $\{(f_1^{-1}(x_i^r), f_1^{-1}(x_j^s)), (L_{i,j}^\alpha, f_2^{-1}(x_i^r))\}$. Suppose first that $x_i^r$ is $x_i^2$. As in the case of 3-clauses, either the subgraph induced by $\{A_i, f_1^{-1}(x_i^1), f_1^{-1}(x_i^2)\}$ or by $\{B_i, f_2^{-1}(x_i^1), f_2^{-1}(x_i^2)\}$ (and one private neighbor of each of the three vertices, where the private neighbor of $A_i$ and $B_i$ is $x_i^N$) is a net. On the other hand, if $x_i^r$ is $x_i^N$, since this literal only occurs in 2-clauses and the vertex $L_{i,j}^\alpha$ for 2-clauses is black, then it cannot be the case that $f_1^{-1}(x_i^N)$ is adjacent to $x_j^s$, and $f_2^{-1}(x_i^N)$ is adjacent to $L_{i,j}^\alpha$. Indeed, if this happened, $L_{i,j}^\alpha$ would be the center of an induced $K_{1,3}$ with leaves $p_{i,j}^\alpha$, $f_2^{-1}(x_i^N)$, and a representative of $x_j^s$ (which is not adjacent to $f_2^{-1}(x_i^N)$ by assumption). This would contradict the fact that $S$ is a unit interval graph. The illustration of the $K_{1,3}$ created can be seen in Figure 6.3 removing the edge $(x_i^1, x_i^2)$, and replacing $x_i^1$ with $x_i^N$.

In both cases, the resulting graph $S$ would not be a unit interval graph, contradicting the hypothesis. ◁

The previous claims imply that there is an isolated literal vertex incident to every 3-clause and to every 2-clause. Since literal vertices that have an isolated representative correspond to true literals in the assignment fixed before, it follows that there is a true literal per clause, and thus, all clauses are satisfied. This finishes the proof of the converse direction. □

As the problem is clearly in NP, the polynomial-time construction together with Lemmas 6.9 and 6.10 conclude the proof of Theorem 6.1. The bound on the degree follows because the constructed graph $G$ has maximum degree 6 (the positive literal vertices have degree 4 in the variable gadget and are incident to 2 external edges).

### 6.2.2. Hardness of Disjoint Unit 2-Interval Recognition

Here, we show that COLORED DISJOINT UNIT 2-INTERVAL RECOGNITION is polynomial-time reducible to DISJOINT UNIT 2-INTERVAL RECOGNITION, which yields the main result of the section:

**Theorem 6.2.** *DISJOINT UNIT 2-INTERVAL RECOGNITION is* NP*-complete, even for graphs of degree at most 7.*

*Proof.* We reduce from COLORED DISJOINT UNIT 2-INTERVAL RECOGNITION, which is NP-hard by Theorem 6.1. Given any instance $(G, \gamma)$ of COLORED DISJOINT UNIT 2-INTERVAL RECOGNITION, where $G = (V, E)$ is a graph and $\gamma : V \to \{\texttt{white}, \texttt{black}\}$ is a vertex-coloring map, we construct an equivalent instance $G' = (V', E')$ of DISJOINT UNIT 2-INTERVAL RECOGNITION. Define $n = |V|$ and $V_{\texttt{c}} = \{u \mid u \in V \wedge \gamma(u) = \texttt{c}\}$ for $c \in \{\texttt{white}, \texttt{black}\}$ (so that $n = |V_{\texttt{white}}| + |V_{\texttt{black}}|$).

We obtain $G' = (V', E')$ from $G$ by replacing every vertex $v \in V_{\texttt{black}}$ by the gadget $B_v$ depicted in Figure 6.11, which we also call black vertex gadget. Formally, for every $v \in V_{\texttt{black}}$, we add the vertices $V_v = \{a_v^i, b_v^i \mid 0 \leqslant i \leqslant 3\}$ and the edges $E_v = \{(v, a_v^0), (a_v^0, a_v^i), (v, b_v^0), (b_v^0, b_v^i), (a_v^0, b_v^0) \mid 1 \leqslant i \leqslant 3\}$. The gadget $B_v$ is exactly the graph induced by the union of $V_v$ and vertex $v$. Note that the vertex $v$ of $B_v$ is *public*, that is, it is adjacent to vertices of $B_v$ and to vertices outside of $B_v$, while the rest of the vertices of $B_v$ are *private*, i.e., they are only adjacent to vertices of $B_v$.

We have thus constructed a graph $G'$ with vertex set $V' = V \cup \{V_v \mid v \in V_{\texttt{black}}\}$ and edge set $E' = E \cup \{E_v \mid v \in V_{\texttt{black}}\}$. Note that $G'$ contains $G$ as an induced subgraph, as $G'[V] = G$. Combining this with the replacement of every vertex in $V_{\texttt{black}}$ by a gadget with 9 vertices and 9 edges, it follows that $|V'| = |V_{\texttt{white}}| + 9\,|V_{\texttt{black}}|$ and $|E'| = |E| + 9\,|V_{\texttt{black}}|$.

The purpose of the black vertex gadget $B_v$ used to replace every $v \in V_{\texttt{black}}$ in the construction of $G'$ is to restrict the disjoint unit 2-interval representations of $G'$. Indeed, we will see that it forces one of the intervals associated to $v$ to be used exclusively to represent the gadget, while the other interval is used exclusively to represent the rest of the neighborhood of $v$ (which is exactly its neighborhood in the original graph $G$). Figure 6.12 shows a disjoint unit 2-interval representation $\mathbf{D}_{B_v} = \{(I_1(x), I_2(x)) \mid x \in V_v \cup \{v\}\}$ of $B_v$ such that $I_1(v)$ does not have any points in common with the rest of the intervals of $\mathbf{D}_{B_v}$ (i.e., only $I_2(v)$ is used to represent the gadget). Furthermore, in the given representation, $I_2(v)$ cannot intersect any interval associated to a vertex outside of the gadget, as there is no point of $I_2(v)$ that does not intersect either $I_1(a_v^0)$ or $I_1(b_v^0)$, and both $a_v^0$ and $b_v^0$ are private vertices for $v$. The next claim proves that any disjoint unit 2-interval representation of $B_v$ is as in Figure 6.12, up to symmetry.

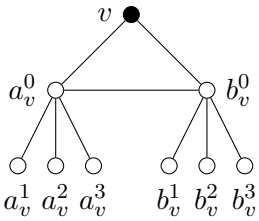



Figure 6.11.: Gadget $B_v$ used to replace every black vertex $v$ of $G$ in the construction of $G'$. Vertex $v$ is a *public* vertex, as it is adjacent to vertices of the gadget ($a_v^0$ and $b_v^0$) and vertices outside the gadget (namely, its neighbors in the original graph $G$), whereas the rest of the vertices are *private*, as their only neighbors are vertices from the gadget (the ones shown in the figure).

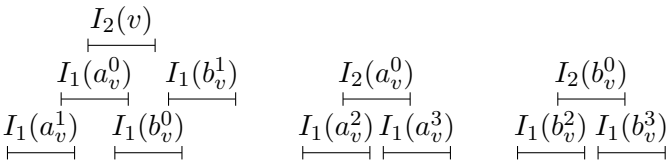



Figure 6.12.: A unit 2-interval representation of $B_v$ (Figure 6.11), i.e., $\mathbf{D}_{B_v}$ for an arbitrary $v \in V_{\texttt{black}}$. Note that only one interval of $v$ is used ($I_2(v)$), while the other one remains free to display the rest of the neighborhood of $v$ (and is not represented here).

**Claim 6.4.** *Let $\{(I_1(x), I_2(x)) \mid x \in V_v \cup \{v\}\}$ be a disjoint unit 2-interval representation of $B_v$. Then, there exist some indices $i, j, k \in \{1, 2\}$ such that the representation of $I_i(v), I_j(a_v^0), I_k(b_v^0)$ is continuous (i.e., the union of the three intervals is an interval) and $I_i(v)$ is properly contained in the union $I_j(a_v^0) \cup I_k(b_v^0)$.*

Proof. In the following, we denote an interval associated to a vertex by the name of the vertex if it refers to an arbitrary interval from the corresponding disjoint 2-interval (i.e., we will write $v$ to denote $I_1(v)$ or $I_2(v)$ when the choice of interval is irrelevant).

Since $a_v^0$ and $b_v^0$ are both centers of an induced $K_{1,4}$, one of the intervals associated to $a_v^0$, say $I_1(a_v^0)$, needs to intersect $v$, $b_v^0$ and one of the $a_v^i$ for some $i \in \{1, 2, 3\}$, say $a_v^1$ without loss of generality (because of the symmetry). Furthermore, the intervals of $v$ and $b_v^0$ that intersect $I_1(a_v^0)$ also need to intersect each other, as otherwise $I_1(a_v^0)$ would intersect three disjoint intervals, contradicting the fact that the representation is unit. On the other hand, $I_2(a_v^0)$ has to intersect the two remaining $a_v^i$, that is, $a_v^2$ and $a_v^3$. Similarly, one of the intervals associated to $b_v^0$, say $I_1(b_v^0)$, needs to intersect $v$ and $a_v^0$ (which also intersect each other), and one of the $b_v^i$ for some $i \in \{1, 2, 3\}$, whereas $I_2(b_v^0)$ intersects the two remaining $b_v^i$. Again, without loss of generality, we can assume that $I(b_v^0)$ intersects $b_v^1$ while $I_2(b_v^0)$ intersects $b_v^2$ and $b_v^3$.

Thus, we have that $I_1(a_v^0)$ intersects $v$ and $I_1(b_v^0)$ (which also intersect each other), and $a_v^1$; while $I(b_v^0)$ intersects $v$ and $I_1(a_v^0)$ (which also intersect each other), and $b_v^1$. This

implies that the representation of $a_v^1$, $I_1(a_v^0)$, $b_v^1$, $I_1(b_v^0)$ has to be contiguous. Finally, since vertex $v$ is not adjacent to either $a_v^1$ nor $b_v^1$, the only possibility to represent the edges $(v, a_v^0)$ and $(v, b_v^0)$ is by placing an interval associated to $v$, say $I_2(v)$, properly contained in the union $I_1(a_v^0) \cup I_1(b_v^0)$, as in Figure 6.12. ◁

The next two claims now prove the correctness of the reduction.

**Claim 6.5.** *If $G$ is a colored disjoint unit 2-interval graph, then $G'$ is a disjoint unit 2-interval graph.*

Proof. Suppose that $G$ is a colored disjoint unit 2-interval graph. Then, by assumption, there exists a collection of unit 2-intervals $\mathbf{D}_{\texttt{white}} = \{(I_1(v), I_2(v)) \mid v \in V_{\texttt{white}}\}$ and a collection of unit intervals $\mathbf{I}_{\texttt{black}} = \{I_1(v) \mid v \in V_{\texttt{black}}\}$ such that $G \simeq \Omega(\mathbf{D}_{\texttt{white}} \cup \mathbf{I}_{\texttt{black}})$.

From $\mathbf{D} = (\mathbf{D}_{\texttt{white}} \cup \mathbf{I}_{\texttt{black}})$, we show how to construct a disjoint unit 2-interval representation $\mathbf{D}'$ of $G'$. Recall that $(V_{\texttt{white}} \cup V_{\texttt{black}}) = V \subset V'$. Similarly, we will construct $\mathbf{D}'$ such that $\mathbf{D} \subset \mathbf{D}'$. In fact, we will have that $\mathbf{D}' = \mathbf{D} \cup (\bigcup_{v \in V_{\texttt{black}}} \mathbf{D}_{B_v})$, where for every $v \in V_{\texttt{black}}$, $\mathbf{D}_{B_v}$ is the interval representation of the gadget $B_v$. More precisely, we construct $\mathbf{D}'$ as follows:

- For every $v \in V_{\texttt{white}}$, we add to $\mathbf{D}'$ the 2-interval $(I_1(v), I_2(v))$ from $\mathbf{D}$.
- For every $v \in V_{\texttt{black}}$, we add to $\mathbf{D}'$ the interval $I_1(v)$ from $\mathbf{D}$ together with $\mathbf{D}_{B_v}$, i.e., the interval $I_2(v)$ plus the 2-intervals $(I_1(a_v^k), I_2(a_v^k))$ and $(I_1(b_v^k), I_2(b_v^k))$ for $0 \leqslant k \leqslant 3$ as defined in Figure 6.12.

By construction, $\mathbf{D}'$ is a collection of disjoint unit 2-intervals. It is now a simple matter to verify that $G' \simeq \Omega(\mathbf{D}')$. ◁

**Claim 6.6.** *If $G'$ is a dijsoint unit 2-interval graph, then $G$ is a colored disjoint unit 2-interval graph.*

Proof. Suppose that $G'$ is a disjoint unit 2-interval graph. Then, by assumption, there exists a collection of disjoint unit 2-intervals $\mathbf{D}' = \{(I_1(v), I_2(v)) \mid v \in V'\}$ such that $G' \simeq \Omega(\mathbf{D}')$. From $\mathbf{D}'$, we show how to construct a set $\mathbf{D}$ of $|V_{\texttt{white}}|$ disjoint unit 2-intervals and $|V_{\texttt{black}}|$ unit intervals.

Recall that $V \subset V'$. Similarly, we will take $\mathbf{D}$ to be a subset of $\mathbf{D}'$. Let $v \in V \subseteq V'$ be a vertex of $G'$. We distinguish two cases depending on the color of $v$ in $G$:

- $\gamma(v) = \texttt{white}$. We add to $\mathbf{D}$ the unit 2-interval $(I_1(v), I_2(v))$ of $\mathbf{D}'$.
- $\gamma(v) = \texttt{black}$. In $\mathbf{D}'$, we have a pair of intervals $(I_1(v), I_2(v))$. By Claim 6.4, w.l.o.g, $I_2(v)$ is used to display the gadget for black vertices, and cannot be used to represent any other edges. This means that all the remaining neighbors of $v$, which are exactly its neighbors in $G$, are displayed by $I_1(v)$. Thus, we add to $\mathbf{D}$ the unit interval $I_1(v)$ from $\mathbf{D}'$.

◁

As the problem is clearly in NP, combining the fact that the construction of $G'$ can be carried out in polynomial time with Claims 6.5 and 6.6, we obtain that Disjoint unit 2-Interval Recognition is NP-complete. The bound on the degree given in the statement of the theorem follows by the bound on the degrees given in Theorem 6.1, since only black vertices get two more neighbors (from adding the black vertex gadgets in Figure 6.11), while white vertices preserve their neighborhood. The bound on the degree given in the statement of the theorem follows from the bound on the degrees given in Theorem 6.1, since only black vertices get two more neighbors (from adding the black vertex gadgets in Figure 6.11), while white vertices preserve their neighborhood. □

### 6.2.3. Consequences and generalizations

We now generalize the result for disjoint unit $d$-interval graphs, with $d \geqslant 2$, which is not directly implied in graph recognition problems. We also extend the hardness result for some specific cases of disjoint unit $d$-interval graphs.

**Corollary 6.1.** *Recognizing disjoint unit $d$-interval graphs is* NP*-complete for every $d \geqslant 2$.*

*Proof.* We reduce recognition of disjoint unit $(d-1)$-interval graphs to recognition of disjoint unit $d$-interval graphs, hence the result holds by Theorem 6.2.

The idea is similar to the proof of Theorem 6.2. Given a graph $G = (V, E)$, we construct in polynomial-time a graph $G'$ by adding to each vertex a gadget similar to the one in Figure 6.11. Indeed, for every vertex $v$ in $G$, we create a triangle with vertices $v, a_v^0$ and $b_v^0$, but now $a_v^0$ and $b_v^0$ are adjacent to $2d-1$ independent vertices instead of just 3 (which is the case in Figure 6.11). Formally, for every $v \in V$, we add the vertices

$$V_v = \{a_v^i, b_v^i \mid 0 \leqslant i \leqslant 2d-1\}$$

and the edges

$$E_v = \{(v, a_v^0), (a_v^0, a_v^i), (v, b_v^0), (b_v^0, b_v^i), (a_v^0, b_v^0) \mid 1 \leqslant i \leqslant 2d-1\}$$

We now prove that $G$ has a disjoint unit $(d-1)$-interval representation if and only if $G'$ has a disjoint unit $d$-interval representation. First, given a unit $(d-1)$-interval representation of $G$, we can build a unit $d$-interval representation of $G'$ as in Figure 6.12. However, in this case, instead of having two intervals $I_1(v), I_2(v)$ associated to every vertex $v$, we have $d$ intervals, say $I_1(v), \ldots, I_d(v)$. W.l.o.g., the intervals $I_1(v), \ldots, I_{d-1}(v)$ are the $d-1$ intervals of the unit $(d-1)$-interval representation of $G$, while $I_d(v)$ plays the role of $I_2(v)$ in Figure 6.12. Similarly, $I_1(a_v^0)$ plays the role of $I_1(a_v^0)$ in Figure 6.12, while every $I_j(a_v^0)$, with $1 < j \leqslant d$ is represented as $I_2(a_v^0)$ in Figure 6.12, each intersecting two different $a_v^i$, with $1 < i \leqslant 2d-1$. The same holds for $b_v^0$.

For the converse implication, if we have a disjoint unit $d$-interval representation of $G'$, then, using the same argument as in the proof of Claim 6.4, we see that for every vertex $v$, we need to use a complete interval of $v$ to represent the gadget added in the construction of $G'$. Therefore, the remaining edges (which correspond exactly to the

edges of $G$), need to be displayed using only $d-1$ intervals associated to $v$. This implies that $G$ has a disjoint unit $(d-1)$-interval representation. □

**Corollary 6.2.** *Recognizing $(x, \ldots, x)$ $d$-interval graphs is* NP*-complete for every $x \geqslant 11$ and every $d \geqslant 2$.*

*Proof.* The result follows because the graph constructed in the reduction is a $(11, 11)$ 2-interval graph, and every $(11, 11)$ 2-interval graph is also a disjoint unit 2-interval graph (so the same reduction can be applied). To see this, the reader can verify the $(11, 11)$ 2-interval representation of the largest contiguous block in Figure 6.13, and check that the black vertex gadget used in the proof of Theorem 6.2 is also a $(11, 11)$ 2-interval graph.

To generalize for $d > 2$, it suffices to check that the gadgets added in the reduction of Corollary 6.1 are $(11, \ldots, 11)$ $d$-interval. Finally, as any $(x, \ldots, x)$ $d$-interval graph can be turned into a $(x+1, \ldots, x+1)$ $d$-interval graph (by partitioning the $dn$ intervals into the minimum number of maximal cliques and stretching the intersection of the intervals in each clique by one unit, as described in [96]), the graph constructed in the main reduction is an $(x, \ldots, x)$ $d$-interval graph for every $x \geqslant 11$. However, the graph constructed is not a $(x, \ldots, x)$ $d$-interval graph for any $x < 11$ (this has been checked with the help of an ILP solver [1]). □

**Corollary 6.3.** *Recognizing depth $r$ disjoint unit $d$-interval graphs is* NP*-complete for every $r \geqslant 4$ and every $d \geqslant 2$.*

*Proof.* The result follows because the depth of the representation constructed in the hardness proof of Theorem 6.1 is 4 (this can be verified by looking at Figure 6.9), and the depth of the representation of the black vertex gadget added in the proof of Theorem 6.2 is 3 (as can be seen in Figure 6.12). Furthermore, the gadgets added to prove the result for $d > 2$ have a representation of depth at most 3. The corollary generalizes for any depth $r > 4$, as for any $r > 4$, it is true that there exists a satisfying assignment if and only if the constructed graph $G'$ has a disjoint unit 2-interval representation of depth at most $r$. □

The following corollary is based on the Exponential Time Hypothesis (ETH).

**Corollary 6.4.** *Unless the ETH fails,* Disjoint Unit $d$-interval recognition *does not admit an algorithm with running time $2^{o(|V|+|E|)}$.*

*Proof.* We have provided a polynomial-time reduction from 3-SAT to Unit 2-interval recognition such that given an instance of 3-SAT of $n$ variables and $m$ clauses, it outputs an equivalent instance of Unit 2-interval recognition whose size is bounded by $O(n+m)$. Indeed, given an instance of 3-SAT of $n$ variables and $m$ clauses, we first build an equivalent instance of a special case of SAT with at most $3m$ variables and $7m$ clauses, and then an instance of colored disjoint unit 2-interval recognition with at most $18m$ vertices and $232m$ edges. Finally, to construct an equivalent instance

[1] `https://gist.github.com/fsikora/9c98e210af0c93487cc81e3d70891814`

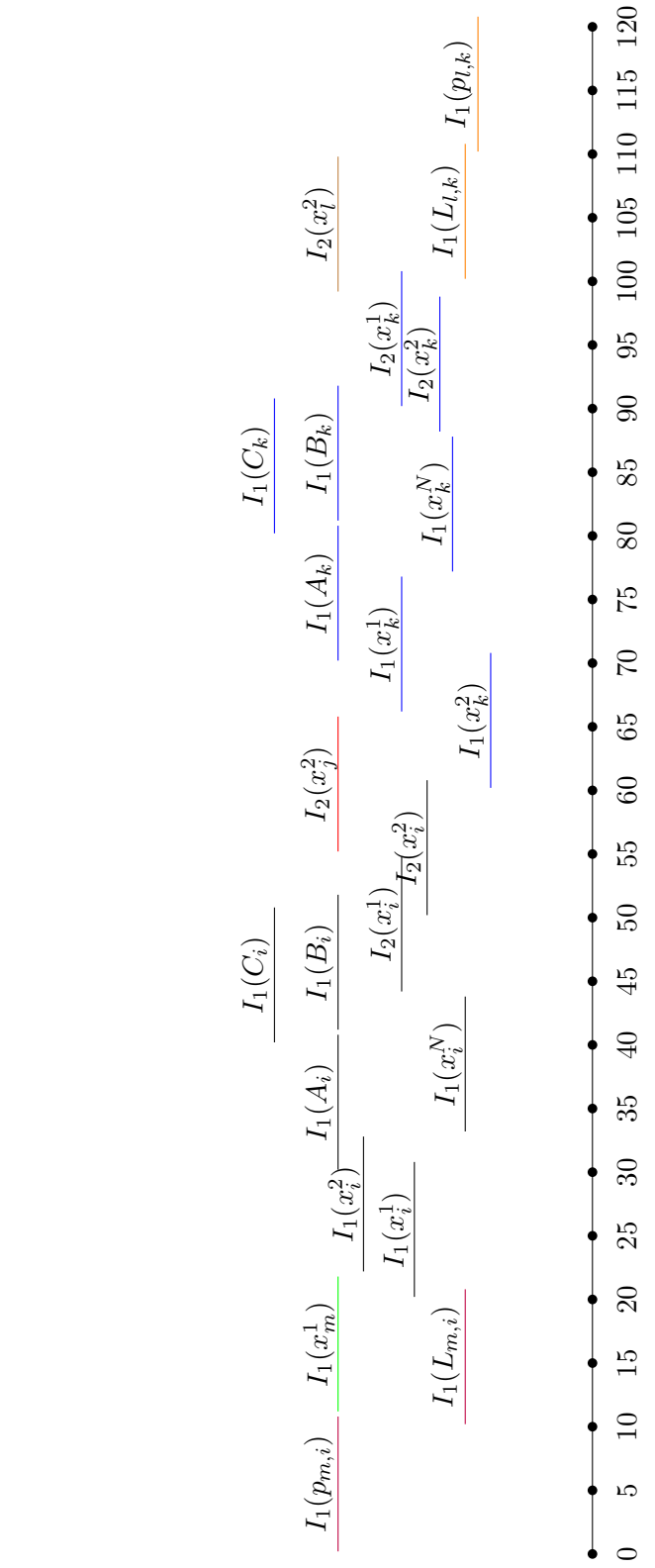



Figure 6.13.: An $(11, 11)$ 2-interval representation of a longest contiguous block of the graph constructed in the main reduction.

of Unit 2-interval recognition, we also add a linear number of vertices and edges (at most $9|V|$ and $9|E|$, respectively). Therefore, if Unit 2-interval recognition admitted an algorithm with running time $2^{o(|V|+|E|)}$, composing the reduction with such an algorithm would yield an algorithm for 3-SAT running in time $2^{o(n+m)}$, which would contradict the ETH. To generalize the result for $d > 2$, notice that the number of vertices and edges that we add in the proof of Corollary 6.1 is also linear. □

This implies that we cannot hope for a significantly better algorithm than the brute-force one for recognizing unit $d$-interval graphs, which has running time $O(2^{d^2 m})$ as shown in Theorem 4.1.

## 6.3. Hardness of recognizing (non-disjoint) unit $d$-interval graphs

In this section, we prove that recognizing unit $d$-interval graphs is also NP-complete by adapting the previous reduction. Recall that by unit $d$-interval graphs, we refer to the intersection graphs of *not necessarily disjoint* $d$-intervals.

We start by introducing a variation of the problem Colored unit disjoint 2-interval recognition, called Colored unit 2-interval recognition, where we drop the condition that the unit 2-interval that represents a white vertex must be disjoint. That is, given a graph $G = (V, E)$ and a coloring $\gamma : V \to \{\texttt{white}, \texttt{black}\}$, the task is to determine whether there exists a colored unit 2-interval representation of $(G, \gamma)$, i.e., a representation where every white vertex is represented by a (not necessarily disjoint) unit 2-interval, and every black vertex is represented by a single unit interval.

We can again characterize colored unit 2-interval graphs in terms of splits. We define a (non-disjoint) split of a colored graph $(G, \gamma)$ as a pair $(S, f)$ formed by a graph $S$ and a function $f : V(S) \to V(G)$ satisfying all the conditions of Definition 6.1 except the third one. That is, for every vertex $v \in V(G)$, $f^{-1}(v)$ is not necessarily an independent set in $S$. Similarly, we define the family of (non-disjoint) splits of a colored graph $G$ that lead to a unit interval graph as $\mathcal{S}^*_{\mathcal{U}}(G)$. We then obtain a characterization analogous to the one in Lemma 4.1.

**Lemma 6.11.** *A colored graph $(G, \gamma)$ is a colored unit 2-interval graph if and only if the family $\mathcal{S}^*_{\mathcal{U}}(G)$ is not empty.*

**Theorem 6.3.** *Deciding whether a graph is a colored unit 2-interval graph is* NP*-complete.*

*Proof.* We use the same reduction as in the proof of NP-hardness of recognizing colored disjoint unit 2-interval graphs in Theorem 6.1. The reader can observe that one of the directions of the proof still holds directly for the non-disjoint case: if the formula is satisfiable and there exists a colored disjoint unit 2-interval representation, then there also exists a colored unit 2-interval representation, as the class of disjoint unit 2-interval graphs is contained in the class of unit 2-interval graphs. Thus, Lemma 6.9 still holds for colored unit 2-interval graphs.

On the other hand, to prove the converse direction, we need to adapt the proof of Lemma 6.10, since it relies on case analysis where we have excluded the possibility that the vertices in $f^{-1}(v)$ are adjacent, for some white vertex $v$. Indeed, the fact that we can have an edge between the two representatives of a vertex invalidates the current proof of the first statement of Lemma 6.3, where it is shown that no black vertex can be adjacent to two representatives of the same literal vertex because it would form an induced $K_{1,3}$. This is no longer true in the non-disjoint case, as there could be an edge between the two literal vertices. However, the result still follows when we consider the whole construction.

**Claim 6.7.** *Let $(S, f)$ be an arbitrary split in $\mathcal{S}^*_{\mathcal{U}}(G_\Psi)$. Then, for every $i \in \{1, \ldots, n\}$, no black vertex of $\hat{V}_i$ can be adjacent to two representatives of the same vertex.*

Proof. Suppose that a black vertex, say $A_i$, is adjacent to two representatives of the same literal vertex. Then the two representatives must be adjacent to each other, and one of them needs to have a private neighbor in $G_\Psi$. This leads to two possible scenarios:

- There exists a representative that has a private neighbor and is adjacent to both $A_i$ and $B_i$. Then, it is the center of an induced $K_{1,3}$. Note that both representatives could be adjacent to the private neighbor outside the gadget, but it suffices that one satisfies the condition.

- There exists a representative of the literal vertex (w.l.o.g., we can assume that it is a vertex representing a positive literal) that has a private neighbor and is not adjacent to $B_i$. Then, the other representative must be adjacent to $B_i$ (and has no private neighbor, because otherwise we would be in the first case). Since we cannot have an induced $C_4$, vertices $A_i$ and $B_i$ must both be adjacent to different representatives of the negated literal vertex (this is proven in Lemma 6.4, which still holds in the non-disjoint case). Thus, there exists a net induced by the two representatives adjacent to $A_i$, the private neighbor, $A_i$, $B_i$, and the representative of the negated literal vertex adjacent to $A_i$ (see Figure 6.14).

◁

This shows that the statement of Lemma 6.3 still holds. We will now prove that if two representatives of the same literal vertex are adjacent to each other, then the constructed graph $G_\Psi$ cannot be a colored unit 2-interval graph. This claim implies that the rest of the proof of Lemma 6.10 holds in the non-disjoint case.

**Claim 6.8.** *Let $(S, f)$ be an arbitrary split in $\mathcal{S}^*_{\mathcal{U}}(G_\Psi)$. Then, no two representatives of the same literal vertex can be adjacent in $S$.*

Proof. Assume that both representatives of a same literal vertex are adjacent. Then, we can have two possible situtations:

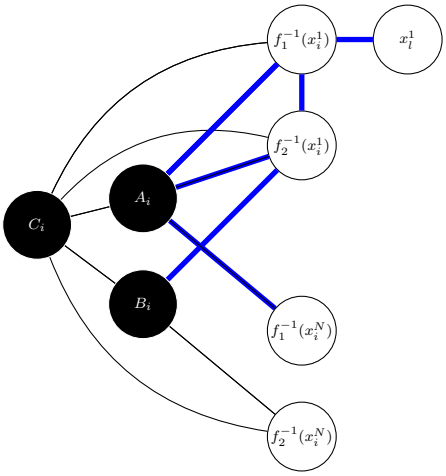



Figure 6.14.: If a black vertex is adjacent to two representatives of a same literal vertex (in this case, $A_i$ is adjacent to $f_1^{-1}(x_i^1)$ and $f_2^{-1}(x_i^1)$), then $S$ is not a unit interval graph (there is an induced net, highlighted in blue). The representatives of literal vertex $x_i^2$ are omitted for clarity.

- Both $A_i$ and $B_i$ are adjacent to the same representative of a literal vertex. Then, this representative induces a $K_{1,3}$ together with $A_i, B_i$ and the other representative, as we have just shown that neither $A_i$ nor $B_i$ can be adjacent to both representatives (see Claim 6.7).

- Vertex $A_i$ is adjacent to one representative and vertex $B_i$ is adjacent to the other representative. Assume, without loss of generality, that these representatives are $f_1^{-1}(x_i^1)$ and $f_2^{-1}(x_i^1)$. Then, since we are assuming that the two representatives are adjacent, vertex $C_i$ needs to be adjacent to both representatives. Indeed, if $C_i$ is not adjacent to any of them, then there is a cycle induced by the two representatives $f_1^{-1}(x_i^1)$ and $f_2^{-1}(x_i^1)$ together with the black vertices $C_i, A_i, B_i$. On the other hand, if $C_i$ is only adjacent to one of the representatives, then $C_i$ and one other black vertex (for example vertex $A_i$ if there was an edge between $C_i$ and the representative of the literal vertex adjacent to $B_i$) would induce a cycle together with the two representatives.

  We will now see that $C_i$ becomes the center of an induced $K_{1,3}$. First, observe that there cannot be a representative of $x_i^N$ adjacent to both $A_i$ and $B_i$, as then it would induce a cycle together with $f_1^{-1}(x_i^1)$ and $f_2^{-1}(x_i^1)$, $A_i$ and $B_i$. Thus, there is one representative adjacent to $A_i$, say $f_1^{-1}(x_i^N)$, and one representative adjacent to $B_i$, say $f_2^{-1}(x_i^N)$. Assume $C_i$ is adjacent to $f_1^{-1}(x_i^N)$ (it must be adjacent to at least one representative). Then, $C_i$ would be the center of an induced $K_{1,3}$ with leaves $f_1^{-1}(x_i^N)$, $B_i$ and the representative of $x_i^1$ adjacent to $A_i$. On the other hand, if $C_i$ were adjacent to $f_2^{-1}(x_i^N)$, the leaves of the $K_{1,3}$ would be $f_2^{-1}(x_i^N)$, $A_i$, and the representative of $x_i^1$ adjacent to $B_i$ (see Figure 6.15 for an illustration of this last situation).

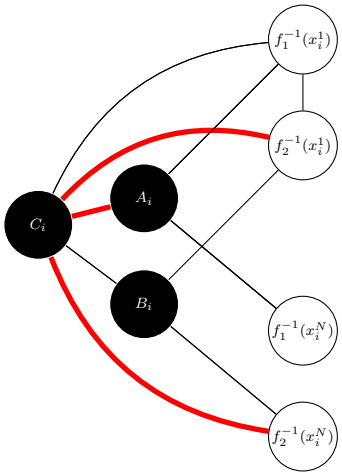



Figure 6.15.: Two representatives of the same literal vertex cannot be adjacent. In this case, there is an induced $K_{1,3}$ caused by the fact that $f_1^{-1}(x_i^1)$ and $f_2^{-1}(x_i^1)$ are adjacent. The representatives of literal vertex $x_i^2$ are omitted for clarity.

In both cases, $G_\Psi$ would not be a unit interval graph. ◁

Since both directions of the theorem hold, this concludes the proof. □

**Theorem 6.4.** *Unit d-interval recognition is* NP*-complete for every* $d \geqslant 2$*, even for graphs of depth at most* 4*.*

*Proof.* We use the same reductions as in Theorem 6.2 and Corollary 6.1. It is straightforward to see that both proofs generalize for the non-disjoint case. □

The lower bound under the ETH obtained in Corollary 6.4 also applies for Unit $d$-interval recognition.

## 6.4. Recognition of (disjoint) balanced d-interval graphs

In this section, we show that recognizing (disjoint and not necessarily disjoint) balanced $d$-interval graphs is also NP-complete. Gambette and Vialette [68] adapted the proof of West and Shmoys [148] to prove that recognizing disjoint balanced 2-interval graphs is NP-complete. Here, we adapt the reduction of West and Shmoys from the recognition of $(d-1)$-interval graphs to $d$-interval graphs [148, Theorem 2] to extend the result for every $d \geqslant 2$. In the previous chapter, we showed that the classes of balanced 2-interval and disjoint balanced 2-interval graphs are equivalent, but this is not the case for $d > 2$ (see Corollary 5.3). We prove that the construction given by West and Shmoys leads to a hardness result for recognizing both balanced and disjoint balanced $d$-interval graphs.

**Theorem 6.5.** *Recognizing (disjoint) balanced d-interval graphs is* NP*-complete for every* $d \geqslant 2$*.*

*Proof.* Given a graph $G$, we prove that the construction in [148, Theorem 2] yields a graph $G'$ that is balanced $d$-interval if and only if $G$ is balanced $(d-1)$-interval.

The graph $G'$ is constructed by creating three copies, $H_i(v)$ with $i \in \{1,2,3\}$, of the gadget $K_{d^2+d-1,d+1}$ for every vertex $v$; and connecting $v$ to a vertex of $H_2(v)$, and gadget $H_2(v)$ to both $H_1(v)$ and $H_3(v)$ by adding just two edges, each incident to a different gadget (see Figure 6.16 for an illustration). For a more detailed description, the reader is referred to [148]. Let $S_{d^2+d-1}$ and $S_{d+1}$ denote the two sides of the bipartition of the graph $K_{d^2+d-1,d+1}$. We denote the vertices of $S_{d^2+d-1}$ by $f^i, i \in \{1,\ldots,d^2+d-1\}$, and the vertices of $S_{d+1}$ by $v^i, i \in \{1,\ldots,d+1\}$. Given a $d$-interval representation of the complete bipartite graph, we refer to the $d$-interval associated to vertex $v^i$ (resp., $f^i$) as $(v^i_1,\ldots,v^i_d)$ (resp., $(f^i_1,\ldots,f^i_d)$).

We start by showing that the union of the three gadgets $H_i(v)$, $H_2(v)$ and $H_3(v)$ has a balanced $d$-interval representation. First, notice that each of the gadgets $H_i(v)$ has a balanced $d$-interval representation. Indeed, the complete bipartite graph $K_{d^2+d-1,d+1}$ has a balanced $d$-interval representation where every vertex of $S_{d^2+d-1}$ is represented by $d$ intervals of length 3 and every vertex of $S_{d+1}$ is represented by $d$ intervals of length $4(d-1)+3$. To construct it, place the intervals of $S_{d+1}$ in a line, leaving a space of one in between them, in the following order: $v^1_1, v^2_1, \ldots, v^d_1, v^1_2, v^2_2, \ldots, v^1_d, \ldots, v^{d+1}_d$. Then, in each of the holes of size one between two contiguous intervals, place an interval associated to a vertex of $S_{d^2+d-1}$. More precisely, place $f^1_1$ so that it intersects one unit of interval $v^1_1$ and one unit of interval $v^2_1$, and then continue placing the intervals $f^2_1,\ldots,f^{d^2+d-1}_1$ in order, one in each of the holes.

Now, the $d$-interval associated to a vertex of $S_{d+1}$ has intersected at least $2d-1$ intervals, each associated to a different vertex of $S_{d^2+d-1}$ (that is, $2d-1$ $d$-intervals). It still needs to intersect $d^2+d-1-(2d-1) = d-1$ $d$-intervals. Thus, it suffices to consider that the intervals associated to vertices of $S_{d^2+d-1}$ have length $4(d-1)+3$. That way, they can contain $d-1$ disjoint intervals of length 3 (with a space of one in between contiguous intervals), and they can also intersect by one unit an interval in each of their extremes. This implies that the whole gadget (the union of $H_2(v)$, $H_1(v)$ and $H_3(v)$) also has a balanced representation, as the edges between them are realized overlapping the intervals at the extremes of the $H_i(v)$'s.

Now, we want to show that if the input graph $G$ is balanced $d-1$ interval, the construction leads to a balanced $d$-interval graph. We will show that, for every vertex $v$ of the original graph, we can represent the intersections between $v$ and the gadget using a single interval associated to $v$, and given the length of this interval, the representation of the gadget can be modified so that it remains balanced. In the construction, vertex $v$ and $H_2(v)$ are connected by a single edge, say $(v, v^i)$. West and Shmoys proved that this edge must be represented by adding an interval associated to $v$ contained in a portion of the interval associated to $v^i$. Let $l(v)$ be the length of the intervals that comprise the $(d-1)$-interval associated to $v$ in the balanced $(d-1)$-interval representation of $G$. Then, it suffices to increase the length of the intervals associated to its neighbor in $H_2(v)$, $v^i$, by $l(v)+2$. This yields a balanced $d$-interval representation of the union of the gadget with vertex $v$. Doing this for every vertex of $G$ leads to a balanced $d$-interval

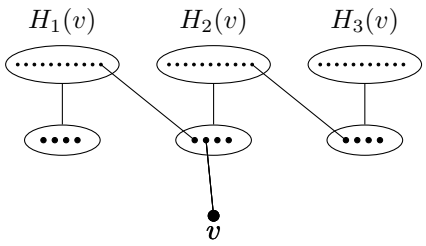



Figure 6.16.: Illustration of the gadget added to every vertex $v$ of the original graph for the case $d = 3$. $K_{11,4}$'s are represented schematically: an edge stopping at the border of an ellipse indicates that it is incident to every vertex within the ellipse.

representation of $G$.

Conversely, if the constructed graph $G'$ has a balanced $d$-interval representation, then the $d-1$ intervals used to represent the adjacencies in $G$ must have the same length. Since the adjacencies of every vertex with the gadget added are represented by a single disjoint interval, disregarding this interval and the $d$-intervals representing the gadgets, we obtain a balanced $(d-1)$-interval representation of $G$.

To show that the result holds also for disjoint balanced $d$-interval graphs, it suffices to observe that in any representation of $G'$, the new interval added associated to a vertex $v$ is disjoint from the other $d-1$ intervals, as it is properly contained in an interval associated to one of the vertices of the gadget added. □

The results of this chapter represent a significant step towards settling the landscape of the complexity of the recognition of the different subclasses of $d$-interval graphs. However, some questions still remain open. Since we have shown that recognizing depth 4 unit $d$-interval graphs is NP-complete and it is known that the recognition of depth 2 unit $d$-interval graphs is polynomial-time solvable [96], it still remains to delineate the exact boundary, i.e., study the case of depth 3 unit $d$-interval graphs. On the other hand, the complexity of recognizing $(x, \ldots, x)$ $d$-interval graphs for $2 < x < 11$ is also unknown (for $x = 1$, the class of graphs obtained is exactly line graphs, which can be recognized in polynomial time). Another interesting direction for future research is to study the fixed parameter tractability of the problem, parameterized by the number of vertices that are allowed to be represented by a 2-interval, and its approximability.

# 7. Proper interval completion on interval graphs

In this chapter, we study the problem of proper interval completion on interval graphs, which is a graph modification problem that consists on adding edges in order to make a given interval graph proper. We start with a brief overview of related modification problems in Section 7.1, and then present our results. In Section 7.2, we give a polynomial-time algorithm to solve the aforementioned problem on interval graphs that have a universal vertex and in some other specific cases. The algorithm is based on dynamic programming on the MPQ-tree of the input interval graph, and can be generalized to obtain an algorithm for arbitrary interval graphs, with the drawback of becoming exponential on the number of non-disjoint centers of maximal claws. In Section 7.3, we present this generalization.

## 7.1. Completion problems

Graph modification problems study how to transform an input graph into a graph that satisfies a particular property. They are fundamental in graph theory, and they have a large number of applications in different disciplines such as machine learning, molecular biology or numerical algebra [76, 134]. The operations allowed in order to transform the input graph can be diverse, but the most well-studied operations are arguably adding or/and deleting edges or/and vertices.

Here, we focus on the operation of adding edges. Such graph modification problems are referred to as *edge completion problems*. More precisely, given a property $\Pi$, a $\Pi$-*completion* of the graph $G = (V, E)$ is a supergraph $H = (V, E \cup F)$ such that $H$ satisfies property $\Pi$ and $E \cap F = \emptyset$. The edges in $F$ are called *fill edges*. A completion $H = (V, E \cup F)$ is *minimum* if for every supergraph $H' = (V, E \cup F')$ that satisfies $\Pi$, it holds that $|F| \leqslant |F'|$. The problem of computing such a completion is called the *minimum* $\Pi$-*completion* problem and its associated decision version consists on deciding whether there exists a $\Pi$-completion of $G$ with at most $k$ fill edges.

Often, the property $\Pi$ that we want to satisfy is that the graph belongs to a given class of graphs. Among such completion problems, three well-known examples are chordal completion, interval completion and proper interval completion, which consist on determining whether there exists a completion with at most $k$ fill edges that is chordal, interval or proper interval, respectively. These problems arise naturally in different domains: among other applications, chordal completion has been well studied due to its importance in sparse matrix computation [134], while interval and proper interval completion find applications in molecular biology [76].

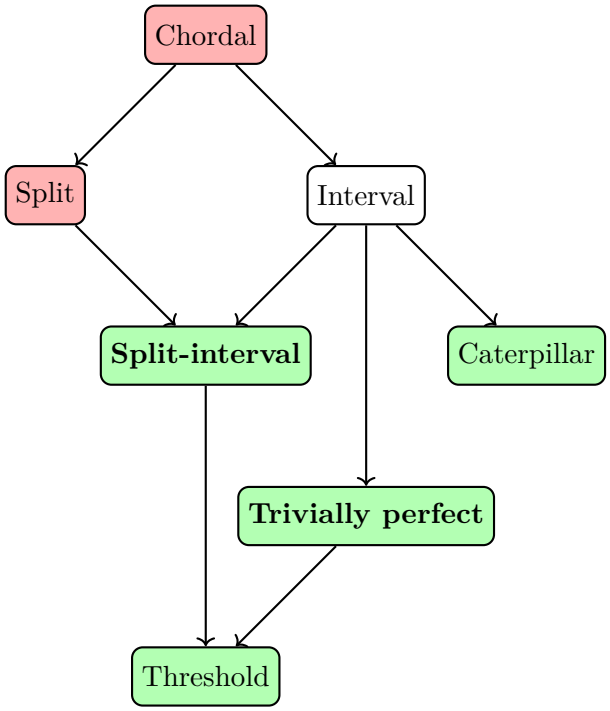



Figure 7.1.: Summary of the complexity of proper interval completion on different subclasses of chordal graphs. A rectangle in red indicates that the problem is NP-hard in this class, while a green rectangle means that it is polynomial. A white rectangle indicates that the complexity is unknown. In bold, the results proven here.

In particular, one of the most well known applications of proper interval completion is physical mapping of DNA. Given a set of clones (contiguous intervals of the DNA chain) and information on their pairwise overlap, the goal of this problem is to build a map describing the relative position of the clones. Since this information is obtained experimentally, there can be some errors, i.e., unidentified overlaps. If we model this as a graph where clones are represented by vertices and overlaps by edges, the problem of building a map assuming the fewest errors possible when all the clones have equal length is equivalent to proper interval graph completion.

Chordal, interval and proper interval completion are all known to be NP-complete [150, 99, 76]. Furthermore, Dross et al. showed that proper interval completion remains NP-complete when the input graph is a split graph, and becomes polynomial if it is a threshold graph or a caterpillar [54]. Since the problem is hard on chordal graphs (as it is hard on split graphs) and polynomial on some subclasses of interval graphs (threshold and caterpillar), a very natural question to pose is whether it remains hard when the input graph is an interval graph. The reminder of the chapter will be dedicated to addressing this question. Figure 7.1 illustrates the complexity of proper interval completion on different subclasses of chordal graphs.

The hardness of the aforementioned completion problems, together with the fact that optimal solutions can be found among inclusion-minimal completions, has also motivated the study of minimal chordal, interval and proper interval completion, that is, finding a completion such that the set of fill edges is minimal for inclusion, but not necessarily minimum. As opposed to minimum completions, these three problems are polynomial [135, 47, 129].

On the other hand, chordal, interval and proper interval completion have also been studied from the perspective of parameterized complexity, parameterized by the number $k$ of fill edges. The parameterized complexity approach is specially relevant in practical scenarios such as physical mapping of DNA, since usually the number of errors, which corresponds to $k$, is relatively small. Kaplan, Shamir and Tarjan showed that chordal completion and proper interval completion are both FPT parameterized by the number of fill edges [97]. The fixed-parameter tractability of interval completion took longer to be settled, and was finally proven by Villanger et al. [146]. Bliznets et al [18] presented the first subexponential parameterized algorithm for proper interval completion. As every FPT problem admits a kernel but not necessarily a polynomial kernel, the search for polynomial kernels arised as an interesting path of research. When the target class $\Pi$ is hereditary and can be characterized by a finite set of forbidden induced subgraphs, Cai showed that the corresponding graph modification problem is always FPT [32]. However, even in such cases, the problem might not admit a polynomial kernel [106]. Kaplan, Shamir and Tarjan proved in their seminal paper that chordal completion has a polynomial kernel [97]. Similarly, proper interval graph completion also admits a kernel with $O(k^3)$ vertices [17], but the kernelization status of interval completion remains an open question.

Here, we study Proper Interval Graph-completion (PIG-completion) on interval graphs. Note that before, proper interval completion referred to the problem with no restrictions on the input graph, but henceforth PIG-completion will always refer to the completion problem on interval graphs. Let us recall the problem formally.

Proper Interval Graph-completion

***Instance:*** An interval graph $G = (V, E)$.
***Goal:*** A set of edges $F$ such that:

- $E \cap F = \emptyset$.
- $H = (V, E \cup F)$ is a proper interval graph.
- The cardinality of $F$ is minimum among all such sets.

The completion $H$ will sometimes be called a *solution*, while the number of fill edges, $|F|$, will be referred to as the *cost* of the solution. Before presenting our results, let us go over some notation. Recall that in Section 3.2, we defined the MPQ-tree of an interval graph $G$ as the tree obtained from the PQ-tree of $G$ by assigning to a P-node $P$ the (possibly empty) set of vertices of $G$ contained in all the maximal cliques represented by

the subtree rooted at $P$ but which do not appear in any other cliques; and assigning to a Q-node $Q$ with children $t_1, \ldots, t_k$ ordered from left to right (recall that this ordering is unique up to reversal, so we fix one of the two possible orderings arbitrarily), a section $S_i$ for every $t_i$, $i \in \{1, \ldots, k\}$. Section $S_i$ contains the set of vertices that are contained in all maximal cliques of the subtree rooted at $t_i$ and in at least one other subtree rooted at some $t_j$ ($j \neq i$), but which do not appear in any clique belonging to some other subtree that is not below $Q$. Since a given vertex can only appear in consecutive sections, it suffices to represent it in the leftmost and rightmost sections in which it appears. We denote the occurrence of vertex $u_1$ in the leftmost section by $u_1^l$, and in the rightmost section by $u_1^r$.

Furthermore, recall that given an MPQ-tree $T$, we refer to the vertices of $T$ as nodes and we let $V(T)$ stand for the set of nodes of $T$, while the vertices of $G$ are referred to as vertices. We denote by $T_v$ the subtree of $T$ rooted at node $v$, and we define $T_{v^*}$ as the subtree obtained from $T_v$ by replacing node $v$ with a node containing the empty set (that is, we consider $T_v$ without the vertices contained in node $v$). Given a set of sections $S = \{S_i, \ldots, S_j\}$ of a Q-node with children $t_1, \ldots, t_i, \ldots, t_j, \ldots, t_k$, we also define the subtree rooted at $S$, $T_S$, as a Q-node with sections $S_i$ until $S_j$ and children $T_{t_i}$ until $T_{t_j}$ (in the same order). Given a subtree $T_v$, we denote the subgraph of $G$ induced by the vertices contained in the nodes of $T_v$ as $G[V(T_v)]$.

The size of node $v$, denoted by $|v|$, is defined as the number of vertices contained in node $v$; while $|T_v|$ stands for the total number of vertices contained in the nodes of $T_v$, i.e., $\sum_{x \in V(T_v)} |x|$; and $|T_{v^*}|$, for the number of vertices of $T_{v^*}$, i.e., $|T_v| - |v|$. On the other hand, given a set of nodes $B := \{b_1, \ldots, b_k\}$, we denote by $|T_B|$ the sum of the vertices in all subtrees, $\sum_{b_i \in B} |T_{b_i}|$.

Finally, given a Q-node $v$ with children $t_1, \ldots, t_k$ and with sections $S_1, \ldots, S_k$, in the ordering given by the Q-node, we define a partial ordering on the vertices contained in the nodes of $T_v$ as follows. Given two vertices $u_1$ and $u_2$, we say that $u_1 \prec u_2$ if one of the following holds:

- $u_1 \in T_{t_i}$ and $u_2 \in T_{t_j}$ with $i < j$.
- $u_1^r \in S_i$ and $u_2^l \in S_j$, with $i < j$.
- $u_1 \in T_{t_i}$ and $u_2^l \in S_j$ with $i < j$.
- $u_1^r \in S_i$ and $u_2 \in T_{t_j}$ with $i < j$.

## 7.2. A polynomial-time algorithm for interval graphs with a universal vertex

In this section, we study the problem PIG on interval graphs that have a universal vertex. The following observation yields light into the structure of a completion of such graphs.

**Observation 7.1.** *Let $G$ be an interval graph with a universal vertex and let $H$ be a PIG-completion of $G$. Then, the vertices of $H$ can be partitioned into two sets $C_1$ and $C_2$ such that each set induces a clique in $H$.*

*Proof.* Suppose otherwise. Then, there must exist three pairwise non-adjacent vertices $x, y$ and $z$ in $H$. But then, the universal vertex of $G$ (which is also universal in $H$) would be the center of an induced $K_{1,3}$ in $H$ with leaves $x$, $y$ and $z$. Since the $K_{1,3}$ is a forbidden induced subgraph for proper interval graphs, we have reached a contradiction. □

Given a PIG-completion $H$ of $G = (V, E)$, we say that it *partitions* $V$ into $(C_1, C_2)$ if $C_1$ and $C_2$ form a partition of $V$ and each set induces a clique in $H$. Note that $C_1$ and $C_2$ are not necessarily unique. We denote by $C_1$ the smallest of such sets. Conversely, given a partition $(C_1, C_2)$ of $V$, we refer to the graph $H$ obtained by adding edges so that each of $C_1$ and $C_2$ induce a clique as the *completion associated* to $(C_1, C_2)$, and we say that the partition $(C_1, C_2)$ *yields* completion $H$.

However, note that not every partition yields a valid PIG-completion. Indeed, notice that Observation 7.1 also holds when the input graph is restricted to the class of threshold graphs [54], but unlike for threshold graphs, PIG-completion on interval graphs is not equivalent to minimum co-bipartite completion (that is, the problem of adding as few edges as possible in order to make the graph co-bipartite). This was already shown by Dross et al., who provided a quasi-threshold graph where a minimum co-bipartite completion returns a graph with an induced $C_4$, and so it is not a proper interval completion (see Figure 2 of [54]). Since the class of quasi-threshold graphs is a proper subclass of the class of interval graphs with a universal vertex, minimum co-bipartite completion is also not equivalent to minimum proper interval completion on interval graphs with a universal vertex.

Thus, in our setting, PIG-completion is a special case of co-bipartite completion where we also need to forbid cycles of length greater than three (note that none of the other induced fobidden subgraphs of proper interval graphs can be present if we have a completion to co-bipartite). In the following, we prove a necessary and sufficient condition to obtain a valid partition of the vertices (that is, a partition that yields a PIG-completion). This condition gives enough structure to develop a dynamic programming algorithm on the MPQ-tree of the input graph.

**Lemma 7.1.** *Let $G = (V, E)$ be an interval graph with a universal vertex and let $T$ be its associated MPQ-tree. Let $(C_1, C_2)$ be a partition of $V$. Let $w$ be an arbitrary P-node of $T$ and let $b_1, \ldots, b_k$ be the children of $w$ and $T_B := \{T_{b_1}, \ldots, T_{b_k}\}$ the set of subtrees rooted at each of the children. Then, partition $(C_1, C_2)$ yields a completion without induced cycles of length greater than 3 if and only if the following three conditions are satisfied:*

- *For every P-node $w$, there exists at most one subtree in the set $T_B$ that contains a pair of vertices $v_1, v_2$ such that $v_1$ belongs to $C_1$ and $v_2$ to $C_2$.*
- *For every Q-node $q$, there do not exist two sections $S_i$ and $S_j$ with children $t_i$ and $t_j$ such that both children are non-empty and $T_{t_i}$ contains a pair of vertices $v_1, v_2$*

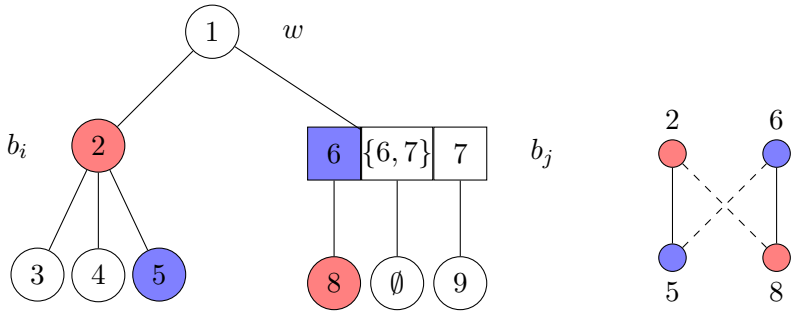



Figure 7.2.: If a node $w$ has two children $b_i$ and $b_j$ such that $T_{b_i}$ and $T_{b_j}$ both contain a pair of vertices with one vertex in $C_1$ and the other in $C_2$, the partition yields a completion with an induced $C_4$. A node in red indicates that the vertex contained inside belongs to $C_1$, while a node in blue indicates that it belongs to $C_2$. To the right of the MPQ-tree we depict the induced $C_4$ in the completion: full edges represent edges already present in $G$ while dashed edges represent fill edges.

*with $v_1 \in C_1$ and $v_2 \in C_2$, while subtree $T_{t_j}$ contains a pair of vertices $u_1, u_2$ with $u_1 \in C_1$ and $u_2 \in C_2$.*

- *For every Q-node $q$, there do not exist two sections $S_i$ and $S_j$ such that the subtree $T_{S_i}$ contains a pair of vertices $v_1, v_2$ with $v_1 \in C_1$ and $v_2 \in C_2$ and the subtree $T_{S_j}$ contains a pair of vertices $u_1, u_2$ with $u_1 \in C_1$ and $u_2 \in C_2$, with $v_1 \prec u_2$ and $v_2 \prec u_1$. Notice that if neither vertex is contained in a section, this condition is equivalent to the second one.*

*Proof.* Let us first prove that the conditions are necessary. Towards a contradiction, assume the first condition does not hold. Fix a P-node $w$ and let $b_1$ and $b_2$ be two children of $T_w$ such that both $T_{b_1}$ and $T_{b_2}$ contain a pair of vertices satisfying the condition that one vertex is in $C_1$ and the other in $C_2$. Note that the root of subtree $T_{b_1}$ is not empty (that is, it contains at least one vertex) because it is a child of a P-node. Thus, it is clear that there is either one node of $T_{b_1}$ that contains a vertex $u \in C_1$ and a vertex $v \in C_2$ or there is a pair of nodes in $T_{b_1}$ such that one is the ancestor of the other and one of the nodes contains a vertex $u \in C_1$ and the other contains a vertex $v \in C_2$ (otherwise, every vertex contained in node $b_1$ would be in the same side of the partition, say $C_1$, and every other vertex of $T_{b_1}$ would be contained in a node which is a descendant of $b_1$, so it would also be in $C_1$). In both cases, we have that the vertices $u$ and $v$ are adjacent in the graph $G$. By the same reasoning, there exist two vertices $u' \in C_1$ and $v' \in C_2$ contained in the nodes of $T_{b_2}$ that are adjacent in $G$.

Now, by the definition of MPQ-tree, there cannot exist an edge in the graph $G$ between a vertex contained in a node of the subtree $T_{b_1}$ and a vertex contained in a node of $T_{b_2}$. Thus, since $u$ and $v'$ (resp., $v$ and $u'$) are each in a different subtree and belong to different sides of the partition, they will not be adjacent in the PIG-completion associated to

partition $(C_1, C_2)$. On the other hand, since $u$ and $u'$ (resp., $v$ and $v'$) are in the same side of the partition, they will be adjacent in the PIG-completion. Therefore, in the completion, we will only have the edges $(u, v)$ and $(u', v')$ of $G$ plus the added edges $(u, u')$ and $(v, v')$. But then, the set of vertices $\{u, v, v', u'\}$ forms an induced cycle of length four, which contradicts the fact that the solution is a proper interval graph (see Figure 7.2).

The proof that the second condition is necessary is analogous.

Next, assume that the third condition doesn't hold. Without loss of generality (by symmetry), assume that $v_1$ is in $S_i$. Then, edge $(v_1, v_2)$ is present in graph $G$. On the other hand, if $u_1$ and $u_2$ are not adjacent in $G$, then by the same argument as before, there must exist two vertices $u'_1 \in C_1$ and $u'_2 \in C_2$ contained in the subtree rooted at the child of $S_j$. Furthermore, these two vertices also satisfy the property that $v_1 \prec u'_2$ and $v_2 \prec u'_1$, which means that edges $(v_1, u'_2)$ and $(v_2, u'_1)$ are not present in $G$. But then, in the completion we only have edges $(v_1, v_2), (v_2, u'_2), (u'_1, u'_2), (v_1, u'_1)$, which means that there is an induced $C_4$. Note that if $u_1$ and $u_2$ were already adjacent in $G$, we replace $u'_1$ and $u'_2$ by them.

To prove the converse direction, assume that the completion contains an induced cycle of length greater than three. First of all, it is clear that the completion associated to partition $(C_1, C_2)$ cannot contain an induced cycle of length greater than four, since it must admit a partition of the vertex set into two cliques. Thus, suppose that there exists a $C_4$ induced by vertices $v_1, v_2, v_3$ and $v_4$ with edges $(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1)$, and suppose that $v_1$ and $v_2$ are in $C_1$ and $v_3$ and $v_4$ are in $C_2$, w.l.o.g.. Since $v_2$ and $v_3$ are adjacent in the completion and in different sides of the partition, they must have been adjacent in $G$, which implies that they are both in the same node of the MPQ-tree, or one is in a node that is an ancestor of the other. The same holds for $v_1$ and $v_4$. On the other hand, the pair $v_1$ and $v_3$ and the pair $v_2$ and $v_4$ are not adjacent in the completion, which means that they were also not adjacent in $G$. Since $v_2$ and $v_3$ are adjacent in $G$, then they are either on the same node or in nodes that are in an ancestor/descendant relationship. But then, in $G$, $v_1$ cannot be in a node that is an ancestor of $v_2$ (as otherwise the edge $(v_1, v_3)$ would also be present in $G$), nor in a node which is a descendant of the node that contains $v_2$ (as otherwise edge $(v_2, v_4)$ would be present in $G$). Thus, there are two possibilities:

- There exists a node of the MPQ-tree with two children $u$ and $w$ such that $v_1$ and $v_4$ are contained in $T_u$, while $v_2$ and $v_3$ are contained in $T_w$ (violating the first or the second condition).
- There exists a section $S_i$ such that $T_{S_i}$ contains $v_1$ and $v_4$ and a section $S_j$ such that $T_{S_j}$ contains $v_2$ or $v_3$, and it holds that $v_1 \prec v_3$ and $v_4 \prec v_2$, or the other way around (violating the third condition).

This analysis concludes the proof. □

The previous lemma gives the key of the dynamic programming algorithm that we will devise: for each node $v$ of the MPQ-tree of the input graph, we will "guess" the

subtree that can be *split* (that is, have vertices in both sides of the partition associated to the completion). Then, for every $r \in \{0, \ldots, \lfloor |T_v|/2 \rfloor + 1\}$, we compute the minimum cost of a completion associated to a partition having $r$ vertices in the smallest set when we restrict the input to the subgraph of $G$ induced by the vertices contained in $T_v$.

**Algorithm** Let $G$ be an interval graph with a universal vertex. Given the MPQ-tree $T$ of $G$, we compute the optimal cost of a PIG-completion of $G$ in a bottom-up fashion, by solving a subproblem for each node. In the following, we write $[k]$ to denote the set of integers $\{1, \ldots, k\}$ and we let $G[V(T_v)]$ be the subgraph of $G$ induced by the vertices contained in subtree $T_v$.

We consider the following family of subproblems: for every node $v \in V(T)$ and every $r \in [\lfloor |T_v|/2 \rfloor + 1]$, we define:

$$\begin{aligned} M_{v,r} = &\min\{|F| : F \text{ is a set of fill edges of } G[V(T_v)] \text{ that yields a partition} \\ &(C_1, C_2) \text{ with } |C_1| = r\} \end{aligned}$$

Then, the completion of $G$ will be given by the minimum over all the values of $r$ of $M_{root,r}$, where *root* is the root of $T$. In the following, we describe how to compute these subproblems in detail.

**Base case: leaves.** For every leaf $v$ of $T$,

$$M_{v,r} = \begin{cases} 0 & r \leqslant |v|, \\ +\infty & \text{otherwise.} \end{cases}$$

This follows because the vertices of node $v$ all form a clique (by definition of MPQ-tree), and so in any solution that partitions the vertices of node $v$ into two cliques, the cost of the solution is zero. On the other hand, having a partition where the smallest clique has size $r$ is not feasible if we are considering a subgraph with less than $r$ vertices. Unfeasibility is represented by the value $+\infty$.

**Step for P-nodes.** Let $v$ be a P-node with $k+1$ children. For every child $u$ of $v$, we denote its siblings by $b_1, \ldots, b_k$, and we define the set $B_i := \{u, b_1, \ldots, b_i\}$ for every $i \in [k]$ (with $B_0 := \{u\}$). For every child $u$ and every $r \in [\lfloor |T_v^*|/2 \rfloor + 1]$, we define the following auxiliary subproblem:

$$\begin{aligned} W_{u,i,r} = &\min\{|F| : F \text{ is a set of fill edges of } G[V(T_{B_i})] \text{ that yields a partition} \\ &(C_1, C_2) \text{ with } |C_1| = r \text{ and where only subtree } T_u \text{ is split } \} \end{aligned}$$

The high level idea is that given a P-node, we will guess which of the children can be split into the two cliques (unique by Lemma 7.1). Recall that we say that a subtree $T_w$ is split into the two cliques if there exists a pair of vertices $u, v$ contained in the nodes of $T_w$ with $u \in C_1$ and $v \in C_2$.

- **Base case.** The base case of these subproblems is when $i = 0$. In that case, we set
$$W_{u,0,r} := M_{u,r}$$
Indeed, $W_{u,0,r}$ represents the cost of a solution of $G[T_u]$ that yields a partition with $|C_1| = r$, which is exactly what we compute in the original subproblem.

- **Step.** For every $1 \leqslant i \leqslant k$, we compute $W_{u,i,r}$ recursively. That is, given the costs of the completions of $G[V(T_{B_{i-1}})]$ which split only subtree $T_u$ (recall that we have one cost per $r$), we compute the cost when we add the vertices contained in subtree $T_{b_i}$. To do so, we determine the optimal way of adding $T_{b_i}$ to the existing solution by computing the minimum between the cost of adding it to the smallest clique and to the largest one.
  - If $T_{b_i}$ is added to the largest clique and $|T_{B_{i-1}}| - r \geqslant 0$, then the cost is $W_{u,i-1,r} + (|T_{B_{i-1}}| - r) \cdot |T_{b_i}| + M_{b_i,0}$. That is, we take the cost of having $r$ vertices in the smallest clique (computed in the previous iteration), and we add the cost of adding the new subtree to the largest clique. Since in iteration $i-1$ we have added all the subtrees in the set $B_{i-1}$, this cost is exactly $(|T_{B_{i-1}}| - r) \cdot |T_{b_i}|$ plus the cost of making $T_{b_i}$ a clique, which is $M_{b_i,0}$. If $|T_{B_{i-1}}| - r < 0$, the cost is set to infinity, since adding $b_i$ to the largest clique would result in the smallest clique having size less than $r$, and so there is no feasible solution for the value of $r$ considered.
  - If $T_{b_i}$ is added to the smallest clique and $r - |T_{b_i}| \geqslant 0$, then the cost is $W_{u,i-1,r-|T_{b_i}|} + (r - |T_{b_i}|) \cdot |T_{b_i}| + M_{b_i,0}$. That is, we consider the cost of a solution with the smallest clique having size $r - |T_{b_i}|$ (computed in the previous iteration) and we add the cost of adding all the edges between the new subtree considered, $T_{b_i}$, and the $r - |T_{b_i}|$ vertices already in the smallest clique. On the other hand, if $r - |T_{b_i}| < 0$, the cost remains infinity, as there is no feasible solution (again, we cannot place all the vertices of the subtree in the smallest clique under the constraint that the smallest clique has size $r$).

  Then, for every $i \in [k]$ and for every $r$, we set $W_{u,i,r}$ to be the minimum between these two costs or $+\infty$ if none of them exist for the value of $r$ considered. That is, for every $i \in [k]$,
$$\begin{aligned} W_{u,i,r} = \min\{&W_{u,i-1,r} + (|T_{B_{i-1}}| - r) \cdot |T_{b_i}| + M_{b_i,0}, \\ &W_{u,i-1,r-|T_{b_i}|} + (r - |T_{b_i}|) \cdot |T_{b_i}| + M_{b_i,0}\} \end{aligned}$$

We can now solve our original subproblem as follows:

$$\begin{aligned} M_{v*,r} &= \min\{W_{u,k,r} \mid u \text{ child of } v\} \\ M_{v,r} &= \min\{M_{v*,r-i} \mid i \in [|v|]\} \end{aligned}$$

In other words, we first compute the value $M_{v*,r}$, which is defined as the minimum over every child $u$ of $v$ of $W_{u,k,r}$, and corresponds to the optimal cost of a completion of $G[V(T_{v*})]$ that yields a partition with $|C_1| = r$. Once we have the optimal cost of a solution for the subtree $T_{v*}$, we incorporate the vertices in node $v$ to the solution and get $M_{v,r}$. Since these vertices are adjacent to every other vertex in the subtree, adding them to either clique will have a cost of zero. The correctness of the step for P-nodes follows from the next lemma.

**Lemma 7.2.** *Let $v$ be a P-node and let $u$ be the child of $v$ such that $T_u$ is allowed to split. Let $b_1, \ldots, b_k$ be the siblings of $u$. Then, the values $M_{v*,r}$ give the optimal cost of a completion of $T_v*$ where the smallest clique has size $r$.*

*Proof.* The solution is feasible because it satisfies Lemma 7.1. Now suppose we have an optimal solution. Without loss of generality, suppose that in the solution, all the vertices of the subtrees $T_{b_1}, \ldots, T_{b_l}$, for some $1 \leqslant l \leqslant k$ are placed in the clique $C_1$ and the rest of the vertices are placed in the clique $C_2$. Then, the total cost of this solution, $W_{u,k,r}$, is equal to

$$\sum_{i=1}^{i=k} M_{b_i,0} + W_{u,0,r-\sum_{i=1}^{i=l}|T_{b_i}|} + \sum_{i=1}^{i=l}(r - \sum_{j=i}^{j=l}|T_{b_j}|)|T_{b_i}| + \sum_{i=l+1}^{i=k}|T_{b_i}|(\sum_{j=i+1}^{j=k}|T_{b_j}|)$$

The expression of the total cost, obtained by summing the previously described costs and substituting at each step the value of $W_{u,i-1,r}$ by the cost computed in the previous iteration, does not depend on the order in which we add the subtrees. Finally, since at each iteration we compute the cost of having $r$ vertices in the smallest clique for every $r$, the value obtained at the end is the minimum cost. □

**Step for Q-nodes.** Let $v$ be a Q-node with children $t_1, \ldots, t_k$ and with sections $S_1, \ldots, S_k$, ordered from left to right in the ordering given by the Q-node. Since the order of the subtrees is fixed up to reversal, in a Q-node we do not need to guess the subtree that is allowed to split, as we can greedily compute the cost of having exactly $r$ vertices in the smallest clique. Indeed, to do so, we will go through an intermediate step, where instead of $C_1$ and $C_2$, we consider a left-clique and a right-clique (again, note that in some sense, this notion of left and right is artificial, as the Q-node gives a unique order up to reversal), and we compute the costs of having $r$ vertices in the left clique. That is, for the Q-node $v$, we define an auxiliary family of subproblems, where for every $r \in [|T_v|]$, we introduce:

$$W_r = \min\{|F| : F \text{ is a set of fill edges of } G[V(T_v)] \text{ that yields a partition with the left clique having size } r\}$$

To compute these costs, we first introduce a left cost and a right cost for each vertex, which represent the number of vertices strictly to their left/right. These costs are monotone, in the sense that if we have a pair of vertices $v_1$ and $v_2$ with $v_1 \prec v_2$, then the left cost of $v_1$ will be smaller than the left-cost of $v_2$, and the right cost of $v_2$ will be greater

than that of $v_1$. Monotonicity will guarantee that we can compute the optimal cost of having $r$ vertices in the left clique in a greedy-like manner without creating forbidden induced subgraphs. Formally, for every subtree $T_{t_i}$ rooted at a child of the Q-node $v$, we compute the left cost (resp. right cost) of $T_{t_i}$ as follows:

- The left cost of a subtree $T_{t_i}$, $l(T_{t_i})$, is defined as the sum over all subtrees $T_{t_j}$ with $j < i$ of the number of vertices of $T_{t_j}$ plus the sum over all the sections $S_j$ with $j < i$ of all the vertices that are contained in $S_j$ but not in $S_i$. That is,

$$l(T_{t_i}) := \sum_{j<i} \left( |T_{t_j}| + \sum_{x^r \in S_j} 1 \right) \tag{7.1}$$

- Similarly, the right cost of a subtree $T_{t_i}$, $r(T_{t_i})$ is the sum over all subtrees $T_{t_j}$ with $j > i$ of the number of vertices of $T_{t_j}$ plus the sum over all the sections $S_j$ with $j > i$ of all the vertices contained in $S_j$ but not in $S_i$. That is,

$$r(T_{t_i}) := \sum_{j>i} \left( |T_{t_j}| + \sum_{x^l \in S_j} 1 \right) \tag{7.2}$$

The left (resp. right) cost of a vertex $q$ contained in a section of the Q-node $v$ is defined similarly:

- To define the left cost of a vertex $q$, we let $S_i$ be the section containing $q^l$. Then, the cost $l(q)$ is equivalent to the left cost of subtree $T_i$:

$$l(q) := l(T_{t_i}), \text{ for } i \text{ such that } q^l \in S_i \tag{7.3}$$

- Similarly, to compute the right cost of $q$, we let $S_i$ be the section containing $q^r$ and let $r(q) = r(T_i)$.

$$r(q) := r(T_{t_i}), \text{ for } i \text{ such that } q^r \in S_i \tag{7.4}$$

The high level idea to compute $W_r$ is the following. We start with all the vertices in the right clique, and then, we compute $W_r$ by iteratively swapping the $r$ vertices with the smallest "swapping cost". The cost of swapping a vertex $q$ of a section is equal to $l(q) - r(q)$, while the cost of swapping a vertex $x$ of a subtree $T_{t_i}$ assuming that we have already swapped $k$ vertices of $T_{t_i}$ is equal to $M_{t_i,k+1} - M_{t_i,k} + l(T_{t_i}) - r(T_{t_i})$ for $0 \leqslant k \leqslant |T_{t_i}| - 1$ (in this case, we also need to account for the cost of splitting $T_{t_i}$ into two cliques, since only $k+1$ of its vertices are in the left clique). Note that the cost of swapping can be negative. Formally, we define a function $update(x, k)$ which returns either the cost of swapping $k+1$ vertices of subtree $x$ given that we have already swapped $k$, or the cost of swapping vertex $x$. If a vertex of a section or all the vertices of a subtree have already been swapped, we set this value to $+\infty$ to ensure that we never choose it

again:

$$update(x,k) := \begin{cases} l(x) - r(x) & \text{if } x \text{ is a vertex and } k = 0, \\ M_{x,k+1} - M_{x,k} & \\ \quad +l(x) - r(x) & \text{if } x \text{ is a subtree and } 0 \leqslant k \leqslant |x| - 1, \\ +\infty & \text{otherwise.} \end{cases} \tag{7.5}$$

Additionally, for every $r$ and every $x$ in the union of the sets of subtrees and vertices of a section, we define:

$$\begin{aligned} c_{r,x} &:= \text{ cost of swapping a vertex of } x \text{ given that the left clique has size } r \\ u_{r,x} &:= \text{ number of vertices of } x \text{ swapped given that the left clique has size } r \end{aligned}$$

Note that when $x$ is a vertex, *vertex of* $x$ refers simply to $x$, and then the only possible value of $u_{r,x}$ is 1. Now, we compute $W_r$ as follows:

- **Base case.** If $r = 0$,

$$\begin{cases} W_0 = \sum_i M_{t_i,0} + r(T_{t_i}) \cdot |T_{t_i}| + \sum_{x^l \in S_i} r(x) \\ c_{0,x} = update(x,0),\ \forall x \\ u_{0,x} = 0,\ \forall x \end{cases} \tag{7.6}$$

  Indeed, when we consider a partition where every vertex is in the clique of the right, for every subtree $T_{t_i}$ we need to account for the cost of having all the vertices of $T_{t_i}$ in the same clique plus the cost of adding all the edges between these vertices and every vertex to its right; and for every vertex in a section, we have a cost of $r(q_i)$.

- **Step.** If $r > 0$,

$$\begin{cases} x & = \arg\min_x c_{r-1,x} \\ W_r & = W_{r-1} + c_{r-1,x},\ r > 0 \\ u_{r,x} & = u_{r-1,x} + 1 \text{ and } u_{r,y} = u_{r-1,y},\ \forall y \neq x \\ c_{r,x} & = update(x, u_{r,x}) \text{ and } c_{r,y} = c_{r-1,y},\ \forall y \neq x \end{cases} \tag{7.7}$$

  That is, we iteratively swap the vertex with smallest cost: we choose the vertex or subtree $x$ with smallest cost, add it to the left-clique and update the cost $c_{r,x}$ of swapping a vertex of $x$ and the number $u_{r,x}$ of vertices of $x$ swapped.

The correctness of this step follows from the next lemma.

**Lemma 7.3.** *For every $r$, the solution computed is feasible and optimal.*

*Proof.* We first prove that the costs are monotone, i.e., given two vertices $u_1 \prec u_2$ in the partial order defined before, the cost of swapping $u_1$ is always smaller than the cost

of swapping $u_2$. This ensures that when we choose to swap a vertex $x$, we have already swapped every vertex strictly to its left (i.e., every vertex $y$ such that $y \prec x$), and so Lemma 7.1 is satisfied. Let $x, y$ be two vertices such that $y \prec x$. we distinguish different cases:

- Vertex $y$ belongs to a section $S_i$, while $x$ belongs to a section $S_j$ with $i < j$. Then, by the definitions of the left and right costs, we have that $l(y) < l(x)$ and $r(y) > r(x)$, so $-r(x) + l(x) > -r(y) + l(y)$, i.e., the cost of swapping $x$ is strictly greater than the cost of swapping $y$.
- Vertex $y$ belongs to $T_{t_i}$ and $x$ belongs to $T_{t_j}$, with $i < j$. Then the cost of swapping $y$ is $M_{t_i,k} - M_{t_i,k-1} + l(T_{t_i}) - r(T_{t_i})$. But $l(T_{t_j}) > l(T_{t_i}) + M_{t_i,r+1} - M_{t_i,r}$ for any $r$, as $M_{t_i,r+1} - M_{t_i,r}$ is upper bounded by $|T_{t_i}|$. On the other hand, by definition, $r(t_j) > r(t_i)$, so combining the inequalities we obtain that the cost of swapping $x$ is strictly greater than the cost of swapping $y$.

The cases when one of the vertices is in a subtree and the other is in a section are analogous. The optimality follows because the cost of swapping a vertex is constant (note that when we update the costs of the subtrees, it is actually the cost of a different vertex in the subtree). □

The solution of our original subproblem $M_{v,r}$ (i.e., the cost of having $r$ vertices in the smallest clique) is given by the minimum between the cost of having $r$ vertices in the left clique and the cost of having $r$ vertices in the right clique. That is, for every $r \in \left[\left\lfloor \frac{|T_v|}{2} \right\rfloor + 1\right]$,

$$M_{v,r} = \min\{W_r, W_{|T_v|-r}\}$$

Finally, as we anticipated before, the final solution of the problem is given by

$$\min_{r \in [|T_{root}|]} M_{root,r}$$

To sum up, we write the algorithm in a compact manner:
pt

**Algorithm**

For every node $v$ of the MPQ-tree and every value $r \in [\lfloor |T_v|/2 \rfloor + 1]$, we define:

$M_{v,r} = \min\{|F| : F$ is a set of fill edges of $G[V(T_v)]$ that yields a partition of the subgraph into two cliques with the smallest one having size $r\}$

- **Base case**: If $v$ is a leaf,

$$M_{v,r} = \begin{cases} 0 & r \leqslant |v| \\ +\infty & \text{otherwise} \end{cases}$$

- **Step if $v$ is a P-node**. Let $v$ be a P-node with $k+1$ children. For every $i \in [k]$, every $r \in [\lfloor |T_v^*|/2 \rfloor + 1]$, and every children $u$ of of $v$ with siblings $b_1 \ldots, b_k$, we define an auxiliary subproblems:

$$W_{u,i,r} = \begin{cases} M_{u,r} & \text{if } i = 0, \\ \min \Big\{ W_{u,i-1,r} + (|T_{B_{i-1}}| - r) \cdot |T_{b_i}| + M_{b_i,0}, & \\ \qquad W_{u,i-1,r-|T_{b_i}|} + (r - |T_{b_i}|) \cdot |T_{b_i}| + M_{b_i,0} \Big\} & \text{if } i > 0. \end{cases}$$

  The recurrence for the original subproblem is then given by :

$$\begin{aligned} M_{v*,r} &= \min\{W_{u,k,r} \mid u \text{ child of } v\} \\ M_{v,r} &= \min\{M_{v*,r-i} \mid i \in [|v|]\} \end{aligned}$$

- **Step if $v$ is a Q-node**. Let $v$ be a Q-node with children $t_1, \ldots, t_m$ and with sections $S_1, \ldots, S_m$. For every $r$ in $[|T_v|]$, we define an auxiliary subproblem:
  - Base case:

$$\begin{cases} W_0 = \sum_i M_{t_i,0} + r(T_{t_i}) \cdot |T_{t_i}| + \sum_{x^l \in S_i} r(x) \\ c_{0,x} = update(x, 0), \forall x \\ u_{0,x} = 0 \, \forall x \end{cases}$$

  - Step:

$$\begin{cases} x & = \arg\min_x c_{r-1,x} \\ W_r & = W_{r-1} + c_{r-1,x},\ r > 0 \\ u_{r,x} & = u_{r-1,x} + 1 \text{ and } u_{r,y} = u_{r-1,y},\ \forall y \neq x \\ c_{r,x} & = update(x, u_{r,x}) \text{ and } c_{r,y} = c_{r-1,y},\ \forall y \neq x \end{cases}$$

  Then, the recurrence for the original subproblem is given by:

$$M_{v,r} = min\{W_r, W_{|T_v|-r}\}$$

This concludes the presentation of the algorithm and the proof of correctness, which implies the following theorem.

**Theorem 7.1.** *Let $G$ be an interval graph with a universal vertex. Then, there exists an $O(n^3)$ algorithm that computes the PIG-completion of $G$.*

The complexity claimed in the statement of the theorem follows because for every node $v$ of the tree, in the worst case, to compute the array $M_{v,r}$ we need to iterate over the children of $v$ twice. Since we need to compute $M_{v,r}$ for every $r$ and both $r$ and the number of children of $v$ are upper-bounded by $n = |V(G)|$, the complexity of the algorithm is $O(n^3)$.

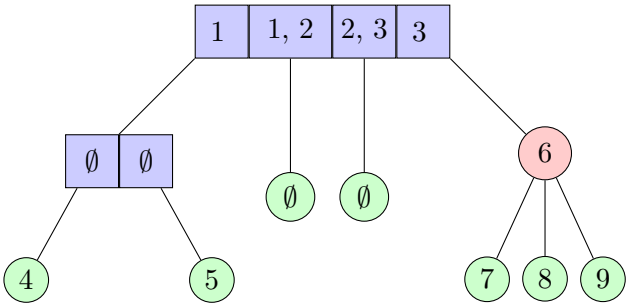



Figure 7.3.: MPQ-tree of an interval graph. Each section of a Q-node is represented by a blue square, P-nodes are represented by red circles and leaves are represented by green circles. Vertices 1 and 3 are disjoint maximal centers, while vertex 6 is a center but not a maximal one. For an example of two intersecting centers, the reader is referred to Figure 7.5.

**Corollary 7.1.** *PIG-completion is polynomial-time solvable when the input graph is restricted to the class of trivially perfect graphs.*

*Proof.* Any connected trivially perfect graph contains a universal vertex, so we can apply the previous algorithm to every connected component. □

In the remainder of this section, we extend the previous positive result to a more general setting, namely the case of interval graphs that do not contain intersecting centers of claws. Let us formally define what we mean by *intersecting centers*. In the following, given an interval graph $G$, we say that a vertex $v$ is a maximal center if it is a center of a claw and $v$ is contained in some node $w$ of the MPQ-tree of $G$ such that every section or P-node which is an ascendant of $w$ contains the empty set.

**Definition 7.1.** *Let $u$ and $v$ be two vertices of $G$ that are maximal centers and let $T$ be the MPQ-tree of $G$. We say that the centers $u$ and $v$ are* disjoint *if one of the following holds:*

- *The subtrees of $T$ rooted at the nodes that contain vertices $u$ and $v$ are disjoint (they do not share any nodes).*
- *Vertices $u$ and $v$ belong to the same Q-node but there does not exist a section that contains both vertices simultaneously (equivalently, $u^r \prec v^l$ or $v^r \prec u^l$).*

Figure 7.3 illustrates the MPQ-tree of an interval graph with two disjoint centers (the associated interval representation is given in Figure 7.4).

**Corollary 7.2.** *Let $G = (V, E)$ be an interval graph and let $T$ be its associated MPQ-tree. Suppose that for every pair of vertices $u, v \in V$ such that $u$ and $v$ are both maximal centers in $G$, centers $u$ and $v$ are disjoint. Then, we can compute a PIG-completion of $G$ in time $O(n^3)$.*

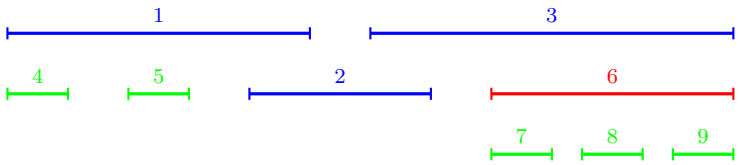



Figure 7.4.: Interval representation of the graph represented by the MPQ-tree on Figure 7.3.

*Proof.* Let $u$ be a maximal center. If $u$ is contained in a P-node $p$, we consider the subtree rooted at $p$ and apply the previous algorithm to $T_p$ in order to obtain a completion of the graph induced by the vertices contained in $T_p$, $G[V(T_p)]$. Such a completion will transform the neighborhood of $u$ into the union of at most two cliques. Instead, if $u$ is contained in a Q-node $q$, we apply the algorithm to the subtree rooted at the sections that contain $u$. For any vertex $w$, let us denote by $T_{n(w)}$ the subtree rooted at the node/sections that contain $w$. Following the previous procedure for every maximal center, we obtain a PIG-completion of the graph $G$. To prove that it is optimal, it suffices to see that for every pair of centers $u$ and $v$, the set of vertices that are adjacent to the two forms a (possibly empty) clique. Thus, a completion of $G[V(T_{n(u)})]$ will not contain any edge such that both endpoints are in the closed neighborhood of $v$. That means that $G[V(T_{n(v)})]$ remains unchanged after adding the fill edges necessary to partition $N(u)$ into two cliques. □

Finally, we obtain a positive result for rigid interval graphs. An interval graph $G$ is a *rigid interval graph* if it has a unique clique tree and that tree is a path.

**Corollary 7.3.** *PIG-completion is polynomial-time solvable on rigid interval graphs.*

*Proof.* A rigid interval graph contains no P-nodes, in particular, it contains a single Q-node at the root. Thus, we can apply the greedy step for Q-nodes to obtain an optimal solution. □

## 7.3. An algorithm for the general case

We have seen in the previous section how to extend the polynomial-time algorithm to interval graphs such that their maximal centers are all pairwise disjoint. In this section, we propose a variation of the algorithm, with the same flavor, that solves the problem for interval graphs that contain non-disjoint centers. First, we analyse the case of split-interval graphs, whose structure allows us to adapt the algorithm and still obtain a polynomial runtime, and then extend the idea used for this case to design an exponential algorithm that works for arbitrary interval graphs.

As a motivation, let us first give an explicit example of a graph that contains two non-disjoint maximal centers where applying the algorithm in the proof of Corollary 7.2 fails. Let $G$ be a graph whose MPQ-tree is composed of a Q-node with sections $S_1 = 1, S_2 = \{1, 2\}, S_3 = 2$, with children $q_1, q_2, q_3$, respectively. Let $q_1$ be a Q-node with

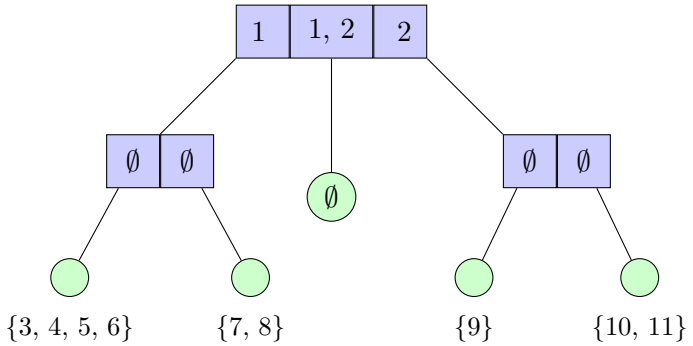



Figure 7.5.: MPQ-tree of an interval graph with two intersecting centers: vertices 1 and 2.

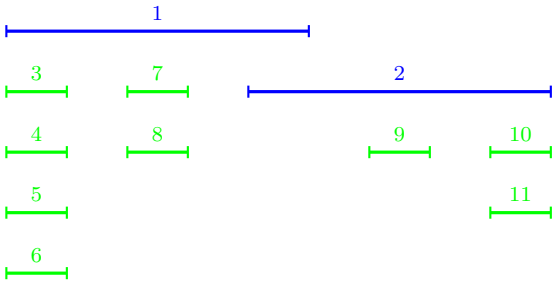



Figure 7.6.: Interval representation of the graph represented by the MPQ-tree on Figure 7.5.

two sections containing both of them the empty set and with two leaves as children, one containing four vertices $3, 4, 5, 6$, and another containing vertices $7, 8$. Let $q_2$ be a leaf containing the empty set and let $q_3$ be another Q-node with two sections containing the empty set and with two leaves as children, one containing a single vertex 9, and the other one comprised of two vertices $10, 11$. The MPQ-tree of $G$ is depicted in Figure 7.5, and an interval representation of the graph is given in Figure 7.6.

Graph $G$ contains two non-disjoint centers: vertices 1 and 2. Then, applying the algorithm to the subtree rooted at $S_1 \cup S_2$ (the sections that contain center 1) returns as an optimal solution the edges $(2, 7)$ and $(2, 8)$, while applying it to the subtree rooted at $S_2 \cup S_3$ (the sections that contain center 2) returns as an optimal solution the edge $(1, 9)$. However, the union of these two solutions is not a valid completion, since vertex 1 is still the center of a $K_{1,3}$. In order to have a PIG-completion, we would need to add edges $(9, 7)$ and $(9, 8)$ on top of the edges already added. Instead, the reader can observe that the global optimum is comprised of the set of edges $\{(2, 7), (2, 8), (9, 10), (9, 11)\}$, of size four instead of five. This example highlights why we need to treat the case of non-disjoint centers differently.

### 7.3.1. Split-interval graphs

In this subsection, we study PIG-completion on split-interval graphs. Recall that a graph $G = (C \cup I, E)$ is a split graph if its vertex set can be partitioned into a set $C$ of pairwise adjacent vertices and a set $I$ of pairwise non-adjacent vertices, and it is split-interval if it is both a split graph and an interval graph. We will see that the structure of split-interval graphs constraints the intersections between maximal centers, so we can easily adapt the previous dynamic programming algorithm. Thus, let us first study the structure of a completion of a split-interval graph.

**Lemma 7.4.** *Let $G = (C \cup I, E)$ be a split-interval graph. Then, every vertex that is a center of a claw belongs to $C$, and at most two vertices of $C$ have private neighbors.*

*Proof.* Let $[v; u_1, u_2, u_3]$ be a claw. Since $G$ is split, vertex $v$ cannot be in $I$, as this would imply that $u_1, u_2$ and $u_3$ would have to be in $C$, which is a contradiction because they are not adjacent to each other. This implies that every vertex which is a center of a claw belongs to $C$. To see that at most two vertices of $C$ have private neighbors, the reader can observe that otherwise there would be an induced net (formed by the three vertices in the clique with private neighbors and the three private neighbors), which contradicts the interval assumption. □

**Lemma 7.5.** *Let $G = (C \cup I, E)$ be a connected split-interval graph. Then, any PIG-completion of $G$ admits a partition of its vertices into three sets $C_1, C_2$ and $C_3$ such that each of them induce a clique in the completion.*

*Proof.* Let $G = (C \cup I, E)$ be a split-interval graph that has $k$ vertices which are maximal centers. By Lemma 7.4, the $k$ centers will belong to $C$, and at most two vertices in the clique will have a private neighbor. If $k = 1$, the center constitutes a universal vertex of the single connected component, so we already know by Observation 7.1 that any completion admits a partition into two sets such that each induces a clique.

Otherwise, the MPQ-tree of $G$ consists of a Q-node with sections $S_1, \ldots, S_m$ containing the $k$ centers $v_1, \ldots, v_k$. Sections $S_1$ and $S_m$ contain the vertices with a private neighbor. Note that the two private neighbors cannot be adjacent to any vertex outside of $C$ (otherwise, these vertices cannot belong to $C$ nor $I$, contradicting the assumption that $G$ is a split graph). In this case, in any valid completion, the neighborhood of each individual center is partitioned into two cliques. Since the set $C$ induces a clique, if the neighborhood of the first center, $v_1$, contains two independent vertices $x$ and $y$, then one of these vertices, say $y$, will also be adjacent to the last center, $v_n$. Indeed, otherwise $v_n, x$ and $y$ would form a set of three independent vertices all adjacent to $c_1$, which contradicts the fact that we have a valid completion. Thus, the neighborhood of $v_n$ cannot contain two non-adjacent vertices which are both not adjacent to $y$. Let $z$ be a vertex contained in $N(v_n)$ which is not adjacent to $y$.

Suppose, towards a contradiction, that there exist four pairwise non-adjacent vertices in the completion of $G$. That, is, there exists a vertex $w$ independent from $x, y$ and $z$, and adjacent to some center $v_2, \ldots, v_{n-1}$. Since the closed neighborhood of every $v_i$, for $i \in \{2, \ldots, n-1\}$, is contained in the union of $N(v_i) \cap N(v_1)$ and $N(v_i) \cap N(v_n)$, this

would mean that $w$ is adjacent to one of $v_1$ or $v_n$. Therefore, we do not have a valid completion, as one of $v_1$ or $v_n$ is adjacent to three independent vertices. □

The previous lemma, together with what we already know about the structure of completions of subtrees rooted at a Q-node, implies that in order to find a completion of a split-interval graph, it suffices to guess the sizes of the first two cliques. We will see that we can do this in polynomial time. Let us first introduce some additional notation. Given a Q-node with sections $S_1, \ldots, S_m$ containing centers $v_1, \ldots, v_k$, we denote by $T_{S(v_i)}$ the subtree rooted at the sections that contain $v_i$, and we define $n_i = |T_{S(v_i)}|$, and $n_{i,j} = |T_{S(v_i)} \cap T_{S(v_j)}|$.

**Theorem 7.2.** *Let $G$ be a split-interval graph. Then, we can compute its PIG-completion in polynomial-time.*

*Proof.* Let $G$ be a split-interval graph with $k$ maximal centers. If $k = 1$, then the center is a universal vertex of one connected component, while the rest of the connected components are formed by isolated vertices. Thus, we can compute a completion in polynomial time using the algorithm from the previous section. Otherwise, if $k > 1$, we know that the root of $G$ is a Q-node with sections $S_1, \ldots, S_m$ containing all the centers $v_1, \ldots, v_k$, and by Lemma 7.5, any completion of $G$ partitions the vertex set into three cliques. Since the root of the MPQ-tree of $G$ is a Q-node, we consider that the cliques are ordered from left to right, and we denote them by $C_1$, $C_2$ and $C_3$, from left to right.

We proceed as follows. For every $r \in [|V(G)|]$ and every $s \in [|V(G)|]$, we define the subproblem:

$$W_{r,s} := \min\{|F| : F \text{ is a set of fill edges of } G[V(T_v)] \text{ that yields a partition}(C_1, C_2, C_3) \text{ with } |C_1| = r, |C_2| = s\}$$

Note that not every pair of values of $r$ and $s$ is feasible, for starters, we need $r + s \leqslant |V(G)|$. We will filter the feasible pairs in a later step. The high-level idea of the algorithm is similar to that of the original algorithm: we start with every vertex in the third clique, $C_3$, and then swap $r$ vertices to the first clique and $s$ to the second clique. To compute the "swapping costs", we need to compute the left and right costs of every subtree/vertex as in the step for Q-nodes of the orginal algorithm, that is, as in Equations 7.1, 7.2, 7.3 and 7.4. Then, the cost of swapping a vertex to the first clique is given by the same function as in the original algorithm, Equation (7.5), whereas to compute the cost of swapping a vertex to the second clique, we need to update the left costs of every vertex/subtree that remains in $C_3$, since now we do not need to account for vertices which are already placed in the first clique. We define:

$$c^1_{r,x} := \text{cost of swapping a vertex of } x \text{ to the first clique when } |C_1| = r$$
$$c^2_{r,s,x} := \text{cost of swapping a vertex of } x \text{ to the second clique when } |C_1| = r, |C_2| = s$$
$$u_{r,s,x} := \text{number of vertices of } x \text{ in } C_1 \text{ or } C_2 \text{ when } |C_1| = r, |C_2| = s$$

- **Initialization**. For every child $t_i$ of a section $S_i$, with $i \in [m]$, and every $r \in \left[\left\lfloor \frac{|T_{t_i}|}{2} \right\rfloor + 1\right]$ we precompute the values $M_{t_i,r}$ as in the original algorithm. Indeed, $G[V(T_{S_i})]$ contains a universal vertex for every $i$, since no section $S_i$ is empty (otherwise, every center to the left of the empty section would be disjoint from every center to the right of the empty section), so the vertices of subtree $T_{t_i}$ will be split into at most two cliques. Thus, it suffices to know the cost of a completion that yields a partition with the smallest clique having $r$ vertices.

- **Base case**. When $r = 0$ and $s = 0$:

$$\begin{cases} W_{0,0} & = \sum_i \left( M_{t_i,0} + r(T_{t_i}) \cdot |T_{t_i}| + \sum_{x^l \in S_i} r(x) \right) \\ c^1_{0,x} & = \mathit{update}(x, 0)\,, \forall x \\ c^2_{0,0,x} & = \mathit{update}(x, 0)\,, \forall x \\ u_{0,0,x} & = 0\,, \forall x \end{cases}$$

- **Step for $\mathbf{W_{r,0}}$**. We compute $W_{r,0}$ from $W_{r-1,0}$ as follows. We will set $c^2_{r,0,x} = +\infty$ to indicate that there is a vertex of $x$ already placed in $C_1$.

$$\begin{cases} x & = \arg\min_x c^1_{r-1,x} \\ W_{r,0} & = W_{r-1,0} + c^1_{r-1,x} \\ u_{r,0,x} & = u_{r-1,0,x} + 1 \text{ and } u_{r,0,y} = u_{r-1,0,y},\ \forall y \neq x \\ c^1_{r,x} & = \mathit{update}(x, u_{r,0,x}) \text{ and } c^1_{r,y} = c^1_{r-1,y},\ \forall y \neq x \\ c^2_{r,0,x} & = +\infty \text{ and } c^2_{r,0,y} = c^2_{r-1,0,y},\ \forall y \neq x \end{cases} \tag{7.8}$$

- **Step for $\mathbf{W_{r,s}}$**. We compute $W_{r,s}$ from $W_{r,s-1}$. First note that we might not always be able to compute $W_{r,1}$. Indeed, there can exist a subtree $x$ such that $c^2_{r,0,x} = \infty$ and $u_{r,0,x} < |x|$ (that is, some of its vertices have been placed in the first clique, but not all). Then, the remaining $|x| - u_{r,0,x}$ vertices must all be placed in the second clique (recall that every subtree can be split into at most two cliques). Furthermore, we cannot compute the value of swapping a single one of the remaining vertices to the second clique, since this would split $x$ into three cliques, and we have not precomputed these values. Thus, we will swap all of the remaining vertices of $x$ at once. By monotonicity, at most one subtree can satisfy this condition. Let $t$ be such subtree. Then, for $s = |t| - u_{r,0,x}$, we have

$$\begin{aligned} W_{r,s} &= W_{r,0} - r(t) \cdot s \\ u_{r,s,t} &= |t| \end{aligned}$$

  It just remains to establish the value of $c^2_{r,s,x}$ for every $x$. To do so, we need to update the left costs of every remaining vertex/subtree, that is, every $x$ such that $c^2_{r,0,x} \neq +\infty$. Indeed, we need to substract from their left cost the number of vertices in the first clique that are strictly to their left. Formally, for every $x$ with

$c^2_{r,0,x} \neq +\infty$,

$$l_{r,x} = l(x) - \sum \{|y| \mid y \neq t, y \prec x, c^2_{r,0,y} = +\infty\}$$

For $s = |t| - u(r,t)$, we let

$$c^2_{r,s,x} = \begin{cases} +\infty & \text{if } c^2_{r,0,x} = +\infty, \\ update_{l_r}(x,0) & \text{otherwise.} \end{cases}$$

Where $update_{l_r}$ is defined as Equation (7.5) but replacing the left cost of $x$ by $l_{r,x}$. Then, for every $s > |t| - u(r,t)$,

$$\begin{cases} x & = \arg\min_x c^2_{r,s-1,x} \\ W_{r,s} & = W_{r,s-1} + c^2_{r,s-1,x} \\ u_{r,s,x} & = u_{r,s-1,x} + 1 \text{ and } u_{r,s,y} = u_{r,s-1,y},\ \forall y \neq x \\ c^2_{r,s,x} & = update_{l_r}(x, u_{r,s,x}) \text{ and } c^2_{r,s,y} = c^2_{r,s-1,y},\ \forall y \neq x \end{cases}$$

We have computed $W_{r,s}$ for almost every possible value of $r$ and $s$. However, some of this pairs might still not be feasible. Indeed, if the value of $s$ is too small, there can be a vertex of a section that is contained in the first clique that is also adjacent to vertices of the other two cliques. Let us thus compute the feasible values of $r$ and $s$. Once we have fixed $r$, we need to ensure that the first center $v_1$ does not intersect more than two independent vertices. That is, if $r$ is smaller than $n_1$, then $s$ needs to be large enough to contain every vertex that belongs to the subtree rooted at the sections that contain $v_1$. Thus, we distinguish two cases:

- If $r > n_1 - n_{1,2}$, then the second clique must contain every other vertex (because center $v_2$ is already adjacent to an independent vertex in this partial solution). Thus, the only feasible value of $s$ is $n_1 + n_2 - n_{1,2} - r$.
- If $r < n_1 - n_{1,2}$, then we only have the constraint that $s$ must be greater or equal to $n_1 - r$ (because every other vertex that is adjacent to $v_1$ must belong to the second clique). Thus, the possible values of $s$ are in the range $[n_1 - r, |V(G)|]$.

Then, the optimum cost of a completion of $G$ is given by the minimum among the feasible pairs $r$ and $s$ of $W_{r,s}$. □

### 7.3.2. Arbitrary interval graphs

In this section, we explain how to modify the previous algorithm to return an optimal solution when the input graph is an arbitrary interval graph.

**Definition 7.2.** *We say that a set of maximal centers $\{v_1, \dots v_k\}$ is* non-disjoint *if there exists a Q-node $q$ with sections $S_1, \dots, S_m$ such that every center is contained in a section of $q$ and for every pair $v_i, v_{i+1}$, with $i \in [k-1]$, there exists a section $S_j$ of $q$ such that $v_i \in S_j$ and $v_{i+1} \in S_j$.*

Let $G$ be an interval graph and let $K$ denote a maximum-cardinality set of maximal centers that are non-disjoint. Given the set $K$, we define $\mathrm{Nb}(K) = \bigcup_{v_i \in K} \mathrm{Nb}(v_i)$. Let $k = |K|$. We call $k$ the *maximum number of non-disjoint centers.*

**Lemma 7.6.** *Let $G$ be an interval graph. Let $K$ be a maximum cardinality set of maximal centers of that are non-disjoint. Let $|K| = k$. Then, a completion of $G$ partitions the neighborhood of $K$ in at most $k + 1$ different cliques.*

*Proof.* Let $v_1, \ldots, v_k$ be the $k$ centers of $K$. By assumption, the intersection of the closed neighborhoods of $v_i$ and $v_{i+1}$ is non-empty for every $i \in \{1, \ldots, k-1\}$. Thus, if $x$ and $y$ are two independent vertices adjacent to $v_1$, then $v_2$ can be adjacent to at most one vertex $z$ that is independent from both $x$ and $y$. Similarly, $v_3$ can be adjacent to at most one vertex independent from $x, y$ and $z$. By induction, a completion of $G$ restricted to the subtrees rooted at the sections that contain centers $v_1$ until $v_{i-1}$ can contain $i$ independent vertices. When we add the vertices of the subtrees rooted at the sections that contain $v_i$, since $v_i$ can be adjacent to at most one more independent vertex, we get that the completion of $G$ restricted to the new subtrees can contain at most $i + 1$ independent vertices. □

We will devise an algorithm running in time $O(n^k)$ to solve PIG-completion. In particular, this implies that the problem is in XP parameterized by the maximum number of non-disjoint centers.

**Theorem 7.3.** *PIG completion is in* XP *parameterized by the maximum number of non-disjoint centers. In particular, given an interval graph $G$ whose maximum number of non-disjoint centers is $k$, there exists an $O(n^k)$ algorithm that solves PIG-completion, where $n = |V(G)|$.*

*Proof.* We provide an explicit algorithm. Suppose that we have a set $K$ of $k$ vertices $v_1, \ldots, v_k$ that are non-disjoint maximal centers. Since they are all in the same Q-node, suppose that they are ordered according to the order given by the Q-node. Note that since every $v_i$ is a maximal center, we have $v_1^l \prec \ldots \prec v_k^l$ and $v_1^r \prec \ldots \prec v_k^r$. Let us refer to the set of sections that contain vertices of $K$ as $S_K$. By Lemma 7.6, if we consider the subtree rooted at $S_K$, we know that its vertices will be partitioned into at most $k+1$ cliques, $C_1, \ldots, C_{k+1}$, ordered from left to right. As in the proof of Corollary 7.2, the completion of $G$ is given by summing up the completions of the subgraphs $G[V(T_{S_K})]$ for every maximum set $K$ of non-disjoint maximal centers. Thus, we explain how to obtain a completion of $G[V(T_{S_K})]$.

Let $v$ be the Q-node that contains all the centers of $K$. Due to the partial order induced by $v$, we can again exploit the monotonicity of the costs to obtain an optimal greedy-like algorithm. For every $r_i \in [n], i \in [k]$, we will compute the cost $W_{r_1,\ldots,r_k}$, which represents the cost of a solution of the subtree rooted at $S_K$ which partitions the vertices into $k+1$ cliques in such a way that the i-th clique contains $r_i$ vertices (and the rightmost clique contains $n - \sum_{i \in [k]} r_i$ vertices). Notice that we label the cliques from left to right according to the ordering given by the Q-node.

Formally, we define:

$W_{r_1,\ldots,r_k} :=$ min$\{|F| : F$ is a set of fill edges of $G[V(T_{S_k})]$ that partitions the vertices into $(C_1,\ldots,C_{k+1})$ with $|C_i| = r_i, i \in [k]\}$

$c^i_{r_1,\ldots,r_i,x} :=$ cost of swapping a vertex of $x$ to clique $i$ when $|C_i| = r_i, i \in [k]$

$u_{r_1,\ldots,r_k,x} :=$ number of vertices of $x$ not in $C_{k+1}$ when $|C_i| = r_i, i \in [k]$

To compute the costs $W_{r_1,\ldots,r_k}$, we first compute the left/right cost of every subtree and every vertex in a section as in Equations 7.1, 7.2, 7.3 and 7.4, and then proceed as follows:

- **Base case**. When $r_i = 0$ for every $i \in [k]$:

$$\begin{cases} W_{0,\ldots,0} & = \sum_i \left(M_{t_i,0} + r(T_{t_i}) \cdot |T_{t_i}| + \sum_{x^l \in S_i} r(x)\right) \\ c^i_{0,\ldots,0,x} & = update(x,0) \text{ for } i \in [k] \\ u_{0,\ldots,0,x} & = 0 \end{cases}$$

- **Step for $\mathbf{W_{r_1,0,\ldots,0}}$**. We compute $W_{r_1,0,\ldots,0}$ from $W_{r_1-1,0,\ldots,0}$ by adapting Equation (7.8).

$$\begin{cases} x & = \arg\min_x c^1_{r_1-1,0,\ldots,0,x} \\ W_{r_1,0,\ldots,0} & = W_{r_1-1,0,\ldots,0} + c^1_{r_1-1,0,\ldots,0,x} \\ u_{r_1,0,\ldots,0,x} & = u_{r_1-1,0,\ldots,0,x} + 1 \text{ and } u_{r_1,0,\ldots,0,y} = u_{r_1-1,0,\ldots,0,y}, \forall y \neq x \\ c^1_{r_1,x} & = update(x, u_{r_1,x}) \text{ and } c^1_{r_1,y} = c^1_{r_1-1,y}, \forall y \neq x \\ c^i_{r_1,0,\ldots,0,x} & = +\infty \text{ and } c^i_{r_1,0,\ldots,0,y} = c^i_{r_1-1,0,\ldots,0,y}, \forall y \neq x \end{cases}$$

- **Step for $\mathbf{W_{r_1,\ldots,r_i,r_{i+1},\ldots,0}}$**. To compute $W_{r_1,\ldots,r_i,1,\ldots,0}$ from $W_{r_1,\ldots,r_i,0,\ldots,0}$, we proceed in a similar manner as in Section 7.3.1. We only compute value $W_{r_1,\ldots,r_i,r_{i+1},0\ldots,0}$ for the tuples of values $(r_1,\ldots,r_i)$ which are feasible. Given a feasible set of values $(r_1,\ldots,r_{i-1})$, whether combination $(r_1,\ldots,r_{i-1},r_i)$ is feasible depends only on the value of $r_{i-1}$:

  - If $r_{i-1}$ is larger than $n_{i-1} - n_{i,i-1}$, then this means that center $v_i$ is already adjacent to vertices of two different cliques, and so every other vertex that is adjacent to $v_i$ and not in the intersection must be in the i-th clique. That is, the only feasible value of $r_i$ is $r_{i-1} - (n_{i-1} - n_{i-1,i})$.

  - Otherwise, $r_i$ can take values between $n_i - n_{i-1,i}$ and $n_i$ (the vertices in the intersection must all be in the right-clique of $v_{i-1}$).

  Now, given a feasible set of values $(r_1,\ldots,r_{i-1},r_i)$, if there exists a subtree $t$ such that $c^i_{r_1,\ldots,r_i,x} = +\infty$ and $u_{r_1,\ldots,r_i,0,\ldots,0,x} < |x|$, we know that $r_{i+1}$ must be greater

than $|t| - u_{r_1,\dots,r_i,0,\dots,0,x}$. For $r_{i+1} = |t| - u_{r_1,\dots,r_i,0,\dots,0,x}$, we let:

$$W_{r_1,\dots,r_{i+1},0,\dots,0} = W_{r_1,\dots,r_i,0,\dots,0} - r(t) \cdot r_{i+1}$$
$$u_{r_1,\dots,r_{i+1},0,\dots,0,t} = |t|$$

Before computing $W_{r_1,\dots,r_{i+1}}$ for $r_{i+1} > |t| - u_{r_1,\dots,r_i,1,\dots,0,x}$, we need to update the left costs of every remaining vertex/subtree, that is, every $x$ such that $c^i_{r_1,\dots,r_i,x} \neq +\infty$. Indeed, we neew to substract from their left cost the number of vertices in the first $i$ cliques that are strictly to their left. Formally, for every $x$ with $c^i_{r_1,\dots,r_i,x} \neq +\infty$,

$$l_{r_1,\dots,r_i,0,\dots,0,x} = l(x) - \sum \{|y| \mid y \neq t, y \prec x, c^i_{r_1,\dots,r_i,x} = +\infty\}$$

For $r_{i+1} = |t| - u_{r_1,\dots,r_i,0,\dots,0,x}$, we let

$$c^{i+1}_{r_1,\dots,r_{i+1},x} = \begin{cases} +\infty & \text{if } c^i_{r_1,\dots,r_i,0,\dots,0,x} = +\infty, \\ update_{l_{r_1,\dots,r_i,0,\dots,0,x}}(x,0) & \text{otherwise.} \end{cases}$$

Where $update_{l_{r_1,\dots,r_i,0,\dots,0,x}}$ is defined as Equation (7.5) but replacing the left cost of $x$ by $l_{r_1,\dots,r_i,0,\dots,0,x}$. Then, for every $r_{i+1} > |t| - u_{r_1,\dots,r_i,0,\dots,0,x}$,

$$\begin{cases} x & = \arg\min_x c^{i+1}_{r_1,\dots,r_{i+1}-1,0,\dots,0,x} \\ W_{r_1,\dots,r_{i+1},0,\dots,0} & = W_{r_1,\dots,r_{i+1}-1,0,\dots,0,x} + c^{i+1}_{r_1,\dots,r_{i+1}-1,0,\dots,0,x} \\ u_{r_1,\dots,r_{i+1},0,\dots,0,x} & = u_{r_1,\dots,r_{i+1}-1,0,\dots,0,x} + 1 \\ u_{r_1,\dots,r_{i+1},0,\dots,0,y} & = u_{r_1,\dots,r_{i+1}-1,0,\dots,0,y},\ \forall y \neq x \\ c^{i+1}_{r_1,\dots,r_{i+1},x} & = update_{l_{r_1,\dots,r_{i+1}}}(x, u_{r_1,\dots,r_{i+1},0,\dots,0,x}) \\ c^{i+1}_{r_1,\dots,r_{i+1},0,\dots,0,y} & = c^{i+1}_{r_1,\dots,r_{i+1}-1,0,\dots,0,y},\ \forall y \neq x \\ c^{j}_{r_1,\dots,r_{i+1},0,\dots,0,x} & = +\infty \\ c^{j}_{r_1,\dots,r_{i+1},0,\dots,0,y} & = c^{j}_{r_1,\dots,r_{i+1}-1,0,\dots,0,y},\ \forall y \neq x,\ \forall j > i+1. \end{cases}$$

The optimal cost of a completion of $G[V(T_{S_K})]$ is given by the minimum over the feasible $k$-tuples $(r_1, \dots, r_k)$ of $W_{r_1\dots,r_k}$. □

We conclude this section by pointing out that even though we have provided an algorithm which is exponential in the maximum number of non-disjoint claws, the complexity of PIG-completion on arbitrary interval graphs remains unsettled. In fact, even whether the problem is FPT parameterized by the maximum number of non-disjoint centers remains as an open question. It would also be interesting to explore whether the dynamic programming approach on the MPQ-tree of the interval graph can be applied when we allow other operations, such as edge deletion.

# 8. Conclusion and future work

In this manuscript, we have studied multiple interval graphs from a structural and algorithmic perspective, as well as a graph modification problem on interval graphs.

We started Chapter 4 by providing a useful characterization of (disjoint) unit 2-interval graphs and an efficient software to test whether a graph belongs to this class of graphs. Then, in Chapter 5, we showed that the classes of disjoint unit and unit $d$-interval graphs are not equivalent and we obtained a complete characterization of unit $d$-interval graphs that are also interval. However, for the class of disjoint unit $d$-interval graphs which are also interval, only a partial characterization was obtained. This raises the question of whether we can obtain an analogous result, or whether we can even recognize this class in polynomial time. We also studied the containment relations between the classes of balanced and disjoint balanced $d$-interval graphs. Nevertheless, some relations between the subclasses obtained with the two different definitions of $d$-interval remain unknown, such as whether the class of unit $d$-interval graphs is contained in the class of disjoint balanced $d$-interval graphs for $d > 2$.

In Chapter 6, we proved that recognizing both unit and disjoint unit $d$-interval graphs of representation depth $r$ is NP-complete for every $d \geqslant 2$ and every $r \geqslant 4$, and obtained as a corollary that recognizing $(x, \ldots, x)$ $d$-interval graphs is NP-complete for $x \geqslant 11$. Apart from closing the gap to settle the complexity of recognizing $(x, \ldots, x)$ $d$-interval graphs for $2 \geqslant x \geqslant 10$, or depth 3 $d$-interval graphs, other interesting lines of research would be to investigate whether there exists a constant-ratio approximation of the unit interval number of a graph, or to study the following parameterized problem:

Parameterized (disjoint) unit $d$-interval recognition

| | |
|---|---|
| ***Instance:*** | An arbitrary graph $G$ and a parameter $k$. |
| ***Goal:*** | Determine whether we can transform $G$ into a unit interval graph by splitting at most $k$ of its vertices. |

Finally, in Chapter 7, we focused on the graph modification problem Proper interval graph-completion on interval graphs. Even though we developed a polynomial-time algorithm for various cases, including interval graphs with a universal vertex, split-interval graphs, or interval graphs with no intersecting centers of maximal claws, the complexity in the general case remains open. Nevertheless, we provided an XP algorithm for arbitrary interval graphs, parameterized by the maximum number of non-disjoint centers of maximal claws. Given that the problem is FPT parameterized by the number of fill edges, it would be worth studying whether it is also FPT by this structural parameter (which is incomparable to the natural parameter). Additional research could focus on exploring whether the approach based on dynamic programming on the MPQ-tree of

the interval graph that we have developed here could be adapted for the edge deletion version of the problem, or for variants that also admit vertex addition/deletion.

# Outline of appendix

The following appendices contain the papers not related to interval graphs as they were originally published. In Appendix A we discuss the classical and parameterized complexity of the problem Parity permutation pattern matching, while in Appendix B, we prove the strong NP-hardness of the problem Balanced Mobiles.

# A. Parity Permutation Pattern Matching

Given two permutations, a pattern $\sigma$ and a text $\pi$,Parity Permutation Pattern Matching asks whether there exists a parity and order preserving embedding of $\sigma$ into $\pi$. While it is known that Permutation Pattern Matching is in FPT, we show that adding the parity constraint to the problem makes it W[1]-hard, even for alternating permutations or for 4321-avoiding patterns. However, the problem remains in FPT if $\pi$ avoids a fixed permutation, thanks to a recent meta-theorem on twin-width. On the other hand, as for the classical version, Parity Permutation Pattern Matching remains polynomial-time solvable when the pattern is separable, or if both permutations are 321-avoiding, but NP-hard if $\sigma$ is 321-avoiding and $\pi$ is 4321-avoiding.

## A.1. Introduction

Permutations are one of the most fundamental objects in discrete mathematics, and in concrete terms, deciding if a permutation contains another permutation as a pattern is one of the most natural decision problems. More precisely, in the well-known Permutation Pattern Matching (PPM) problem, given two permutations $\sigma$ and $\pi$, the task is to determine if $\sigma$ is a pattern of $\pi$, or equivalently, if $\pi$ *contains* a subsequence which is order-isomorphic to $\sigma$. For example, if $\pi = 3\,1\,5\,4\,2$, it contains $\sigma = 2\,3\,1$, since $3\,5\,2$ is a subsequence of $\pi$ with the same relative order as $\sigma$, but $\pi$ does not contain $\sigma = 1\,2\,3$, as there are no three left-to-right increasing elements in $\pi$. In the latter case, we say that $\pi$ *avoids* $1\,2\,3$. The notion of avoidance allows to define various classes of permutations as sets of permutations that avoid some given patterns. For example, 321-avoiding permutations, or $(2413, 3142)$-avoiding permutations, which are known as *separable permutations*.

PPM has been proved to be NP-complete by Bose, Buss, and Lubiw in 1998 [25]. However, some special cases, such as Longest Increasing Subequence, or the cases where both $\sigma$ and $\pi$ are separable or 321-avoiding, are known to be polynomial time solv-

able [43, 25, 82, 4, 29]. In fact, it was shown in [93] that PPM is always polynomial-time solvable if the pattern avoids any fixed permutation $\tau \in \{1, 12, 21, 132, 231, 312, 213\}$, and NP-complete if it avoids any other fixed permutation. This result was then extended in [94]. Furthermore, exact exponential time issues have been considered [3, 16, 73, 27].

A breakthrough result of Guillemot and Marx [81] shows that PPM is fixed parameter tractable when parameterized by the size of the pattern $\sigma$. The key element is the use of a new *width measure* structure theory of permutations. They proved that the problem can be solved in time $2^{\mathcal{O}(k^2 \log k)}n$, and later on, Fox improved the running time of the algorithm by removing a $\log k$ factor from the exponent [63]. This led to the question of whether a graph-theoretic generalization of their permutation parameter could exist. This was answered positively in [22], by introducing the notion of *twin-width*, which has proven huge success recently. It was shown that graphs of bounded twin-width define a very natural class with respect to computational complexity, as FO model checking becomes linear in them.

Pattern matching for permutations, together with its many variants, has been widely studied in the literature (the best general reference is [101], see also [26]). Here we introduce a natural variation of PPM, called Parity Permutation Pattern Matching, that incorporates the additional constraint that the elements of $\sigma$ have to map to elements of $\pi$ with the same parity, i.e., even (resp. odd) elements of $\sigma$ have to be mapped to even (resp. odd) elements of $\pi$. For one, pattern avoidance with additional constraints [5, 28], including parity restrictions [144, 74, 102], has emerged as a promising research area. For another, Parity Permutation Pattern Matching aims at providing concrete use cases of the 2-colored extension of PPM introduced in [82]. Note that Parity Permutation Pattern Matching can be viewed as a variant of 2-colored PPM where the colors encode the parity of the elements.

Surprisingly, we show that it does not fit into the twin-width framework, and this increases the complexity of the problem, as it becomes W[1]-hard parameterized by the length of the pattern. In fact, the approach used by Guillemot and Marx [81] to prove that PPM is FPT is based on a result that states that given a permutation $\pi$, there exists a polynomial time algorithm that either finds an $r \times r$-grid of $\pi$ or determines that the permutation has bounded width (and returns the merge sequence of the decomposition, which is used to solve the PPM problem in FPT time). This win-win approach works because, if $\pi$ contains an $r \times r$-grid, it's not hard to see that it contains every possible pattern $\sigma$. However, this cannot be generalized to Parity PPM, as here we have no information on the parity of the elements of the grid, and thus, it is not guaranteed that every pattern maps via a parity respecting embedding into the grid.

**Structure of the paper** The paper is organized as follows. Section A.2 briefly introduces the necessary concepts and definitions. Section A.3 is devoted to proving that Parity PPM parameterized by the length of the pattern is W[1]-hard but remains in FPT when the *twin-width* of the host permutation is bounded. Section A.4 is concerned with P vs NP issues and we show that Parity PPM and PPM behave similarly. A summary of the complexity of the problems is given in Table A.1. An extended abstract

| | PPM | Parity PPM |
|---|---|---|
| General case | NP-hard, FPT | W[1]-hard |
| Separable permutations | P | P |
| 321-av $\sigma$ and 321-av $\pi$ | P | P |
| 321-av $\sigma$ and 4321-av $\pi$ | NP-hard | NP-hard |
| 4321-av $\sigma$ | FPT | W[1]-hard |
| Alternating $\pi$ and $\sigma$ | FPT | W[1]-hard |
| $\pi$ is fixed pattern avoiding | FPT | FPT |

Table A.1.: Summary of known results (for PPM) and our results (for Parity PPM). All the results regarding the parameterized complexity consider the size of the pattern $\sigma$ as the parameter.

of this paper appeared in [10].

## A.2. Preliminaries

We let $[n]$ stand for the set $\{1, 2, \ldots, n\}$. A permutation of length $n$ is a bijection $\pi : [n] \to [n]$ and we write $S_n$ for the set of all bijections of length $n$. Given two permutations $\sigma \in S_k$ and $\pi \in S_n$, we say that $\pi$ (the text, or the host) *contains* $\sigma$ (the pattern) if there is an embedding from the elements of $\sigma$ into the elements of $\pi$, i.e., an injective function $f$ such that for every pair of elements $x$ and $y$ of $\sigma$, their images $f(x)$ and $f(y)$ of $\pi$ are in the same relative order as $x$ and $y$. Otherwise, we say that $\pi$ *avoids* $\sigma$. If $\pi$ contains $\sigma$, we write $\sigma \preceq \pi$.

A *permutation class* is a set $\mathcal{C}$ of permutations such that for every permutation $\pi \in \mathcal{C}$, every pattern of $\pi$ is also contained in $\mathcal{C}$. Every permutation class can be defined by the minimal set of permutations that do not lie inside it, and we define this as $\mathcal{C} = \mathrm{Av}(B)$, where $B$ is the minimal set of avoided permutations. *Alternating permutations* are those permutations $\sigma \in S_n$ such that $\sigma_1 > \sigma_2 < \sigma_3 > \ldots$.

In this manner, we can define the class $\mathrm{Av}(4321)$, which is the set of permutations that avoid 4321, $\mathrm{Av}(321)$, which is the set of permutations that avoid 321, and $\mathrm{Av}(2413, 3142)$, i.e., the class of permutations that avoid both 2413 and 3142. As we mentioned in the introduction, the latter is known as the class of *separable permutations*, and it can also be characterized as the set of permutations that have a separating tree. In other words, a permutation is separable if there exists an ordered binary tree $\mathcal{T}$ in which the elements of the permutation appear in the leaves and such that the descendants of a tree node form a contiguous subset of these elements.

The problem of determining whether a fixed pattern is contained in a permutation has been well studied in the literature, and it is referred to as Permutation Pattern Matching. Here, we study a natural variation of PPM, Parity Permutation Pattern Matching, which is defined below.

**Definition A.1.** *Given two permutations, a* pattern $\sigma \in S_k$ *and a* text $\pi \in S_n$, *Per-*

*mutation Pattern Matching asks whether $\pi$ contains $\sigma$.*

**Definition A.2.** *An injective function $f$ from $\sigma$ to $\pi$ is a parity respecting embedding if for all elements $x$ and $y$ of $\sigma$, $f(x)$ and $f(y)$ are in the same relative order as $x$ and $y$, and for every element $x$ of $\sigma$, $f(x)$ has the same parity as $x$.*

We say that an occurrence of a pattern $\sigma$ in a permutation $\pi$ respects parity if there is a parity respecting embedding of $\sigma$ into $\pi$. Furthermore, if there is an occurrence of $\sigma$ in $\pi$ which respects parity, we say that $\pi$ *parity contains* $\sigma$, and we write $\sigma \preceq_P \pi$. Otherwise, we say that $\pi$ *parity avoids* $\sigma$.

**Definition A.3.** *Given a pattern $\sigma$ and a text $\pi$, Parity PPM is the problem of determining whether there exists a parity respecting embedding of $\sigma$ into $\pi$.*

As a remark, note that if instead of considering the problem PPM with the constraint that even (resp. odd) elements have to map to even (resp. odd) elements, we require that elements in even (resp. odd) indices (positions) map to elements in even (resp. odd) indices, the problem is equivalent. Indeed, $\sigma$ parity avoids $\pi$ if and only if $\sigma^{-1}$ parity index avoids $\pi^{-1}$. For example, the parity + order preserving embedding of $\sigma = 2\,4\,1\,3$ into $\pi = 4\,\mathbf{2}\,7\,\mathbf{6}\,3\,\mathbf{1}\,\mathbf{5}$ yields the parity-index + order-preserving embedding of $\sigma^{-1} = 3\,1\,4\,2$ into $\pi^{-1} = \mathbf{6}\,\mathbf{2}\,5\,1\,\mathbf{7}\,\mathbf{4}\,3$ (occurrences are depicted with bold integers).

To see this, assume that there is a parity respecting embedding of $\sigma$ into $\pi$. Denote by $P_\sigma(i)$ the position in $\sigma$ of the element with value $i$ and by $f$ the parity respecting injective map between $\sigma$ and $\pi$ associated to the embedding. Since $\sigma^{-1} = P_\sigma(1)\ ...\ P_\sigma(k)$, and $f$ respects parity, if $f(i) = j$, both $i$ and $j$ have the same parity, and thus, the indices in the inverses will also have the same parity (by definition, odd elements are placed in odd indices in the inverses, and vice versa). Furthermore, since $f$ is an embedding, for $i < j$, $\sigma_i < \sigma_j$ if and only if $f(\sigma_i) < f(\sigma_j)$. Thus, $P(\sigma_i)$ is to the left of $P(\sigma_j)$ in both $\sigma^{-1}$ and $\pi^{-1}$, and by assumption, we also have $i < j$, so $f$ induces a parity index respecting embedding between the inverses.

In this paper, we focus mainly on the parameterized complexity of the above-mentioned problem. Parameterized complexity allows the classification of NP-hard problems on a finer scale than in the classical setting. Fixed parameter tractable (FPT) algorithms are those with running time $O(f(k) \cdot poly(n))$, where $n$ is the size of the input and $f$ is a computable function that depends only on some well-chosen parameter $k$. On the other hand, problems for which we believe that there does not exist an algorithm with that running time belong to the W-hierarchy. Indeed, it is widely believed that W[1] $\neq$ FPT, and so being W[1]-hard is strong evidence against being in FPT. We refer to [49] for more background on the topic.

## A.3. Parameterized complexity

We already saw in the introduction that PPM is in FPT in general, and why the win-win approach of Guillemot and Marx for the parameterized algorithm for PPM doesn't work

for Parity PPM. This intuition is correct and we prove that Parity PPM is W[1]-hard. In fact, we prove something stronger, which is that Parity PPM is W[1]-hard even when restricted to alternating permutations or when the pattern is 4321-avoiding. Note that both results are independent, as alternating permutations and 4321-avoiding permutations are not comparable, but they both imply the W[1]-hardness of the general case. However, we see that the twin-width framework (of which the parameterized algorithm of Guillemot and Marx is an initial step) will be useful to prove that Parity PPM is FPT when the text avoids a fixed pattern.

### A.3.1. Parameterized hardness for alternating permutations

**Theorem A.1.** *Parity PPM is* W[1]*-hard parameterized by the length $k$ of the pattern, even for alternating permutations $\sigma$ and $\pi$.*

*Proof.* We reduce from $k$-Clique in general graphs, which is known to be W[1]-hard parameterized by the size $k$ of the clique [49]. Given as input a graph $G$ and a positive integer $k$, $k$-Clique asks whether $G$ contains a clique of size $k$. For our reduction, let $G = (V, E)$ be a graph with $|V| = n$ and $|E| = m$, and a parameter $k$. We construct a permutation $\sigma$ that depends only on the parameter $k$, and a permutation $\pi$ that depends on $G$, such that there exists a clique of size $k$ in the graph $G$ if and only if there is a parity respecting embedding of $\sigma$ into $\pi$.

**Construction** We describe the construction of $\pi$ from $G$. The high-level idea is to construct different gadgets to represent the vertices and the edges of the graph, and to *link* each edge gadget to the corresponding vertex gadgets. More precisely, we *link* the gadget associated to edge $(u, v)$ to the gadgets associated to vertices $u$ and $v$ by placing elements of value greater than the minimum element of each vertex gadget and smaller than the maximum element of each vertex gadget between the elements of the edge gadget.

We define the following gadgets (see also Figure A.1):

- A *vertex gadget* $\pi[V]$ that is a direct sum of $n$ decreasing permutations, all order-isomorphic to 21 and composed of odd elements. It contains $2n$ elements, starts at position 1 and has as smallest element $8m + 3$.
- An *edge gadget* $\pi[E]$ that is a direct sum of $m$ permutations, all order-isomorphic to 435261 and composed of odd elements. It contains $6m$ elements, starts at position $2n + 5$ and has as smallest element 3.
- The *separator gadget* is composed of the four even integers $4(n+3m)+4$, $8m+2$, $4(n+3m)+2$ and 2 (and hence is order-isomorphic to 4231). The *separator gadget* lies between the *vertex gadget* and the *edge gadget*.
- Let even be the $2(n + 3m) - 2$ even integers between 4 and $4(n + 3m)$ that do not appear in the *vertex gadget*, the *separator gadget* or the *edge gadget*. The *even garbage gadget* is the alternating sequence composed of the even integers of even.

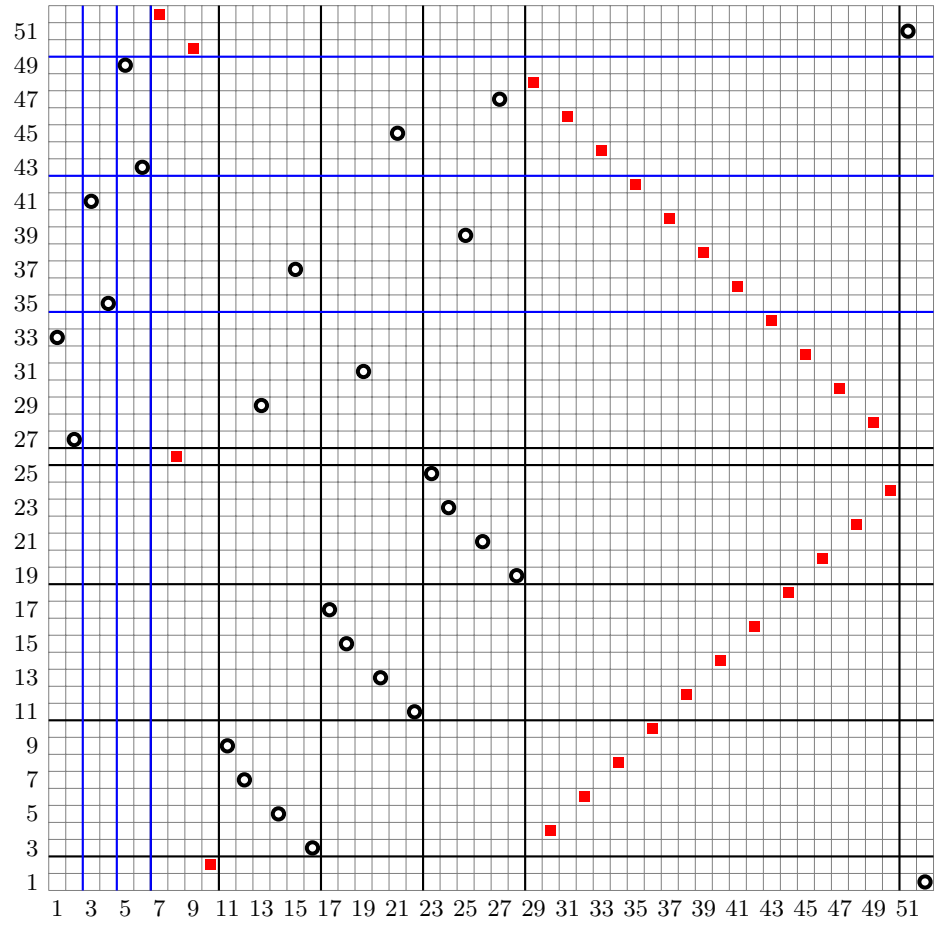



Figure A.1.: Illustration of the construction introduced in the proof of Theorem A.1. The permutation in the figure corresponds to a clique of size 3 with vertices $v_1$, $v_2$, $v_3$ and edges $(v_1, v_2)$, $(v_1, v_3)$, $(v_2, v_3)$. Odd elements are represented with black dots and even elements with red squares. Furthermore, blue lines delimit the vertex boxes.

It is constructed recursively from left to right as follows: place (and remove from EVEN) the maximum of EVEN, place (and remove from EVEN) the minimum of EVEN and recurse. It is placed to the right of the edge gadget.

- Let ODD be the two odd elements $4(n + 3m) + 3$ and 1 that do not appear in the *vertex gadget*, the *separator gadget* or the *edge gadget*. The *odd garbage gadget* is the decreasing sequence composed of the two odd integers of ODD. It is constructed as the *even garbage gadget* and placed directly to its right.

Formally, define,

$$\forall v_i \in V,\ \pi[v_i] = \boxed{8m+2+2\sum_{j<i}(deg(v_j)+2)+2(deg(v_i))+3}\ \boxed{8m+2+2\sum_{j<i}(deg(v_j)+2)+1} \quad \text{(A.1)}$$

$$\forall e_k = (i,j) \in E,\ \pi[e_k] = \boxed{8k+1}\ \boxed{8k-1}\ \boxed{8m+2+2\sum_{j'<i}(deg(v'_j)+2)+2\sum_{(i,j')/j'<j}(1)+3}\ \boxed{8k-3}\ \boxed{8m+2+2\sum_{j'<j}(deg(v'_j)+2)+2\sum_{(i',j)/i'<i}(1)+3}\ \boxed{8k-5} \quad \text{(A.2)}$$

$$\pi = \boxed{\pi[v_1]}\ldots\boxed{\pi[v_n]}\ \boxed{4(n+3m)+4}\ \boxed{4(n+3m)+2}\ \boxed{8m+2}\ \boxed{2}\ \boxed{\pi[e_1]}\ldots\boxed{\pi[e_m]}\ \boxed{\text{EVEN}}\ \boxed{\text{ODD}} \quad \text{(A.3)}$$

(boxes are used for readability purposes only and denote individual elements).

The permutation $\sigma$ is constructed similarly to the permutation $\pi$ but using $K_k$ instead of graph $G$. Clearly, this construction can be carried out in polynomial time and $\sigma$ depends only on the parameter $k$, *i.e.*, the new parameter $|\sigma|$ is a function of $k$. Furthermore, both $\sigma$ and $\pi$ are alternating permutations. We claim that there exists a clique of size $k$ in the graph $G$ if and only if there is a parity respecting embedding of $\sigma$ into $\pi$.

**Notation** Before proving the correctness of the reduction, we need to define some notation for the elements of the permutations. Let us denote by $w_i$ and $w'_i$ ($i \in \{1,2,3,4\}$) the four even elements of the separator gadget, placed in between the vertex gadget and the edge gadget of $\sigma$ and $\pi$, respectively. For each vertex $v_i$, with $i \in \{1,\ldots,n\}$, we will refer to the decreasing subsequence of length two associated to it, $\sigma[v_i]$, as the vertex box associated to $v_i$. For each edge $e_i$, with $i \in \{1,\ldots,m\}$, we will refer to the decreasing subsequence of length four associated to it (i.e., the elements of $\sigma[e_i]$ which correspond to 4321 in the permutation 435261) as the edge box of $e_i$.

We will denote the elements of the vertex box associated to vertex $v_i$ as $v_{i,1}$ and $v_{i,2}$, from left to right (i.e., $v_{i,1} > v_{i,2}$), and the elements of the edge box associated to edge $e_i$ as $e_{i,1}$, $e_{i,2}$, $e_{i,3}$ and $e_{i,4}$, again from left to right. On the other hand, for each edge, we denote the two elements placed in between $e_{i,2}$ and $e_{i,3}$, and between $e_{i,3}$ and $e_{i,4}$, as $h_{i,1}$ and $h_{i,2}$, respectively, where here $h_{i,1} < h_{i,2}$ (these are the elements that correspond to the subsequence 56 in $\sigma[e_i]$).

Finally, the even elements to the right of the edge gadget placed below $w_2$ are referred to as $w_{i,1}$, $w_{i,2}$, $w_{i,3}$ and $w_{i,4}$, for every edge $e_i$ with $i \in \{1,\ldots,m\}$, where $w_{i,1}$ is the element $e_{i,4}+1$, $w_{i,2}$ is $e_{i,3}+1$, $w_{i,3}$ is $e_{i,2}+1$, and $w_{i,4}$ is $e_{i,1}+1$. Note that $w_{i,4}$ is not defined for the last edge. On the other hand, the even elements to the right of the edge gadget placed above $w_2$ are denoted as $x_{i,t}$, for every vertex $v_i$ with $i \in \{1,...,n\}$ and for every edge $e_t$ incident to $v_i$, $t \in \{1,...,m_i\}$, (note that $x_{i,t} = h_{a,b}+1$ for some pair $a,b$). Furthermore, we denote by $x_{i,0}$ and $x_{i,m_i+1}$ the even elements in the extremes such that $x_{i,0} = v_{i,2}+1$ and $x_{i,m_i+1} = v_{i,1}+1$. Again, note that $x_{n,m_n+1}$ is not defined.

For the elements of $\pi$, we follow an analogous notation denoting the elements by $v'_{i,1}$, $e'_{i,1}$, etc.

**Direct Implication**

**Claim A.1.** *If there exists a clique of size $k$ in the graph $G$, then there is a parity respecting embedding of $\sigma$ into $\pi$.*

Proof. We will assume that there exists a clique and prove that there is a parity respecting order isomorphism between the two permutations. Let $f$ be the injective map associated to it. Note that the vertices of the clique and the graph are both ordered according to the same linear ordering. Similarly, the edges are ordered according to the ordering of the vertices, as we say that $(u_1, v_1) < (u_2, v_2)$ if $u_1 < u_2$ or $u_1 = u_2$ and $v_1 < v_2$. Fix a clique in $G$ and denote by $v'_1, \ldots, v'_k$ the $k$ vertices of the clique in $G$, and by $e'_1, ..., e'_l$ its $l = \binom{k}{2}$ edges.

The idea is to map each vertex box (resp. each edge box) of $\sigma$ to a vertex box (resp. edge box) associated to a vertex (resp. edge) of a $k$-clique in $G$.

We define $f$ with the following properties:

- $f(v_{i,j}) = v'_{i,j}$, for $i \in \{1, ..., k\}$ and $j \in \{1, 2\}$.
- $f(e_{i,j}) = e'_{i,j}$, for $i \in \{1, ..., l\}$ and $j \in \{1, 2, 3, 4\}$.
- $f(w_i) = w'_i$ for $i \in \{1, 2, 3, 4\}$.
- $f(h_{i,j}) = h'_{i,j}$, for $i \in \{1, ..., l\}$ and $j \in \{1, 2\}$.
- $f(w_{i,1}) = f(e_{i,4}) + 1$, $f(w_{i,2}) = f(e_{i,3}) + 1$, $f(w_{i,3}) = f(e_{i,2}) + 1$, and $f(w_{i,4}) = f(e_{i,1}) + 1$, for every edge $e_i$, with $i \in \{1, \ldots, l\}$ (except the last edge $e_l$, for which only $w_{i,1}, w_{i,2}$ and $w_{i,3}$ are defined).
- For every vertex $v_i$, with $i \in \{1, ..., n\}$, and every edge $e_t$ with $t \in \{1, \ldots, l_i\}$ incident to vertex $v_i$, we have $f(x_{i,0}) = f(v_{i,2}) + 1$; $f(x_{i,t}) = f(h_t) + 1$ (where $h_t$ corresponds to some $h_{t,j}$ with $e_t$ the edge incident to $v_i$ in position $t$); and $f(x_{i,t+1}) = f(v_{i,1}) + 1$. (Note that for the last vertex, $x_{n,t+1}$ is not defined).
- $f(1) = 1$ and $f(N - 1) = N' - 1$ (where $N$ and $N'$ are the lengths of $\sigma$ and $\pi$, respectively).

It is clear that $f$ is well-defined and it is a parity respecting embedding between $\sigma$ and $\pi$. ◁

**Reverse implication** Suppose now that there exists a parity respecting embedding between $\sigma$ and $\pi$ and let $f$ be the associated injective mapping. We want to show that we have enough structure in the permutations to infer that there must be a clique of size $k$ in the graph $G$. In order to do so, we will prove the following sequence of claims that will restrict the map $f$.

**Claim A.2.** *Any parity respecting embedding $f$ from $\sigma$ to $\pi$ must map $w_i$ to $w'_i$, for $i \in \{1,2,3,4\}$.*

Proof. Since the pattern matching needs to respect parity, $f$ must map the $w_i$'s to even elements of $\pi$. Towards a contradiction, assume first that $f(w_i) \neq w'_j$, $i,j \in \{1,2,3,4\}$. That means that $f(w_i) = w'_{i',j}$ or $x'_{i',t}$, for some indices $i',j$ or $i',t$. But then, the odd elements to the right of $w_i$ in $\sigma$ cannot map to elements to the right of $f(w_i)$ in $\pi$ (as there would be at most 2 odd elements to the right of $f(w_i)$ and there are strictly more than 2 odd elements to the right of $w_i$), so $f$ cannot be an embedding of $\sigma$ into $\pi$. Finally, since both the $w_i$'s and the $w'_i$'s form 4231 subsequences, it is clear that there exists a unique way to embed the $w_i$'s into the $w'_i$'s, which is mapping each $w_i$ to its corresponding $w'_i$, for every $i \in \{1,2,3,4\}$. Thus, if $f(w_i) \neq w'_i$, $f$ cannot be an embedding. ◁

**Claim A.3.** *All the elements to the left (resp. to the right) of the $w_i$'s in $\sigma$ map to elements to the left (resp. to the right) of $w'_i$'s in $\pi$. Similarly, the elements above (resp. below) $w_2$ in $\sigma$ map to elements above (resp. below) $w'_2$ in $\pi$.*

Proof. This is a direct corollary of Claim A.2. ◁

**Claim A.4.** *Any parity respecting embedding $f$ from $\sigma$ to $\pi$ must map vertex blocks of $\sigma$ to vertex blocks of $\pi$.*

Proof. By Claim A.3, since elements to the left of $w_2$ in $\sigma$ map to elements to the left of $w'_2$ in $\pi$, we have that $f(v_{i,j}) = v'_{i',j'}$, for $i \in \{1,...,k\}, i' \in \{1,...,n\}$ and $j,j' \in \{1,2\}$. Assume that $f(v_{i,1}) = v'_{i',j'}$ and $f(v_{i,2}) = v'_{i'',j''}$, with $i' \neq i''$. Since $v_{i,1}$ is to the left of $v_{i,2}$, it means that $f$ must map $v_{i,2}$ to an element placed to the right of $f(v_{i,1}) = v'_{i',j'}$. But $v_{i,1} > v_{i,2}$ and every element which is to the right of $v'_{i',j'}$ and which does not belong to the vertex block of $v'_{i'}$, is greater than $v'_{i',j'}$. Thus, if $i'' \neq i'$, then $f$ would not be an embedding. ◁

**Claim A.5.** *Any parity respecting embedding $f$ from $\sigma$ to $\pi$ must map edge blocks of $\sigma$ to edge blocks of $\pi$.*

Proof. Again, we have that $f(e_{i,j}) = e'_{i',j'}$ for some pair $i',j'$, and since the structure of the gadget has the same properties as the vertex gadget, we can use the same argument as in the proof of Claim A.4. ◁

**Claim A.6.** *Any parity respecting embedding $f$ from $\sigma$ to $\pi$ must map $h_{i,j}$ to $h'_{i',j}$, where $e'_{i'}$ is the edge associated to the edge block where $e_{i,1}$ maps to.*

Proof. By Claim A.3, we have that necessarily, $f(e_{i,j}) = e'_{i',j}$, for $i \in \{1,...,l\}, i' \in \{1,...,m\}$ and $j \in \{1,2,3,4\}$.

First, since $f(e_{i,2}) = e'_{i',2}$ and $f(e_{i,3}) = e'_{i',3}$, and $f$ is an embedding, the fact that $h_{i,1}$ is in between $e_{i,2}$ and $e_{i,3}$ implies that it must map to an element between $e'_{i',2}$ and $e'_{i',3}$. Similarly, $h_{i,2}$ must map to an element in between $e'_{i',3}$ and $e'_{i',4}$. Since edge blocks map

to edge blocks, there is at most one element that satisfies each of these conditions. And these elements are $h'_{i',1}$ and $h'_{i',2}$, respectively. ◁

**Claim A.7.** *All the even elements to the right of the edge gadgets in $\sigma$ must map to even elements to the right of the edge gadgets in $\pi$.*

Proof. This follows from Claim A.2. Since $f(w_i) = w'_i$ for $i \in \{1, 2, 3, 4\}$ and $f$ has to respect parity, the rest of the even elements cannot map anywhere else. ◁

Now, suppose that there is a parity respecting embedding $f$ of $\sigma$ into $\pi$ and assume, towards a contradiction, that $G$ does not contain a clique of size $k$. Since there is no clique of size $k$, it means that we cannot have $l = \binom{k}{2}$ edges between the $k$ vertices of $G$ which are in the image of $f$ (that is, the vertices associated to the images of the $k$ vertex boxes of $\sigma$).

We know that the $k$ vertex blocks of $\sigma$ map to $k$ vertex blocks in $\pi$ and the $\binom{k}{2}$ edge blocks of $\sigma$ map to $\binom{k}{2}$ edge blocks of $\pi$. Since $G$ does not contain a clique, one of the $k$ vertices corresponding to the $k$ vertex blocks in the image of $f$ will have degree strictly smaller than $k-1$ when we restrict $G$ to the $k$ selected vertices. Let $i'$ be the vertex with degree strictly smaller than $k-1$ and suppose it is the image of vertex block $i$ in $\sigma$. Then, there are two possible cases. The first case is that in the image of $f$, between the values $f(v_{i,1})$ and $f(v_{i,2})$, there are less than $k$ odd elements (these elements are necessarily of the form $h'_{i,j}$). Since in between $v_{i,1}$ and $v_{i,2}$ in $\sigma$ there are $k$ odd elements of the form $h_{i,j}$, this would imply that $f$ cannot be a parity respecting embedding. The second possibility is that in between the values $f(v_{i,1})$ and $f(v_{i,2})$ there are $k$ odd elements (which again are necessarily of the form $h'_{i,j}$) but one of them is not in between $f(e_{l,2})$ and $f(e_{l,3})$, or $f(e_{l,3})$ and $f(e_{l,4})$, for some $l \in \{1, \ldots, m\}$. This would also contradict the fact that $f$ is a parity respecting isomorphism, as all the $h_{i,j}$ in $\sigma$ are between some pair $e_{l,2}, e_{l,3}$, or $e_{l,3}, e_{l,4}$ (with respect to the x-axis). Therefore, if there is a parity respecting embedding of $\sigma$ into $\pi$, it must map the $k$ vertex boxes of $\sigma$ into $k$ vertex boxes of $\pi$ associated to $k$ vertices that form a clique in $G$. □

**Corollary A.1.** *Given a pattern $\sigma \in S_k$ and a text $\pi \in S_n$, Parity PPM cannot be solved in time $f(k) \cdot n^{o(\sqrt{k})}$ for any computable function $f$, under the Exponential Time Hypothesis (ETH).*

*Proof.* Suppose that there exists an algorithm that solved Parity PPM in time $f(|\sigma|) \cdot n^{o(\sqrt{|\sigma|})}$. Therefore, since we have a reduction from $k$-Clique to Parity PPM that outputs instances with the parameter $|\sigma|$ bounded by $\mathcal{O}(k^2)$, Clique is solvable in time $f(k^2) \cdot n^{o(k)}$. But, Clique is not solvable in time $f(k) \cdot n^{o(k)}$ for any computable function $f$, under the Exponential Time Hypothesis, as proven in [37]. □

### A.3.2. Parameterized hardness for 4321-avoiding patterns

In this subsection, we extend the previous hardness result by showing that the problem remains hard for 4321-avoiding patterns. Our proof uses a colored version of PPM that was shown to be W[1]-hard parameterized by $k = |\sigma|$ in [82].

**Definition A.4.** *Given a 321-avoiding permutation $\sigma \in S_k$ and an arbitrary permutation $\pi$ such that both $\sigma$ and $\pi$ are 2-colored permutations, 2-colored 2IPP (2 Increasing Permutation Pattern) asks to find a color-preserving embedding of $\sigma$ into $\pi$.*

**Theorem A.2.** *Parity PPM is* W[1]*-hard parameterized by the length $k$ of the pattern, even if the pattern is 4321-avoiding.*

*Proof.* We reduce from 2-colored 2IPP. Given two 2-colored permutations $\sigma \in S_k$ and $\pi \in S_n$, with $\sigma$ 321-avoiding, we will construct a 4321-avoiding permutation $\sigma' \in S_{2k+1}$ and an arbitrary permutation $\pi' \in S_{2n+1}$ such that there exists a color-preserving embedding of $\sigma$ into $\pi$ if and only if there is a parity respecting embedding of $\sigma'$ into $\pi'$.

The high level idea is to build two permutations such that all the elements of color $c_1$ are embedded into even elements, all the elements of color $c_2$ are embedded into odd elements, and the remaining elements needed to have a valid permutation are placed to the right separated by a delimiter element.

We construct both permutations as follows. Let $c_1$ and $c_2$ be the two colors in the coloring of the original permutations. For every element $j$, if it is colored with $c_1$, we replace $j$ by $2j$ (it is now always even), and if it is colored with $c_2$, we replace it by $2j-1$ (it is now always odd). Then, we place the element $2k+1$ (resp. $2n+1$ for $\pi$) to the right of all these elements (the delimiter element). Finally, we place all the remaining elements that have not appeared (i.e., $2j-1$ for every element $j$ colored with $c_1$, and $2j$ for every $j$ colored with $c_2$) to the right of $2k+1$ forming an increasing subsequence (these elements are needed to construct a valid permutation). It is clear that this construction can be carried out in polynomial time and that the new parameter $k' = |\sigma'| = 2k+1$ is a function of $k$. Furthermore, $\sigma'$ is 4321-avoiding. Indeed, the left part is order isomorphic to $\sigma$ and the right part contains decreasing subsequences of length at most two, formed by the element $2k+1$ and an element of the increasing subsequence to its right. Thus, the maximum length of a decreasing subsequence is 3: a decreasing subsequence of length two of the left part followed by a smaller element from the increasing subsequence placed to the right.

To illustrate this construction, suppose that we have the permutations

$$\sigma = 2\ 1\ 3\ 4\ 6\ 5 \text{ and } \pi = 1\ 3\ 2\ 4\ 5\ 8\ 7\ 6$$

colored with $c_1$ = blue and $c_2$ = black. Then

$$\sigma' = 4\ 1\ 6\ 7\ 12\ 9\ 13\ 2\ 3\ 5\ 8\ 10\ 11 \text{ and } \pi' = 1\ 6\ 3\ 8\ 9\ 16\ 13\ 11\ 17\ 2\ 4\ 5\ 7\ 10\ 12\ 14\ 15$$

where 13 and 17 are our delimiters here (see also Figure A.2).

We will first show that if there exists a color-preserving embedding of $\sigma$ into $\pi$, then there is a parity respecting embedding of $\sigma'$ into $\pi'$. Let $f : [\sigma] \longrightarrow [\pi]$ be the injective mapping associated to the color-preserving embedding. We construct the parity respecting embedding $f' : [\sigma'] \longrightarrow [\pi']$ as follows. For an element $j$, we denote by $P_\sigma(j)$ the position of $j$ in the permutation $\sigma$, and given an index $i$, we denote by $\sigma[i]$ the value of the element in position $i$.

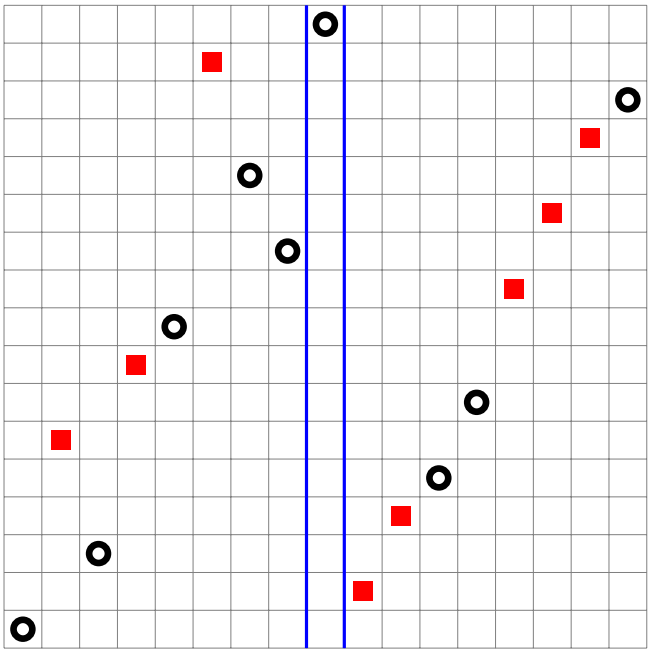


Figure A.2.: The 4321-avoiding permutation $\pi' = 1\ 6\ 3\ 8\ 9\ 16\ 13\ 11\ 17\ 2\ 4\ 5\ 7\ 10\ 12\ 14\ 15$ from the initial 2-colored 321-avoiding permutation $\pi = 1\ \underline{3}\ 2\ \underline{4}\ 5\ \underline{8}\ 7\ 6$. Red squares correspond to even elements and black dots to odd elements, while the two blue lines are used to indicate the presence of the element that serves as the delimiter.

$$f'(i) = \begin{cases} f(i) & if \quad i \leqslant k \\ n+1 & if \quad i = k+1 \\ P_{\pi'}(\pi'[f'(P_{\sigma'}(\sigma'[i]+1))]-1) & if \quad i > k+1 \text{ and } i \text{ is odd} \\ P_{\pi'}(\pi'[f'(P_{\sigma'}(\sigma'[i]-1))]+1) & if \quad i > k+1 \text{ and } i \text{ is even} \end{cases} \tag{A.4}$$

First of all, note that if we have an even element $2j$ to the right of the delimiter, that means that the element $2j-1$ must be to its left. Similarly, if the element $2j-1$ is to the right of the delimiter, then $2j$ must be on the left. This is because given an element $j$ of the original permutation, we either transform it into $2j$ or into $2j-1$, according to its color, and since elements are unique in value, we cannot have both to the left of the delimiter. We call $2j$ the *pair* of $2j-1$ (and vice versa).

It is clear that $f'$ is well defined and that it is a parity respecting embedding. Indeed, the elements $j < k+1$ and $j < n+1$ in $\sigma'$ and $\pi'$ are order isomorphic to $\sigma$ and $\pi$, respectively, and even elements correspond to one color and odd elements to another, so the fact that there is a parity respecting embedding of this part follows from the assumption that there is a color preserving embedding of $\sigma$ into $\pi$. On the other hand, since there is a parity respecting embedding of the elements to the left of the delimiter of $\sigma'$ into the elements to the left of the delimiter of $\pi'$, this implies that there also exists a parity respecting embedding of the right part. Indeed, for every element $j$ to the right in $\sigma'$, we can look where its pair $j'$ maps to in $\pi'$ and then map $j$ to the pair of $f'(j')$. Since every element has a pair and $f'$ defines a parity respecting embedding between the left parts of both permutations, then this extension of $f'$ to the right part is well defined and respects parity as well. Furthermore, it is clear that it is also an embedding, because if an element $j$ is greater than $i$, then the pair of $j$ is also greater than the pair of $i$.

For the converse implication, let us assume that there is a parity respecting embedding $f'$ of $\sigma'$ into $\pi'$. It is enough to see that every element $j$ with $P_{\sigma'}(j) < k+1$ needs to map to an element $j'$ with $P_{\pi'}(j) < n+1$. Indeed, suppose that there exists an element in position $j < k+1$ in $\sigma$ such that $f'(j) \geqslant n+1$. Then the element $2k+1$ of $\sigma$, which is to the right of $j$, cannot map anywhere, as it is greater than $j$ and followed by a smaller element (i.e., we need to find a decreasing subsequence of size two, with the first element greater than $j'$). But all the elements to the right of $n+1$ in $\pi'$ form an increasing subsequence, so $f'$ would not be an embedding. Thus, the elements to the left of $2k+1$ in $\sigma'$ must map to the elements to the left of $2n+1$ in $\pi'$. But these elements are order isomorphic to $\sigma$ and $\pi$, respectively, and the even elements correspond to the ones colored with $c_1$ and the odd ones to the ones colored with $c_2$, so the fact that there is a parity respecting embedding of $\sigma'$ into $\pi'$ implies that there is a color-preserving embedding of $\sigma$ into $\pi$.

□

### A.3.3. Parameterized algorithm when the text avoids a fixed pattern

In the previous subsection, we showed that restricting the pattern does not necessarily reduce the complexity of the problem. However, we now prove that restricting the text allows us to use the twin-width meta-theorem [22] to obtain a positive result. In fact, to see that Parity PPM is FPT if the text avoids a fixed pattern $x$, it suffices to show that we can describe the problem using first-order (FO) logic, i.e., that we can express it as a formula which uses quantified variables over non-logical objects, and sentences (formulas without free variables) that contain the variables. Indeed, adding unary relations to mark the odd and even values preserves bounded twin-width, and therefore FPT tractability. The result follows from [22]:

**Lemma A.1** ([22])**.** *FO model checking is* FPT *on every hereditary proper subclass of permutation graphs.*

This implies that FO model checking is FPT in the class of permutations avoiding a fixed pattern. Here, FO model checking refers to the problem of, given a first-order sentence $\phi$ of FO and a finite model $\mathcal{M}$ of FO (which specifies the domain of disclosure of the variables), deciding whether $\mathcal{M}$ satisfies $\phi$, i.e., whether there exists an assignment of the variables which respects the domain imposed by $\mathcal{M}$ and that satisfies $\phi$. Therefore, we can state the following theorem:

**Theorem A.3.** *Parity PPM is* FPT *if the text $\pi$ avoids a fixed permutation.*

*Proof.* As stated above, we only need to express Parity PPM using FO logic.

Let us consider the permutations $\sigma = \sigma_1 \ \dots \ \sigma_k$ and $\pi = \pi_1 \ \dots \ \pi_n$. Let $\mathcal{U} = [n]$ be the universe. We receive as input $\sigma$, $\pi$ and the following relations:

- A binary relation $\leqslant_{\pi_1}$, which indicates the left-to-right ordering of the elements in $\pi$ (i.e., the ordering of the elements with respect to their position in $\pi$).
- A binary relation $\leqslant_{\pi_2}$, which indicates the ordering of the elements of $\pi$ according to the natural linear ordering.
- A unary relation $\mathcal{R}$ such that $u \in \mathcal{R}$ if and only if $u$ is in an even position when ordered by $\leqslant_{\pi_2}$.

We define the FO logic formula $\exists\, x_1, \dots, x_k\ (\phi)$, where $x_1, \dots, x_k$ represent the images of $\sigma_1, \dots, \sigma_k$ under a parity respecting embedding of $\sigma$ into $\pi$, and where $\phi$ is the conjunction of the following clauses:

- For every pair $i, j$, with $1 \leqslant i < j \leqslant k$, $\sigma_i \leqslant \sigma_j$, we add the clause $\psi_{ij} = x_i \leqslant_{\pi_2} x_j$.
- For every pair $i, j$, with $1 \leqslant i < j \leqslant k$, $\sigma_i > \sigma_j$, we add the clause $\neg\psi_{ij}$.
- The clause $x_1 \leqslant_{\pi_1} x_2 \leqslant_{\pi_1} \dots \leqslant_{\pi_1} x_k$.
- For every $i \in \{1, \dots, k\}$ such that $\sigma_i$ is even, we add the clause $x_i \in \mathcal{R}$.

The first three types of clauses model the definition of an embedding of $\sigma$ into $\pi$, while the latter defines the parity constraint.

Since the problem can be expressed using FO logic, it follows by Lemma A.1 that Parity PPM is FPT if $\pi$ avoids a fixed pattern.

□

## A.4. Classical complexity

### A.4.1. Hardness

A nice quite recent result showed that PPM remains NP-hard, even if the pattern is 321-avoiding and the text is 4321-avoiding [93]. In the following, we show that it remains true for Parity PPM.

**Theorem A.4.** *Parity PPM is* NP*-hard, even if $\sigma$ is a* 321*-avoiding permutation and $\pi$ is a* 4321*-avoiding permutation.*

*Proof.* We reduce from PPM for 321-avoiding pattern and 4321-avoiding text, which was proven to be NP-hard in [93]. Let $\pi \in S_n$ and $\sigma \in S_k$ be two arbitrary 4321-avoiding and 321-avoiding permutations, respectively. We construct a 4321-avoiding permutation $\pi' \in S_{2n}$ and a 321-avoiding permutation $\sigma' \in S_{2k}$ such that $\sigma \preceq \pi$ if and only if $\sigma' \preceq_P \pi'$ (i.e., there is a parity respecting embedding of $\sigma'$ into $\pi'$). The idea is to construct two permutations that have an embedding of the original permutations into its even elements and such that its odd elements are placed in a manner that does not create 321 (resp. 4321) subsequences.

We construct $\sigma'$ as follows. We first multiply each element $\sigma(i)$ by two, for every $i \in \{1, \ldots, k\}$. Then, we place each (missing) odd element $j \in \{3, ..., 2k-1\}$ to the right of $j-1$ and we let 1 be the leftmost element of the permutation. The text $\pi'$ is also constructed following the same procedure. This construction is clearly done in polynomial time.

**Claim A.8.** *The constructed permutations $\sigma'$ and $\pi'$ are indeed* 321*-avoiding and* 4321*-avoiding, respectively.*

Proof. We will only prove it for $\sigma'$ as the argument for $\pi'$ is the same. Suppose towards a contradiction that there is a 321 subsequence in $\sigma'$. Since $\sigma$ is 321-avoiding and there is an embedding of $\sigma$ into the even elements of $\sigma'$ (by construction), we have that there must be at least one odd element in the 321 subsequence. We will prove that we can replace each of the odd elements $j$ of the 321 subsequence by the even element $j-1$ and still preserve said decreasing subsequence, which contradicts the fact that the original permutation $\sigma$ was 321-avoiding. Indeed, if an odd element $j$ belongs to a 321 subsequence, then its left neighbour $j-1$ cannot be part of this subsequence, as $j-1 < j$. Thus, any element to the left of $j$ in the subsequence is also to the left of $j-1$ in $\sigma'$, and it has to be greater than both $j$ and $j-1$. On the other hand, every element to the right of $j$ is trivially to the right of $j-1$, and since there are no values

between $j-1$ and $j$, every element smaller than $j$ in a 321 subsequence, is also smaller than $j-1$. Both conditions imply that we could replace the odd element $j$ by $j-1$ in the subsequence and preserve the existing decreasing subsequence of length 3, which contradicts the assumption that $\sigma$ is 321-avoiding. ◁

To finish the proof, we will show that $\sigma' \leqslant_P \pi'$ if and only if $\sigma \leqslant \pi$. To see that if $\sigma' \leqslant_P \pi'$ then $\sigma \leqslant \pi$, notice first that there is an embedding of $\sigma$ into the even elements of $\sigma'$ and an embedding of $\pi$ into the even elements of $\pi'$. Thus, since there is a parity respecting embedding of $\sigma'$ into $\pi'$, then the even elements of $\sigma'$ are mapped to $\pi'$, which implies that, by construction, there is an embedding of $\sigma$ into $\pi$.

For the converse implication, assume that there is an order isomorphism from $\sigma$ to $\pi$ and let $f : [k] \longrightarrow [n]$ be the associated injective mapping. Then, we build $f' : [2k] \longrightarrow [2n]$ as follows:

$$f'(i) = \begin{cases} 1 & if \quad i = 1 \\ 2\,f\left(\frac{i}{2}\right) & if \quad i \text{ is even} \\ 2\,f\left(\frac{i-1}{2}\right)+1 & if \quad i \text{ is odd} \end{cases} \tag{A.5}$$

It is clear that $f'$ is well defined and it is an order isomorphism. Indeed, $f$ is an order isomorphism from the even (resp. odd) elements of $\sigma'$ to the even (resp. odd) elements of $\pi'$.

□

### A.4.2. Polynomial-time solvable cases

For some specific cases of Permutation Pattern Matching, polynomial time algorithms that solve the problem exactly have been proposed. Here, we show that some of these algorithms can be adapted to solve the problem Parity Permutation Pattern Matching while still running in polynomial time.

**Theorem A.5.** *Let $\sigma$ be a permutation in $S_k$ and $\pi$ be a permutation in $S_n$. Parity PPM can be solved in polynomial time in the following cases:*

1. *The pattern $\sigma$ is separable. In particular, if both permutations are (231, 213)-avoiding, it can be solved in linear time.*

2. *Both permutations are 321-avoiding.*

*Proof.* 1. For separable patterns, we adapt the algorithm by Bose et al. [25]. As in the original algorithm, we use the separating tree $\mathcal{T}$ of the pattern $\sigma$, and we solve one subproblem for each node $V$ and for each substring $(\pi_i, ..., \pi_j)$ of $\pi$ and each subrange $a, a+1, ..., b$ of the range $1, ..., n$. For each subproblem, we count the number of matches $M(V, i, j, a, b)$ of the pattern $\sigma_l, ..., \sigma_m$ (the substring corresponding to the subtree rooted at $V$) into text $\pi_i, ..., \pi_j$, using $\pi_i$, and using

text values in the range $a, \ldots, b$ (i.e., $a \leqslant \pi_k \leqslant b$ for all $k$ with $i \leqslant k \leqslant j$), including $a$.

The solutions to the subproblems for the leaves of the tree $\mathcal{T}$ are immediately obtained: we have a parity match if and only if there is a regular match and the two elements have the same parity, that is, $M(V_0, i, j, a, b) = 1$ if and only if $\pi_i = a$ and the parity of the element corresponding to leaf $V_0$ is the same as the parity of $\pi_i$.

The way to compute $M(V, i, j, a, b)$ based on the values of $M$ for the children of node $V$ is exactly the same as in the original algorithm. The correctness of the algorithm follows directly from the fact that we can view the parity respecting embeddings of $\sigma$ into $\pi$ as a subset of the general embeddings.

Since we only introduce an operation on the leaves that can be carried out in constant time, the running time of this modified algorithm is still $\mathcal{O}(kn^6)$.

In particular, if both $\sigma$ and $\pi$ are (231, 213)-avoiding permutations, we can adapt the algorithm for general PPM by Neou et al. [122], which still runs in linear time for the parity version. Indeed, instead of considering a bijection between the set of wedge permutations of size $n$ and the set of binary words of size $n-1$, we can consider a bijection between the set of wedge permutations of size $n$ and the set of quaternary words, that is, words over an alphabet of size 4. We follow the same notation as in [122] and we say that an element $\pi_i$ is a right-to-left maximum (RLMax) of $\pi$ if it is the topmost element of $\pi_i \ldots \pi_n$, and a right-to-left minimum (RLMin) if and only if it is the bottommost element of $\pi_i \ldots \pi_n$. Then, given a subsequence $s$, the quaternary word $B(s)$ is the word where the letter at position $i$ indicates whether $s[i]$ is an even RLMax element, an odd RLMax element, an even RLMin element or an odd RLMin element. To solve the problem, we can follow the same procedure and read both $B(\sigma)$ and $B(\pi)$ from left to right, and when two letters are equal, we match them and move to the next in both words (if they exist). Otherwise we stay at the same letter of $B(\sigma)$ but move to the next one in $B(\pi)$ (if it exists). We accept exactly when all letters of $B(\sigma)$ have been matched. This algorithm runs in linear time.

2. For 321-avoiding permutations, we adapt the polynomial-time algorithm for general PPM given by Albert et al. in [4] and show that it can be extended to the parity version. Following the notation of the paper, we define the set of upper and lower elements of $\pi$, $U_\pi$ and $L_\pi$, respectively. An element $x$ belongs to $U_\pi$ (resp., $L_\pi$) if there exists an embedding of 21 into $\pi$ such that $x$ is the image of 2 (resp., 1). The union of upper and lower elements forms the set of rigid elements, and we say that a permutation is rigid if all of its elements are rigid, while elements that are neither upper or lower are referred to as fluid, and a permutation is said to be fluid if it contains fluid elements. We also say that a map $f : \sigma \longrightarrow \pi$ is a rigid mapping if $f$ maps upper (resp., lower) elements of $\sigma$ to upper (resp., lower) elements of $\pi$. Furthermore, we define $x^{\blacktriangleleft}_{T(x),p(x)}$ as the rightmost element that is to the left of $x$ and has the same type (i.e., upper or lower) and parity as $x$, and similarly for the

other directions (right, up and down). We refer the reader to [4] for more detailed definitions.

As in the original paper, we first see how to adapt the algorithm for rigid permutations, and then extend it for fluid elements. The main result that the algorithm makes use of gets transformed into:

**Proposition A.6.** *Suppose that $e : \sigma \longrightarrow \pi$ is a parity respecting embedding, $f : \sigma \longrightarrow \pi$ is a rigid mapping respecting parity, and, for all $x \in \sigma$, $f(x) \leqslant e(x)$. Then, for all $x \in \sigma$:*

$$\max\left\{f(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x),p(x)}, f(x^{\blacktriangledown})^{\blacktriangle}_{T(x),p(x)}\right\} \leqslant e(x)$$

Proof. Let us first see that $f(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x),p(x)} \leqslant e(x)$. Since $x$ lies strictly to the right of $x^{\blacktriangleleft}$ (and $e$ is a parity respecting embedding), $e(x)$ lies strictly to the right of $e(x^{\blacktriangleleft})$, and is of the same type and parity as $x$, so it cannot lie to the left of $e(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x),p(x)}$. Thus, $e(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x),p(x)} \leqslant e(x)$. But $f(x^{\blacktriangleleft}) \leqslant e(x^{\blacktriangleleft})$ and so $f(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x),p(x)} \leqslant e(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x),p(x)} \leqslant e(x)$.
The other case is analogous. ◁

We say that element $x$ is a problem if $f(x) < \max\left\{f(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x),p(x)}, f(x^{\blacktriangledown})^{\blacktriangle}_{T(x),p(x)}\right\}$ and we denote by $P(f)$ the set of problems for $f$.

**Corollary A.2.** *Let $\sigma$ be a rigid permutation and $\pi$ a 321-avoiding permutation. A rigid mapping $f$ respecting parity is a parity respecting embedding of $\sigma$ into $\pi$ if and only if the set of problems $P(f)$ is empty.*

Proof. The proof is analogous to the one on [4] replacing $f(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x)}$ by $f(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x),p(x)}$.
If $f$ is a parity respecting embedding, it follows from the proposition above that no element $x \in \sigma$ violates the condition, and thus $P(f)$ is empty. For the converse implication, assume that $f$ is not a parity respecting embedding. Then, there exists $x \in \sigma$ such that $f(x)$ is below or equal to $f(x^{\blacktriangledown})$ or such that $f(x)$ is left or equal to $f(x^{\blacktriangleleft})$. In the first case, we would have that $f(x)$ is strictly below $f(x^{\blacktriangledown})^{\blacktriangle}_{T(x),p(x)}$ and so $x \in P(f)$. In the second case, if $f(x)$ is left or equal to $f(x^{\blacktriangleleft})$, then $f(x)$ is strictly left of $f(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x),p(x)}$. Now, for $f(x), f(y) \in \sigma$ of the same type and parity, $f(x)$ is left of $f(y)$ if and only if $f(x)$ is below $f(y)$. And since $f$ preserves types and parity, $f(x)$ and $f(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x),p(x)}$ have the same type and parity, so $f(x) < f(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x),p(x)}$, which implies $x \in P(f)$. ◁

We now describe the adaptation of the original algorithm, which given a rigid permutation $\sigma$ and a 321-avoiding permutation $\pi$ as input, returns the minimum parity respecting embedding $e_{min}$ of $\sigma$ into $\pi$ if it exists and fails otherwise. Note that there exists a minimum parity respecting rigid embedding $e_{min}$ since these rigid embeddings form a distributive lattice. Let $f_0$ be the map that sends all the even elements of $U_\sigma$ to the least even element of $U_\pi$ and all the odd elements of $U_\sigma$ to the least odd element of $U_\pi$, and similarly for $L_\sigma$ and $L_\pi$. It is clear that $f_0$ is a

parity respecting rigid mapping of $\sigma$ into $\pi$. The algorithm is shown in Algorithm 1.

**Algorithm 2** Algorithm

Initialise $f$ as $f_0$
Compute $P(f)$
**while** $f$ is defined everywhere and $P(f)$ is not empty **do**
  Choose $x \in P(f)$
  Update:
  $f(x) \longleftarrow \max\{f(x^{\blacktriangleleft})^{\blacktriangleright}_{T(x),p(x)}, f(x^{\blacktriangledown})^{\blacktriangle}_{T(x),p(x)}\}$
  Recompute $P(f)$
**end while**
**return** $f$

The proof of correctness and run-time analysis of the modified algorithm are analogous to those of the original one. Thus, the running time remains $\mathcal{O}(kn)$.

Finally, we extend the algorithm to fluid elements, using the fact that any 321-avoiding permutation has a unique decomposition as a direct sum, where each term is either rigid or a singleton. Given a parity respecting embedding $e_i$ of $\sigma_1 \bigoplus \sigma_2 \bigoplus \cdots \bigoplus \sigma_i$ into $\pi$, we will construct a pair of parity respecting embeddings of $\sigma_1 \bigoplus \sigma_2 \bigoplus \cdots \bigoplus \sigma_{i+1}$ into $\pi$ such that at least one extends to a parity respecting embedding of $\sigma$ into $\pi$. Let $T_i$ be the set of elements that lie above and to the right of the image of $e_i$. Then, the image of $e$ restricted to the elements corresponding to $\sigma_{i+1}$ is contained in $T_i$. There are three cases:

- $\sigma_{i+1}$ is rigid: in this case, we use the algorithm above to find the minimal parity respecting embedding of $\sigma_{i+1}$ into $T_i$.
- $\sigma_{i+1}$ is a singleton and $T_i$ restricted to the elements with the same parity begins with its least element: we map $\sigma_{i+1}$ to that element.
- $\sigma_{i+1}$ is a singleton and the first element of $T_i$ with the same parity is not its minimum: then we can extend $e_i$ in two ways, either to the leftmost element of $T_i$ with the same parity or to the minimum element with the same parity. Note that given two partial embeddings of $\sigma_1 \bigoplus \sigma_2 \bigoplus \cdots \bigoplus \sigma_i$, they might extend to three or four candidate embeddings of $\sigma_1 \bigoplus \sigma_2 \bigoplus \cdots \bigoplus \sigma_{i+1}$, but we only need to keep the ones where the image of the singleton $\sigma_{i+1}$ is minimal within either $U_\pi$ or $L_\pi$.

The running time of the modified algorithm is again $\mathcal{O}(kn)$, as checking the parity of the elements can be done in constant time.

□

As a final remark, note that other polynomial-time algorithms for specific cases of PPM might also be adaptable for Parity Permutation Pattern Matching. For example, the algorithm of Berendsohn et al. that formulates PPM as a CSP instance [16],

with runtime $O(n^{tw(\sigma)+1})$, where $tw(\sigma)$ is the tree-width of the incidence graph of $\sigma$, can also be modified for the parity-respecting case by restricting the domain of the variables of the CSP problem so that the domain of each variable contains only the indices corresponding to elements of the same parity (note that this algorithm is more general, but can have a worse running time in the specific cases studied here, as the tree-width of separable permutations can be as high as 7 [3], and the tree-width of 321-avoiding permutations is not upper-bounded by a constant [16]).

# B. Hardness of Balanced Mobiles

Measuring tree dissimilarity and studying the shape of trees are important tasks in phylogenetics. One of the most studied shape properties is the notion of tree imbalance, which can be quantified by different indicators, such as the Colless index. Here, we study the generalization of the Colless index to mobiles, i.e., full binary trees in which each leaf has been assigned a positive integer weight. In particular, we focus on the problem Balanced Mobiles, which given as input $n$ weights and a full binary tree on $n$ leaves, asks to find an assignment of the weights to the leaves that minimizes the Colless index, i.e., the sum of the imbalances of the internal nodes (computed as the difference between the total weight of the left and right subtrees of the node considered). We prove that this problem is strongly NP-hard, answering an open question given at IWOCA 2016.

## B.1. Introduction

Phylogenetics is the study of evolutionary relationship among biological entities (taxa). Its main task is to infer trees whose leaves are bijectively labeled by a set of taxa and whose patterns of branching reflect how the species evolved from their common ancestors (*phylogenetic trees*). The inferred trees are often studied by comparing them to other phylogenetic trees or to existing models. Thus, it is important to be able to formally quantify how different trees differ from each other and to have measures that give information about the shape of the trees. With respect to the latter, one of the most studied shape properties of phylogenetic trees is that of *tree balance*, measured by metrics such as the Sackin index [137] or the Colless index [40] (see also the survey of Fischer et al. [61]). The Colless index is defined for binary trees as the sum, over all internal nodes $v$ of the tree, of the absolute value of the difference of the number of leaves in the two children of $v$. It is one of the most popular and used metrics, see for example [14, 121, 46, 19, 140].

The natural generalization with *weights* on the leaves has later been studied within *mobiles*[1], defined as full binary trees with positive weights on their leaves. In particular, given a set of $n$ integer weights $\{w_1, \ldots, w_n\}$, the problem Balanced Mobiles asks to find a mobile whose $n$ leaves have weights $w_1, \ldots, w_n$, and which minimizes the total Colless index (i.e., the sum of the imbalances $|x - y|$ of every internal node, where $x$ and $y$ represent the total weight of the leaves on the left and right subtrees of the node considered) [89]. Despite being a natural generalization, the complexity of this problem

[1] The term "mobile" comes from what can be found in modern art (e.g. the ones of Calder, well known in TCS being the illustration of the cover of the famous book CLRS' Introduction to Algorithms [42]) or the toy above toddler beds [89].

is still not yet known. In fact, it was proposed as an open problem by Hamoudi, Laplante and Mantaci in IWOCA 2016 [88].

Still, some results are known for some specific cases. For example, if all the leaves have unit weight, it is known that building a partition tree or a left complete tree are both optimal solutions, and their imbalance can be computed in polynomial time using a recursive formula. On the other hand, if all the weights are powers of two or if a perfectly balanced mobile can be constructed, the well known Huffman's algorithm [90] is optimal. This algorithm recursively builds a mobile by grouping the two smallest weights together (where the weight of the constructed subtree is added to the list of weights in each step).

With respect to the complexity, it is only known that the problem is in the parameterized class XP, parameterized by the optimal imbalance [89] (i.e. it is polynomial for constant values of the parameter). This result was obtained by using a relaxation of Huffman's algorithm, which gives an algorithm of complexity $\mathcal{O}(\log(n)n^{C^*})$, where $C^*$ is the optimal imbalance. An ILP is also given to solve the problem [89]. However, no polynomial time approximation algorithm has been proposed for this problem, although it is known that Huffman's algorithm does not construct an approximate solution in the general case, being arbitrarily far away from the optimum for some instances [89].

In this paper, we shed some light into the complexity of the problem by showing that Balanced Mobiles is strongly NP-hard when both the full binary tree and the weights are given as input.

## B.2. Preliminaries

We first give the necessary definitions to present the problem.

**Definition B.1.** *A* full binary tree *is a rooted tree where every node that has at least one child has precisely two children. A full binary tree is said to be* perfect *when all its leaves are at the same depth. The depth $d(v)$ of a node $v$ is defined by*

$$d(v) := \begin{cases} 0 \text{ if } v = r, \text{ the root,} \\ 1 + d(F(v)) \text{ otherwise,} \end{cases}$$

*where $F(v)$ denotes the father of node $v$. Also, for every non-leaf node $v$, $L(v)$ (resp., $R(v)$) denotes the left (resp., right) child of node $v$.*

**Definition B.2.** *A binary tree is said to be* leaf-weighted *when a natural number $w(v)$ is assigned to each one of its leaf nodes $v$. Then, the recurrence $w(v) := w(L(v)) + w(R(v))$ extends $w$ defining it also on every internal node $v$ as the total weight over the leaves of the subtree rooted at $v$. A leaf-weighted full binary tree is also called a* mobile.

In this paper, we focus only on the Colless index to measure the balance of mobiles. Thus, we will just refer to the cost at each node as *imbalance*, and to the total Colless index of the tree as the *total cost*.

**Definition B.3.** *The* imbalance *of an internal node $v$ is defined as $imb(v) := |w(L(v)) - w(R(v))|$. The* total cost *of a leaf-weighted full binary tree (mobile) is the sum of the imbalances over the internal nodes. If the total cost is equal to 0, the mobile is said to be* perfectly balanced.

We can now define the problem Balanced Mobiles studied in this paper.

Balanced Mobiles

| | |
|---|---|
| ***Instance:*** | $n$ natural numbers and a full binary tree $\mathcal{T}$ with $n$ leaves. |
| ***Goal:*** | Assign each number to a different leaf of the given full binary tree in such a way that the sum of the imbalance over the internal nodes of the resulting leaf-weighted binary tree is minimum. |

## B.3. Balanced Mobiles is NP-hard in the strong sense

We prove that Balanced Mobiles as formulated above is NP-hard in the strong sense.

To do so, we will reduce from ABC-partition, a variant of the problem 3-partition which we define below.

ABC-partition

| | |
|---|---|
| ***Instance:*** | A target integer $T$, three sets $A, B, C$ containing $n$ integers each such that the total sum of the $3n$ numbers is $nT$. |
| ***Goal:*** | Construct $n$ triplets, each of which contains one element from $A$, one from $B$ and one from $C$, and such that the sum of the three elements of each triplet is precisely the target value $T$. |

The ABC-partition problem is known to be strongly NP-hard, that is, it is NP-hard even when restricted to any class of instances in which all numbers have magnitude $\mathcal{O}(poly(n))$. This fact is reported in [70], where the problem, labeled as [SP16], is also called Numerical 3-D Matching, as it can also be reduced to the 3-Dimensional Matching problem.

### B.3.1. Preliminary steps on the ABC-partition problem

As a first step in the reduction, given an instance of ABC-partition, we will reduce it to an equivalent instance of the same problem with some specific properties that will be useful for the final reduction.

A class of instances is called *shallow* when it comprises only instances all of whose numbers have magnitude $\mathcal{O}(poly(n))$. Since we aim at proving strong NP-hardness of the target problem, we need to make sure that, starting from any shallow class of instances, the classes of instances produced at every step remain shallow.

We start with some easy observations.

**Observation B.1.** *For any natural constant $k$, we can assume that all numbers are divisible by $2^k$, simply by multiplying all of them by $2^k$.*

Note that, since $k$ is a constant, after this first reduction we are still dealing with a shallow class of instances.

For the next step, we assume all numbers are greater than 1, which follows from the above with $k = 1$.

**Observation B.2.** *We can then assume that $n$ is a power of two, otherwise, let $h$ be the smallest natural such that $2^h > n$, we can just add $2^h - n$ copies of the number $T - 2$ to the set $A$, and $2^h - n$ copies of the number 1 to both sets $B$ and $C$.*

Note that we are still dealing with a shallow class of instances.

The next step requires to be more formal. Assume the three given sets of natural numbers to be $A = \{a_1^0, a_2^0, \ldots, a_n^0\}$, $B = \{b_1^0, b_2^0, \ldots, b_n^0\}$ and $C = \{c_1^0, c_2^0, \ldots, c_n^0\}$, the target value to be $T^0$ and let $M^0$ be the maximum number in $A \cup B \cup C$. Here, $M^0 = \mathcal{O}(poly(n))$ since this generic instance is taken from a shallow class. Consider the instance of the problem where the $n$ input numbers and the target value $T^0$ are transformed as follows:

$$
\begin{aligned}
a_i^1 &:= a_i^0 + 8n^2M^0 && \text{for every } i = 1, 2, ..., n \\
b_i^1 &:= b_i^0 + 4n^2M^0 && \text{for every } i = 1, 2, ..., n \\
c_i^1 &:= c_i^0 + 2n^2M^0 && \text{for every } i = 1, 2, ..., n \\
T^1 &:= T^0 + 14n^2M^0 \\
M^1 &:= T^1
\end{aligned}
$$

Notice that the new $M^1$ does not represent any longer the maximum value of the numbers of the input instance because it is equal to $T^0 + 14n^2M^0$, while the value of ever number is bounded above by $a_i^0 + 8n^2M^0$. The role that parameter $M^1$ plays in our reduction will be seen only later.

Clearly, this new instance is equivalent to the previous one in the sense that either both or none of them are yes-instances of ABC-PARTITION. Moreover, this reduction yields a shallow class of instances $\mathcal{C}^1$ when applied to a shallow class of instances $\mathcal{C}^0$. Therefore, thanks to Observation B.2 and this transformation, we can assume that we are dealing with a shallow class of instances each of which satisfies the following two properties:

$$n \text{ is a power of two} \tag{B.1}$$

$$b > c \text{ and } a > b + c \text{ for every } a \in A,\ b \in B \text{ and } c \in C. \tag{B.2}$$

### B.3.2. ABCDE-partition problem

Once Properties (B.1) and (B.2) are in place, we create a next equivalent instance through one further reduction, this time yielding an instance of a slightly different version

of the multi-dimensional partition problem, the ABCDE-partition problem, which we define below.

ABCDE-Partition

***Instance:*** A target integer $T$, five sets $A, B, C, D, E$, with $n$ integers in each, such that the sum of the numbers of all sets is $nT$.

***Goal:*** Construct $n$ 5-tuples, each of which contains one element of each set, with the sum of these five elements being precisely $T$.

If not known, the next transformation in our reduction proves that this variant is also strongly NP-hard. In fact, where $a_i^1, b_i^1, c_i^1, M^1, T^1$ comprise the modified input of the ABC-partition problem after the last transformation detailed just above, consider the equivalent instance of the ABCDE-partition problem where the input numbers and the target value are defined as follows:

$$
\begin{aligned}
a_i &:= a_i^1 + 8n^2M^1 && \text{for every } i = 1, 2, ..., n \\
b_i &:= b_i^1 + 4n^2M^1 && \text{for every } i = 1, 2, ..., n \\
c_i &:= c_i^1 + 2n^2M^1 && \text{for every } i = 1, 2, ..., n \\
d_i &:= n^2M^1 && \text{for every } i = 1, 2, ..., n \\
e_i &:= n^2M^1 && \text{for every } i = 1, 2, ..., n \\
T &:= T^1 + 16n^2M^1 \\
M &:= M^1
\end{aligned}
$$

Notice that, once again, the new $M$ parameter does not represent the maximum value of the numbers comprising the new input instance. In fact, it is significantly smaller.

Thanks to this last transformation, we see that the ABCDE-Partition problem is NP-hard even when restricted to a shallow class of instances each of which satisfies the following three properties:

$$n \text{ is a power of two (say } n = 2^h) \tag{B.3}$$

$$\text{the numbers in } D \cup E \text{ are all equal} \tag{B.4}$$

$$c > d + e, b > c + d + e \text{ and } a > b + c + d + e, \text{ for every } (a, b, c, d, e) \in A \times B \times C \times D \times E \tag{B.5}$$

This instance also possesses other useful properties that we will exploit in the reduction to the Balanced Mobiles problem we are going to describe next.

### B.3.3. Reduction to Balanced Mobiles

To the above instance $(T, A, B, C, D, E)$ of ABCDE-Partition, we associate the following instance $(\mathcal{T}, W)$ of Balanced Mobiles:

**Weights ($W$).** Besides the weights $a_i, b_i, c_i, d_i, e_i$ defined above for every $i = 1, 2, ..., n$, we also introduce $n = 2^h$ copies of the number $T$. Notice that all these numbers have magnitude $\mathcal{O}(poly(n))$ since it was assumed that $M = \mathcal{O}(poly(n))$.

**Full binary tree ($\mathcal{T}$).** Before describing how to construct the tree $\mathcal{T}$, which completes the description of the instance and the reduction, we still have one proviso.

While describing how to obtain the instance of the target problem from the instance of the source problem, it often helps to describe simultaneously how to obtain a yes-certificate for the target instance from a hypothetical yes-certificate for the source instance. Hence, let $\sigma_B$ and $\sigma_C$ be two permutations in $S_n$ meant to encode a generic possible solution to the generic ABCDE-PARTITION problem instance (since all the elements in $D$ and $E$ are equal, it is enough to consider these two permutations). The pair $(\sigma_B, \sigma_C)$ is a truly valid solution, i.e., a yes-certificate, iff $a_i + b_{\sigma_B(i)} + c_{\sigma_C(i)} + d_i + e_i = T$ for every $i = 1, 2, ..., n$. We are now going to describe not only how to construct an instance of the target problem but also a solution $S = S(\sigma_B, \sigma_C)$ for it, which depends solely on the hypothetical yes-certificate $(\sigma_B, \sigma_C)$.

The tree $\mathcal{T}$ and the solution $S = S(\sigma_B, \sigma_C)$ are constructed as follows:

1. Start with a perfect binary tree of height $h + 1$, with $2n = 2^{(h+1)}$ leaves. Its $n$ internal nodes at depth $h$ are called test nodes, denoted $t_i, i \in [n]$ (also called $r_i^0$). This tree is a full binary tree thanks to Property B.3. Moreover, each test node $t_i$ will next become the root of a subtree of depth 5, all these subtrees having the very same topology, described in the following and illustrated in Figure B.1.

2. For $i = 1, ..., n$, the left child of the test node $t_i$ is a leaf of the subtree rooted at $t_i$ (and hence also of the global tree under construction). The certificate assigns one different copy of $T$ to each left child of a test node. These $n$ nodes will be the only leaves at depth $2n = 2^{(h+1)}$ in the final tree under construction. However, the right child of $t_i$, called $r_i^1$, has two children described next.

3. The left child of $r_i^1$ is a leaf, and the certificate assigns to this leaf the number $a_i$. On the other hand, the right child of $r_i^1$, which we denote $r_i^2$, will not be a leaf, which means that it has two children, described next.

4. In the next step, we also let the left child of $r_i^2$ be a leaf. The certificate assigns the number $b_{\sigma_B(i)}$ to the left child. On the other hand, the right child of $r_i^2$, called $r_i^3$, will have two children, described next.

5. As before, the left child of $r_i^3$ is also a leaf, and the certificate assigns the number $c_{\sigma_C(i)}$ to it. The right child of $r_i^3$, called $r_i^4$, will also have two children, but, finally, both of them are leaves: to the left (resp., right) child leaf, the certificate associates the number $d_i$ (resp., $e_i$).

The set $I$ of the internal nodes of $\mathcal{T}$ partitions into $I_<$ and $I_\geqslant$, those of depth less than $h$ and those of depth at least $h$, respectively. In other words, all the strict ancestors of test nodes $t_i$ versus $I_\geqslant = \bigcup_{i=1,...,n}\{t_i, r_i^1, r_i^2, r_i^3, r_i^4\}$. We also define $I_>$ as the set of internal nodes at depth strictly greater than $h$.

Figure B.1.: Subtree rooted at the test node $t_i$ with a canonical weight assignment. Recall that in the full tree $\mathcal{T}$, a full binary tree connects all the test nodes $t_i$.

**Definition B.4.** *A weight assignment is called* canonical *if it is of the form $S(\sigma_B, \sigma_C)$ for some pair $(\sigma_B, \sigma_C)$. Equivalently, the canonical assignments are those where all $2n$ leaf nodes at depth $h+5$ have been assigned one different copy of weight $Mn^2$, and all $n$ leaf nodes at depth $h+1$ (resp., $h+2$, $h+3$, or $h+4$) have weight precisely $T$ (resp., falling in the interval $(8n^2M, 8n^2M+M)$, $(4n^2M, 4n^2M+M)$, or $(2n^2M, 2n^2M+M)$).*

The NP-hardness result now follows from the following discussion.

**Lemma B.1.** *The total imbalance cost of $S(\sigma_B, \sigma_C)$ in the nodes $I_< \cup \{t_i \,|\, i \in [n]\}$ is equal to 0 if and only if $S(\sigma_B, \sigma_C)$ encodes a yes-certificate.*

*Proof.* First of all, the imbalance at the internal node $t_i$ is equal to 0 if and only if

$$T = a_i + b_i + c_i + d_i + e_i$$

or equivalently,

$$T^1 = a_i^1 + b_i^1 + c_i^1$$

for every $i \in [n]$. That is, every 5-tuple (resp., every triplet) needs to sum up to $T$ (resp., $T^1$), the target value. To complete the proof, we just need to observe that nodes at depth $h-1$ have as children two test nodes. Thus, their imbalance is 0 if and only if the two test nodes have exactly the same weight (equivalently, the triplets associated to them sum to the same value). Going up the (full binary) tree, it is easy to see that we need all the test nodes to have exactly the same weight, i.e., all the 5-tuples to sum up to the same value. □ □

**Lemma B.2.** *The total imbalance cost of $S(\sigma_B, \sigma_C)$ is greater or equal to $\sum_{i=1}^n (a_i^1 - c_i^1)$ and equality holds if and only if $S(\sigma_B, \sigma_C)$ encodes a yes-certificate.*

*Proof.* We have already seen in B.1 that $\sum_{v \in I_< \cup \{t_i\}} imb(v) = 0$ if and only if $S(\sigma_B, \sigma_C)$ encodes a yes-certificate. We will now prove that $\sum_{v \in I_>} imb(v) = \sum_{i=1}^n (a_i^1 - c_i^1)$ for canonical assignments, from where the result follows. First, for any canonical assignment, $imb(r_i^4) = n^2M - n^2M = 0$ for every $i = 1, \ldots, n$. We will next see that, for every canonical assignment, $\sum_{v \in \{r_i^1, r_i^2, r_i^3 \,:\, i=1,\ldots,n\}} imb(v) = \sum_{i=1}^n (a_i^1 - c_i^1)$.

First of all,

$$\sum_{i=1}^n imb(r_i^3) = \sum_{i=1}^n |c_i - w(r_i^4)| = \sum_{i=1}^n |(c_i^1 + 2n^2M) - 2n^2M| = \sum_{i=1}^n c_i^1$$

Similarly,

$$\sum_{i=1}^{n} imb(r_i^2) = \sum_{i=1}^{n} |b_i - w(r_i^3)| = \sum_{i=1}^{n} |b_i^1 - c_i^1| = \sum_{i=1}^{n} (b_i^1 - c_i^1)$$

where the last equality follows from the property that $b_i > c_i$. Finally,

$$\sum_{i=1}^{n} imb(r_i^1) = \sum_{i=1}^{n} |a_i - w(r_i^2)| = \sum_{i=1}^{n} |a_i^1 - (b_i^1 + c_i^1)| = \sum_{i=1}^{n} (a_i^1 - b_i^1 - c_i^1)$$

where again we use the property that $a_i^1 > b_i^1 + c_i^1$. Thus, summing all the costs below every test node, we get that the total cost is

$$\sum_{i=1}^{n} ((a_i^1 - b_i^1 - c_i^1) + (b_i^1 - c_i^1) + c_i^1) = \sum_{i=1}^{n} (a_i^1 - c_i^1)$$

□ □

Before continuing, note that $\sum_{i=1}^{n} (a_i^1 - c_i^1) \ll n^2 M$. Indeed,

$$\sum_{i=1}^{n} (a_i^1 - c_i^1) \leqslant \sum_{i=1}^{n} (M^0 + 8n^2 M^0 - 2n^2 M^0) \leqslant nM$$

The last inequality follows since $M = T^0 + 14n^2 M^0$, which is clearly greater than the term in the sum.

**Lemma B.3.** *Any assignment $f$ of cost less than $n^2 M$ is canonical.*

*Proof.* Let $f$ be any assignment of cost less than $n^2 M$. Notice that the constructed tree has precisely $n$ internal nodes whose two children are both leaves (those labeled $r_i^4$). At the same time, we have $2n$ weights of value $n^2 M$, whereas all other weights exceed $2n^2 M$. Therefore, all copies of weight $n^2 M$ should be assigned to the leaves that are children of some $r_i^4$, that is, to the $2n$ nodes of largest depth $h + 5$. Indeed, if at least one of the copies of weight $n^2 M$ were assigned to a node which is not at largest depth, then the imbalance at its parent node would be $|n^2 M - w_r|$, with $w_r$ being the weight of the right child, which is always greater or equal than $2n^2 M$. Thus, the imbalance would be already at least $n^2 M$.

After this, we can go up the tree. The $n$ nodes of weight in the interval $[2n^2 M, 2n^2 M + M]$ are mapped to nodes that are left children of $r_i^3$ nodes (all leaf nodes at depth $h+4$). Otherwise, the weight of that node would be at least $4n^2 M$, yielding an imbalance of at least $2n^2 M$. Similarly, the $n$ nodes of weight in the interval $[4n^2 M, 4n^2 M + M]$ are mapped to nodes that are left children of $r_i^2$ nodes (all leaf nodes at depth $h + 3$), and the $n$ nodes of weight in the interval $[8n^2 M, 8n^2 M + M]$ are mapped to nodes that are left children of $r_i^2$ nodes (all leaf nodes at depth $h + 2$). Finally, the $n$ nodes of weight $T$ are mapped to nodes that are left children of test nodes $t_i$ (all leaf nodes at depth $h + 1$).

This shows that if we want an assignment of cost less than $n^2M$, then every weight, while it can be assigned to the leaves of a subtree rooted at any of the test nodes $t_i$, it has to be assigned to a leaf node of the right depth/category. But then $f$ is a canonical assignment. □ □

**Theorem B.1.** *Balanced Mobiles is strongly* NP*-hard.*

*Proof.* We have described a log-space reduction that, given a generic instance $I$ of $ABCDE$-partition yields an instance $(\mathcal{T}, W)$ of Balanced Mobiles and a lower bound $L := \sum_{i=1}^{n}(a_i^1 - c_i^1)$ such that:

1. Every possible solution to $(\mathcal{T}, W)$ has total imbalance cost at least $L$.
2. $(\mathcal{T}, W)$ admits a solution of cost $L$ iff $I$ is a yes-instance of the $ABCDE$-partition problem.

This already shows that the Balanced Mobiles optimization problem is NP-hard. The Balanced Mobiles problem is strongly NP-hard because the $ABCDE$-partition problem is strongly NP-complete, and when the reduction is applied on top of any shallow class of instances of $ABCDE$-partition, it yields a shallow class of instances of Balanced Mobiles. □ □

Note that this implies that the decision version of the problem is (strongly) NP-complete. Indeed, one can check that the problem is in NP because given a potential solution, it can be verified in polynomial time whether it is valid or not.

## B.4. Conclusion

We have shown that Balanced Mobiles is strongly NP-hard when the full binary tree is given as input. However, note that the complexity when the tree is not given remains open. Indeed, our reduction cannot be directly extended to this case since then, there is no structure to ensure that weights of set $A$ are grouped with weights of sets $B$ and $C$. On the other hand, the complexity when the weights are constant is also unknown, as in our proof, the constructed weights depend on $n$. Finally, with respect to the parameterized complexity, as we mentioned before, it is only known that the problem is in the parameterized class XP, parameterized by the optimal imbalance [89], so other future work includes to study whether there exists a fixed parameter algorithm or not.

# Bibliography


[1] Faisal N. Abu-Khzam, Judith Egan, Serge Gaspers, Alexis Shaw, and Peter Shaw. Cluster editing with vertex splitting. In Jon Lee, Giovanni Rinaldi, and Ali Ridha Mahjoub, editors, *Combinatorial Optimization - 5th International Symposium, ISCO 2018, Marrakesh, Morocco, April 11-13, 2018, Revised Selected Papers*, volume 10856 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2018. doi:10.1007/978-3-319-96151-4_1.

[2] Ranendu Adhikary, Kaustav Bose, Satwik Mukherjee, and Bodhayan Roy. Complexity of maximum cut on interval graphs. *Discret. Comput. Geom.*, 70(2):307–322, 2023. URL: https://doi.org/10.1007/s00454-022-00472-y, doi:10.1007/S00454-022-00472-Y.

[3] Shlomo Ahal and Yuri Rabinovich. On complexity of the subpattern problem. *SIAM Journal on Discrete Mathematics*, 22(2):629–649, 2008.

[4] Michael H. Albert, Marie-Louise Lackner, Martin Lackner, and Vincent Vatter. The Complexity of Pattern Matching for 321-Avoiding and Skew-Merged Permutations. *Discret. Math. Theor. Comput. Sci.*, 18(2), 2016. URL: http://dmtcs.episciences.org/2607.

[5] Per Alexandersson, Samuel Asefa Fufa, Frether Getachew, and Dun Qiu. Pattern-avoidance and Fuss-Catalan numbers. *arXiv preprint arXiv:2201.08168*, 2022.

[6] Thomas Andreae. On the unit interval number of a graph. *Discrete applied mathematics*, 22(1):1–7, 1988.

[7] Virginia Ardévol Martínez, Romeo Rizzi, Abdallah Saffidine, Florian Sikora, and Stéphane Vialette. Generalizing roberts characterization of unit interval graphs, 2024. arXiv:2404.17872.

[8] Virginia Ardévol Martínez, Romeo Rizzi, and Florian Sikora. Hardness of balanced mobiles. In Sun-Yuan Hsieh, Ling-Ju Hung, and Chia-Wei Lee, editors, *Combinatorial Algorithms - 34th International Workshop, IWOCA 2023, Tainan, Taiwan, June 7-10, 2023, Proceedings*, volume 13889 of *Lecture Notes in Computer Science*, pages 25–35. Springer, 2023. doi:10.1007/978-3-031-34347-6_3.

[9] Virginia Ardévol Martínez, Romeo Rizzi, Florian Sikora, and Stéphane Vialette. Recognizing unit multiple intervals is hard. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation, ISAAC 2023*, volume 283 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl -

Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.ISAAC.2023.8, doi:10.4230/LIPICS.ISAAC.2023.8.

[10] Virginia Ardévol Martínez, Florian Sikora, and Stéphane Vialette. Parity permutation pattern matching. In Chun-Cheng Lin, Bertrand M. T. Lin, and Giuseppe Liotta, editors, *WALCOM: Algorithms and Computation - 17th International Conference and Workshops, WALCOM 2023, Hsinchu, Taiwan, March 22-24, 2023, Proceedings*, volume 13973 of *Lecture Notes in Computer Science*, pages 384–395. Springer, 2023. doi:10.1007/978-3-031-27051-2_32.

[11] Emmanuel Arrighi, Matthias Bentert, Pål Grønås Drange, Blair D. Sullivan, and Petra Wolf. Cluster editing with overlapping communities. In Neeldhara Misra and Magnus Wahlström, editors, *18th International Symposium on Parameterized and Exact Computation, IPEC 2023, September 6-8, 2023, Amsterdam, The Netherlands*, volume 285 of *LIPIcs*, pages 2:1–2:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.IPEC.2023.2, doi:10.4230/LIPICS.IPEC.2023.2.

[12] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001. doi:10.1145/502102.502107.

[13] Reuven Bar-Yehuda, Magnús M Halldórsson, Joseph Naor, Hadas Shachnai, and Irina Shapira. Scheduling split intervals. *SIAM J. Comput.*, 36(1):1–15, 2006.

[14] Krzysztof Bartoszek, Tomás M. Coronado, Arnau Mir, and Francesc Rosselló. Squaring within the Colless index yields a better balance index. *Mathematical Biosciences*, 331:108503, 2021. URL: https://www.sciencedirect.com/science/article/pii/S0025556420301498, doi:https://doi.org/10.1016/j.mbs.2020.108503.

[15] Seymour Benzer. On the topology of the genetic fine structure. *Proceedings of the National Academy of Sciences*, 45(11):1607–1620, 1959.

[16] Benjamin Aram Berendsohn, László Kozma, and Dániel Marx. Finding and Counting Permutations via CSPs. *Algorithmica*, 83(8):2552–2577, 2021. doi:10.1007/s00453-021-00812-z.

[17] Stéphane Bessy and Anthony Perez. Polynomial kernels for proper interval completion and related problems. *Inf. Comput.*, 231:89–108, 2013. URL: https://doi.org/10.1016/j.ic.2013.08.006, doi:10.1016/J.IC.2013.08.006.

[18] Ivan Bliznets, Fedor V Fomin, Marcin Pilipczuk, and Michał Pilipczuk. A subexponential parameterized algorithm for proper interval completion. *SIAM Journal on Discrete Mathematics*, 29(4):1961–1987, 2015.

[19] Michael GB Blum, Olivier François, and Svante Janson. The mean, variance and limiting distribution of two statistics sensitive to phylogenetic tree balance. *The Annals of Applied Probability*, 16(4):2195–2214, 2006.

[20] Hans L Bodlaender, Rodney G Downey, Michael R Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.

[21] Kenneth P. Bogart and Douglas B. West. A short proof that 'proper = unit'. *Discret. Math.*, 201(1-3):21–23, 1999. doi:10.1016/S0012-365X(98)00310-0.

[22] Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

[23] Kellogg S. Booth and J. Howard Johnson. Dominating sets in chordal graphs. *SIAM J. Comput.*, 11(1):191–199, 1982. doi:10.1137/0211015.

[24] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-Tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

[25] Prosenjit Bose, Jonathan F. Buss, and Anna Lubiw. Pattern Matching for Permutations. *Inf. Process. Lett.*, 65(5):277–283, 1998. doi:10.1016/S0020-0190(97)00209-3.

[26] Marie-Louise Bruner and Martin Lackner. The computational landscape of permutation patterns. *Pure Mathematics and Applications: Special Issue for the Permutation Patterns 2012 Conference*, 24(2):83–101, 2013.

[27] Marie-Louise Bruner and Martin Lackner. A Fast Algorithm for Permutation Pattern Matching Based on Alternating Runs. *Algorithmica*, 75(1):84–117, 2016. doi:10.1007/s00453-015-0013-y.

[28] Laurent Bulteau, Guillaume Fertin, Vincent Jugé, and Stéphane Vialette. Permutation Pattern Matching for Doubly Partially Ordered Patterns. In *Proc. of CPM*, volume 223 of *LIPIcs*, pages 21:1–21:17, 2022.

[29] Laurent Bulteau, Romeo Rizzi, and Stéphane Vialette. Pattern Matching for k-Track Permutations. In Costas S. Iliopoulos, Hon Wai Leong, and Wing-Kin Sung, editors, *Combinatorial Algorithms - 29th International Workshop, IWOCA 2018, Singapore, July 16-19, 2018, Proceedings*, volume 10979 of *Lecture Notes in Computer Science*, pages 102–114. Springer, 2018. doi:10.1007/978-3-319-94667-2_9.

[30] Peter Buneman et al. A characterisation of rigid circuit graphs. *Discret. Math.*, 9(3):205–212, 1974.

[31] Ayelet Butman, Danny Hermelin, Moshe Lewenstein, and Dror Rawitz. Optimization problems in multiple-interval graphs. *ACM Trans. Algorithms*, 6(2):1–18, 2010.

[32] Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

[33] Yixin Cao. Recognizing (unit) interval graphs by zigzag graph searches. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 92–106. SIAM, 2021.

[34] Márcia R Cerioli, Fabiano de S Oliveira, and Jayme L Szwarcfiter. On counting interval lengths of interval graphs. *Discrete Applied Mathematics*, 159(7):532–543, 2011.

[35] Pierre Chambon. Split genes. *Scientific American*, 244(5):60–71, 1981.

[36] F. Cheah and Derek G. Corneil. on the structure of trapezoid graphs. *Discret. Appl. Math.*, 66(2):109–133, 1996. doi:10.1016/0166-218X(94)00158-A.

[37] Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David Juedes, Iyad A Kanj, and Ge Xia. Tight lower bounds for certain parameterized np-hard problems. *Information and Computation*, 201(2):216–231, 2005.

[38] Rayan Chikhi, Paul Medvedev, Martin Milanic, and Sofya Raskhodnikova. On the readability of overlap digraphs. *Discret. Appl. Math.*, 205:35–44, 2016. URL: https://doi.org/10.1016/j.dam.2015.12.009, doi:10.1016/J.DAM.2015.12.009.

[39] Joel E. Cohen. *Food Webs and Niche Space*, volume 11 of *Monographs in Population Biology*. Princeton University Press, 1978.

[40] Donald H Colless. Phylogenetics: The theory and practice of phylogenetic systematics., 1982.

[41] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.

[42] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

[43] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[44] Derek G Corneil. A simple 3-sweep lbfs algorithm for the recognition of unit interval graphs. *Discrete Applied Mathematics*, 138(3):371–379, 2004.

[45] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The LBFS structure and recognition of interval graphs. *SIAM J. Discret. Math.*, 23(4):1905–1953, 2009. doi:10.1137/S0895480100373455.

[46] Tomás M Coronado, Mareike Fischer, Lina Herbst, Francesc Rosselló, and Kristina Wicke. On the minimum value of the colless index and the bifurcating trees that achieve it. *Journal of Mathematical Biology*, 80(7):1993–2054, 2020.

[47] Christophe Crespelle and Ioan Todinca. An o (n2)-time algorithm for the minimal interval completion problem. *Theoretical Computer Science*, 494:75–85, 2013.

[48] Maxime Crochemore, Danny Hermelin, Gad M. Landau, Dror Rawitz, and Stéphane Vialette. Approximating the 2-interval pattern problem. *Theor. Comput. Sci.*, 395(2-3):283–297, 2008. doi:10.1016/j.tcs.2008.01.007.

[49] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

[50] Peter Damaschke. Forbidden ordered subgraphs. *Topics in Combinatorics and Graph Theory: Essays in Honour of Gerhard Ringel*, pages 219–229, 1990.

[51] Jitender S. Deogun and K. Gopalakrishnan. Consecutive retrieval property-revisited. *Inf. Process. Lett.*, 69(1):15–20, 1999. doi:10.1016/S0020-0190(98)00186-0.

[52] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[53] Mitre C Dourado, Fábio Protti, Dieter Rautenbach, Jayme L Szwarcfiter, et al. Mixed unit interval graphs. *Discrete Mathematics*, 312(22):3357–3363, 2012.

[54] François Dross, Claire Hilaire, Ivo Koch, Valeria Leoni, Nina Pardal, Maria Inés Lopez Pujato, and Vinícius Fernandes dos Santos. On the proper interval completion problem within some chordal subclasses. *CoRR*, abs/2110.07706, 2021. URL: https://arxiv.org/abs/2110.07706, arXiv:2110.07706.

[55] Guillermo Durán, Florencia Fernández Slezak, Luciano N. Grippo, Fabiano de S. Oliveira, and Jayme Luiz Szwarcfiter. On unit d–interval graphs. VII Latin American Workshop on Cliques in Graphs, 2016. URL: https://www.mate.unlp.edu.ar/~liliana/lawclique_2016/ffslezak.pdf,https://www.mate.unlp.edu.ar/~liliana/lawclique_2016/prolist.pdf.

[56] Peter Eades and Candido Ferreira Xavier de Mendonça Neto. Vertex splitting and tension-free layout. In Franz-Josef Brandenburg, editor, *Graph Drawing, Symposium on Graph Drawing, GD '95, Passau, Germany, September 20-22, 1995, Proceedings*, volume 1027 of *Lecture Notes in Computer Science*, pages 202–211. Springer, 1995. URL: https://doi.org/10.1007/BFb0021804, doi:10.1007/BFB0021804.

[57] David Eppstein, Philipp Kindermann, Stephen G. Kobourov, Giuseppe Liotta, Anna Lubiw, Aude Maignan, Debajyoti Mondal, Hamideh Vosoughpour, Sue Whitesides, and Stephen K. Wismath. On the planar split thickness of graphs. *Algorithmica*, 80(3):977–994, 2018. URL: `https://doi.org/10.1007/s00453-017-0328-y`, `doi:10.1007/S00453-017-0328-Y`.

[58] Paul Erdös and Douglas B West. A note on the interval number of a graph. *Discret. Math.*, 55(2):129–133, 1985.

[59] Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. `doi:10.1016/j.tcs.2008.09.065`.

[60] Michael R. Fellows, Jan Kratochvíl, Matthias Middendorf, and Frank Pfeiffer. The complexity of induced minors and related problems. *Algorithmica*, 13(3):266–282, 1995. `doi:10.1007/BF01190507`.

[61] Mareike Fischer, Lina Herbst, Sophie Kersting, Luise Kühn, and Kristina Wicke. Tree balance indices: a comprehensive survey, 2021. URL: `https://arxiv.org/abs/2109.12281`, `doi:10.48550/ARXIV.2109.12281`.

[62] Peter C. Fishburn. *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets*. Wiley, 1985.

[63] Jacob Fox. Stanley-wilf limits are typically exponential. *arXiv preprint arXiv:1310.8378*, 2013.

[64] Mathew C. Francis, Daniel Gonçalves, and Pascal Ochem. The maximum clique problem in multiple interval graphs. *Algorithmica*, 71(4):812–836, 2015. `doi:10.1007/s00453-013-9828-6`.

[65] Peter Frankl and Hiroshi Maehara. Open-interval graphs versus closed-interval graphs. *Discret. Math.*, 63(1):97–100, 1987.

[66] Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific journal of mathematics*, 15(3):835–855, 1965.

[67] Philippe Gambette and Stéphane Vialette. On restrictions of balanced 2-interval graphs. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *Graph-Theoretic Concepts in Computer Science, 33rd International Workshop, WG 2007, Dornburg, Germany, June 21-23, 2007. Revised Papers*, volume 4769 of *LNCS*, pages 55–65. Springer, 2007. `doi:10.1007/978-3-540-74839-7\_6`.

[68] Philippe Gambette and Stéphane Vialette. On restrictions of balanced 2-interval graphs. In *WG 2007*, volume 4769 of *LNCS*, pages 55–65. Springer, 2007. `doi:10.1007/978-3-540-74839-7\_6`.

[69] Frédéric Gardi. The Roberts characterization of proper and unit interval graphs. *Discret. Math.*, 307(22):2906–2908, 2007. `doi:10.1016/j.disc.2006.04.043`.

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

[71] Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

[72] Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.

[73] Paweł Gawrychowski and Mateusz Rzepecki. Faster Exponential Algorithm for Permutation Pattern Matching. In Karl Bringmann and Timothy Chan, editors, *5th Symposium on Simplicity in Algorithms, SOSA@SODA 2022, Virtual Conference, January 10-11, 2022*, pages 279–284. SIAM, 2022. doi:10.1137/1.9781611977066.21.

[74] Juan B. Gil and Jessica A. Tomasko. Restricted Grassmannian permutations. *Enumerative Combinatorics and Applications*, 2(4):Article #S4PP6, 2021.

[75] Paul C Gilmore and Alan J Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16:539–548, 1964.

[76] Paul W Goldberg, Martin C Golumbic, Haim Kaplan, and Ron Shamir. Four strikes against physical mapping of dna. *Journal of Computational Biology*, 2(1):139–152, 1995.

[77] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

[78] Martin Charles Golumbic. Interval graphs. In *Annals of Discrete Mathematics*, volume 57, pages 171–202. Elsevier, 2004.

[79] Jerrold R Griggs. Extremal values of the interval number of a graph, ii. *Discrete Mathematics*, 28(1):37–47, 1979.

[80] Jerrold R. Griggs and Douglas B. West. Extremal values of the interval number of a graph. *SIAM J. Algebraic Discret. Methods*, 1(1):1–7, 1980. doi:10.1137/0601001.

[81] Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 82–101. SIAM, 2014. doi:10.1137/1.9781611973402.7.

[82] Sylvain Guillemot and Stéphane Vialette. Pattern Matching for 321-Avoiding Permutations. In Yingfei Dong, Ding-Zhu Du, and Oscar H. Ibarra, editors, *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Hon-*

*olulu, Hawaii, USA, December 16-18, 2009. Proceedings*, volume 5878 of *Lecture Notes in Computer Science*, pages 1064–1073. Springer, 2009. doi:10.1007/978-3-642-10631-6_107.

[83] András Gyárfás. On the chromatic number of multiple interval graphs and overlap graphs. *Discrete mathematics*, 55(2):161–166, 1985.

[84] András Gyárfás and Jenö Lehel. A helly-type problem in trees. *Combinatorial Theory and its applications*, 4:571–584, 1970.

[85] András Gyárfás and Douglas West. Multitrack interval graphs. *Congressus Numerantium*, pages 109–116, 1995.

[86] Michel Habib, Ross M. McConnell, Christophe Paul, and Laurent Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theor. Comput. Sci.*, 234(1-2):59–84, 2000. doi:10.1016/S0304-3975(97)00241-7.

[87] G. Hajos. Uber eine art von graphen. *Internationale Math. Nachrichten*, 11:65, 1957.

[88] Yassine Hamoudi, Sophie Laplante, and Roberto Mantaci. Balanced mobiles, 2016. www.iwoca.org, Problems Section.

[89] Yassine Hamoudi, Sophie Laplante, and Roberto Mantaci. Balanced mobiles with applications to phylogenetic trees and Huffman-like problems. Technical report, 2017. https://hal.science/hal-04047256.

[90] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[91] Russell Impagliazzo and Ramamohan Paturi. Complexity of k-sat. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity, Atlanta, Georgia, USA, May 4-6, 1999*, pages 237–240. IEEE Computer Society, 1999. doi:10.1109/CCC.1999.766282.

[92] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. URL: https://doi.org/10.1006/jcss.2001.1774, doi:10.1006/JCSS.2001.1774.

[93] Vít Jelínek and Jan Kynčl. Hardness of Permutation Pattern Matching. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 378–396. SIAM, 2017. doi:10.1137/1.9781611974782.24.

[94] Vít Jelínek, Michal Opler, and Jakub Pekárek. Griddings of Permutations and Hardness of Pattern Matching. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 65:1–65:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.65.

[95] Minghui Jiang. On the parameterized complexity of some optimization problems related to multiple-interval graphs. *Theor. Comput. Sci.*, 411(49):4253–4262, 2010. doi:10.1016/j.tcs.2010.09.001.

[96] Minghui Jiang. Recognizing d-interval graphs and d-track interval graphs. *Algorithmica*, 66(3):541–563, 2013. doi:10.1007/s00453-012-9651-5.

[97] Haim Kaplan, Ron Shamir, and Robert E Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM Journal on Computing*, 28(5):1906–1922, 1999.

[98] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

[99] Toshinobu Kashiwabara and Toshio Fujisawa. An np-complete problem on interval graphs. In *IEEE Symp. of Circuits and Systems*, pages 82–83, 1979.

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

[101] Sergey Kitaev. *Patterns in Permutations and Words*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2011.

[102] Sergey Kitaev and Jeffrey Remmel. Classifying descents according to parity. *Annals of Combinatorics*, 11:173–193, 2007.

[103] Pavel Klavík, Yota Otachi, and Jiří Šejnoha. On the classes of interval graphs of limited nesting and count of lengths. *Algorithmica*, 81(4):1490–1511, 2019.

[104] Norbert Korte and Rolf H Möhring. *Transitive orientation of graphs with side constraints*. Hochsch. Hildesheim, 1985.

[105] Norbert Korte and Rolf H Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM Journal on Computing*, 18(1):68–81, 1989.

[106] Stefan Kratsch and Magnus Wahlström. Two edge modification problems without polynomial kernels. In *International Workshop on Parameterized and Exact Computation*, pages 264–275. Springer, 2009.

[107] Nishit Kumar and Narsingh Deo. Multidimensional interval graphs. *Congressus Numerantium*, pages 45–56, 1994.

[108] Rochelle Leibowitz, Susan F Assmann, and GW Peck. The interval count of a graph. *SIAM Journal on Algebraic Discrete Methods*, 3(4):485–494, 1982.

[109] C Lekkerkerker and Johan Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962.

[110] Peng Li and Yaokun Wu. Maximal neighborhood search and rigid interval graphs. *Journal of Graph Algorithms and Applications*, 17(3):245–264, 2013.

[111] Peng Li and Yaokun Wu. A four-sweep LBFS recognition algorithm for interval graphs. *Discret. Math. Theor. Comput. Sci.*, 16(3):23–50, 2014. URL: `https://doi.org/10.46298/dmtcs.2094`, `doi:10.46298/DMTCS.2094`.

[112] Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

[113] Peter J Looges and Stephan Olariu. Optimal greedy algorithms for indifference graphs. *Computers & Mathematics with Applications*, 25(7):15–25, 1993.

[114] R Duncan Luce. Semiorders and a theory of utility discrimination. *Econometrica, Journal of the Econometric Society*, pages 178–191, 1956.

[115] Christoph Maas. Determining the interval number of a triangle-free graph. *Computing (Wien. Print)*, 31(4):347–354, 1983.

[116] Matthias Mayer and Fikret Erçal. Genetic algorithms for vertex splitting in dags. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, Urbana-Champaign, IL, USA, June 1993*, page 646. Morgan Kaufmann, 1993.

[117] Robert McGuigan. Presentation at NSF-CBMS Conference at Colby College, 1977.

[118] Terry A McKee and Fred R McMorris. *Topics in intersection graph theory*. SIAM, 1999.

[119] George B. Mertzios and Derek G. Corneil. Vertex splitting and the recognition of trapezoid graphs. *Discret. Appl. Math.*, 159(11):1131–1147, 2011. URL: `https://doi.org/10.1016/j.dam.2011.03.023`, `doi:10.1016/J.DAM.2011.03.023`.

[120] George B. Mertzios, Ignasi Sau, and Shmuel Zaks. The recognition of tolerance and bounded tolerance graphs. In Jean-Yves Marion and Thomas Schwentick, editors, *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, volume 5 of *LIPIcs*, pages 585–596. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. URL: `https://doi.org/10.4230/LIPIcs.STACS.2010.2487`, `doi:10.4230/LIPICS.STACS.2010.2487`.

[121] Arnau Mir, Lucía Rotger, and Francesc Rosselló. Sound Colless-like balance indices for multifurcating trees. *PloS one*, 13(9):e0203401, 2018.

[122] Both Emerite Neou, Romeo Rizzi, and Stéphane Vialette. Permutation pattern matching in (213, 231)-avoiding permutations. *Discret. Math. Theor. Comput. Sci.*, 18(2), 2016. URL: `http://dmtcs.episciences.org/3199`.

[123] Rolf Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31. OUP Oxford, 2006.

[124] Martin Nöllenburg, Manuel Sorge, Soeren Terziadis, Anaïs Villedieu, Hsiang-Yun Wu, and Jules Wulms. Planarizing graphs and their drawings by vertex splitting. In Patrizio Angelini and Reinhard von Hanxleden, editors, *Graph Drawing and Network Visualization - 30th International Symposium, GD 2022, Tokyo, Japan, September 13-16, 2022, Revised Selected Papers*, volume 13764 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2022. `doi:10.1007/978-3-031-22203-0\_17`.

[125] Stephan Olariu. An optimal greedy heuristic to color interval graphs. *Information Processing Letters*, 37(1):21–25, 1991.

[126] Bhawani S Panda and Sajal K Das. A parallel algorithm for generating bicompatible elimination orderings of proper interval graphs. *Information Processing Letters*, 109(18):1041–1046, 2009.

[127] Andrzej Proskurowski and Jan Arne Telle. Classes of graphs with restricted interval models. *Discrete Mathematics & Theoretical Computer Science*, 3, 1999.

[128] Ganessan Ramalingam and C Pandu Rangan. A unified approach to domination problems on interval graphs. *Information Processing Letters*, 27(5):271–274, 1988.

[129] Ivan Rapaport, Karol Suchan, and Ioan Todinca. Minimal proper interval completions. *Information Processing Letters*, 106(5):195–202, 2008.

[130] Dieter Rautenbach and Jayme L Szwarcfiter. Unit interval graphs of open and closed intervals. *J. Graph Theory*, 72(4):418–429, 2013.

[131] Fred S. Roberts. Indifference graphs. In F. Harary, editor, *Proof Techniques in Graph Theory*, pages 139–146. Academic Press, NY, 1969.

[132] Fred S Roberts. On the boxicity and cubicity of a graph. *Recent progress in combinatorics*, 1(1):301–310, 1969.

[133] Fred S. Roberts. *Graph theory and its applications to problems of society*. SIAM, 1978.

[134] Donald J Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph theory and computing*, pages 183–217. Elsevier, 1972.

[135] Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. doi:10.1137/0205021.

[136] Nicholas D Roussopoulos. A max {m, n} algorithm for determining the graph h from its line graph g. *Information Processing Letters*, 2(4):108–112, 1973.

[137] Michael J Sackin. "Good" and "bad" phenograms. *Systematic Biology*, 21(2):225–226, 1972.

[138] Edward R Scheinerman and Douglas B West. The interval number of a planar graph: Three intervals suffice. *J. Comb. Theory, Ser. B*, 35(3):224–239, 1983.

[139] Dana S. Scott and Patrick Suppes. Foundational aspects of theories of measurement. *J. Symb. Log.*, 23(2):113–128, 1958. doi:10.2307/2964389.

[140] Kwang-Tsao Shao and Robert R. Sokal. Tree Balance. *Systematic Biology*, 39(3):266–276, 09 1990. arXiv:https://academic.oup.com/sysbio/article-pdf/39/3/266/39805211/sysbio_39_3_266.pdf, doi:10.2307/2992186.

[141] Alexandre Simon. Algorithmic study of 2-interval graphs. Master's thesis, Delft University of Technology, 2021.

[142] Dale J Skrien. A relationship between triangulated graphs, comparability graphs, proper interval graphs, proper circular-arc graphs, and nested interval graphs. *Journal of graph Theory*, 6(3):309–316, 1982.

[143] Edward Szpilrajn-Marczewski. Sur deux propriétés des classes d'ensembles. *Fundamenta Mathematicae*, 33(1):303–307, 1945.

[144] Shinji Tanimoto. Combinatorics of the group of parity alternating permutations. *Adv. Appl. Math.*, 44(3):225–230, 2010.

[145] William T. Trotter and Frank Harary. On double and multiple interval graphs. *J. Graph Theory*, 3(3):205–211, 1979. doi:10.1002/jgt.3190030302.

[146] Yngve Villanger, Pinar Heggernes, Christophe Paul, and Jan Arne Telle. Interval completion is fixed parameter tractable. *SIAM Journal on Computing*, 38(5):2007–2020, 2009.

[147] James Richard Walter. *Representations of rigid cycle graphs*. Wayne State University, 1972.

[148] Douglas B. West and David B. Shmoys. Recognizing graphs with fixed interval number is NP-complete. *Discret. Appl. Math.*, 8(3):295–305, 1984. doi:10.1016/0166-218X(84)90127-6.

[149] Kazuaki Yamazaki, Toshiki Saitoh, Masashi Kiyomi, and Ryuhei Uehara. Enumeration of nonisomorphic interval graphs and nonisomorphic permutation graphs. *Theor. Comput. Sci.*, 806:310–322, 2020. URL: `https://doi.org/10.1016/j.tcs.2019.04.017`, `doi:10.1016/J.TCS.2019.04.017`.

[150] Mihalis Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

[151] Peisen Zhang, Eric A. Schon, Stuart G. Fischer, Eftihia Cayanis, Janie Weiss, Susan Kistler, and Philip E. Bourne. An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA. *Comput. Appl. Biosci.*, 10(3):309–317, 1994. URL: `https://doi.org/10.1093/bioinformatics/10.3.309`, `doi:10.1093/BIOINFORMATICS/10.3.309`.

# Résumé étendu en français

Dans ce qui suit, certains termes techniques resteront en anglais. Les graphes d'intersection ont été largement étudiés dans la littérature, car ils sont importants à la fois d'un point de vue théorique et pratique [118]. Étant donnée une famille d'ensembles $\mathcal{F} = \{S_1, \ldots, S_k\}$, nous définissons le *graphe d'intersection* de $\mathcal{F}$, noté $\Omega(\mathcal{F})$, comme le graphe ayant $\mathcal{F}$ comme ensemble de sommets, où le sommet $S_i$ est adjacent au sommet $S_j$ si et seulement si $i \neq j$ et que l'intersection des ensembles est non vide, c'est-à-dire $S_i \cap S_j \neq \emptyset$. Un graphe est un graphe d'intersection s'il existe une famille $\mathcal{F}$ telle que $G = \Omega(\mathcal{F})$. La famille $\mathcal{F}$ est alors appelée une *représentation en ensembles* ou *modèle* de $G$.

Il existe de nombreux types différents de graphes d'intersection. Dans cette thèse, nous étudions des sous-classes particulières des graphes d'intersection, notamment, les graphes d'intervalles et leurs généralisations.

## Graphes d'intervalles

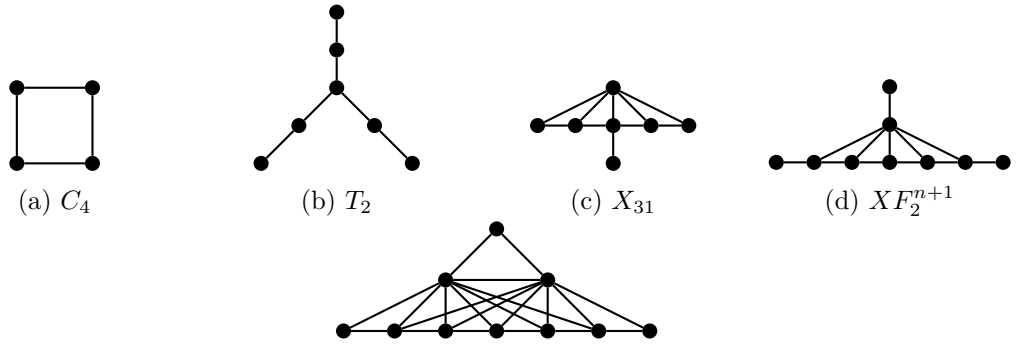
**Définition.** *Un graphe $G = (V, E)$ est un* graphe d'intervalles *s'il existe une bijection des sommets de $G$ vers un multiensemble d'intervalles, $f : V \to \mathcal{I}$, telle qu'il existe une arête entre deux sommets $u$ et $v$ si et seulement si leurs intervalles correspondants se croisent, c'est-à-dire, si $f(u) \cap f(v) \neq \emptyset$. Le multiensemble $\mathcal{I}$ est appelé une* représentation d'intervalles *de $G$ (ou* modèle d'intervalles *dans certaines références).*

Cette classe de graphes a été largement étudiée, principalement en raison de ses nombreuses applications en planification, allocation de ressources et bioinformatique [12, 151, 39], mais aussi en raison de ses avantages algorithmiques : de nombreux problèmes NP-difficiles deviennent résolubles en temps linéaire lorsqu'ils sont restreints à cette classe de graphes, par exemple, *dominating set* [23] ou *Hamiltonian cycle* [100]. L'un des rares problèmes classiques qui reste NP-complet dans cette classe de graphes est *maximum cut* [2].

La première caractérisation des graphes d'intervalles a été donnée par Lekkerkerker et Boland [109].

**Théorème** ([109])**.** *Un graphe non orienté $G$ est un graphe d'intervalles si et seulement si les deux conditions suivantes sont satisfaites :*

1. *$G$ ne contient pas de cycles induits de longueur supérieure à trois.*
2. *$G$ ne contient aucun triplet astéroïdal (un ensemble de trois sommets d'un graphe forme un* triplet astéroïdal *si, pour chaque paire de sommets, il existe un chemin les contenant qui ne passe pas par un voisin du troisième sommet).*



(a) $C_4$ (b) $T_2$ (c) $X_{31}$ (d) $XF_2^{n+1}$

(e) $XF_3^n$

Figure 1.: Sous-graphes induits interdits pour les graphes d'intervalles

**Théorème** ([109])**.** *Un graphe $G$ est un graphe d'intervalles si et seulement s'il ne contient aucun des graphes $C_{n+4}, T_2, X_{31}, XF_2^{n+1}$ ou $XF_3^n$ comme sous-graphe induit (voir Figure 1 pour une illustration des sous-graphes induits interdits).*

**Arbres PQ** Étant donné un ensemble fini $X$ et une collection $\mathcal{I}$ de sous-ensembles de $X$, les arbres PQ servent à représenter toutes les permutations de $X$ dans lesquelles les membres de chaque sous-ensemble $I \in \mathcal{I}$ apparaissent comme une sous-suite consécutive de la permutation. Formellement, un arbre PQ $T$ est un arbre enraciné dont les feuilles sont étiquetées de manière bijective par les éléments de l'ensemble $X$ et dont les nœuds internes sont de deux types, chacun imposant une contrainte différente sur l'ordre de ses enfants :

- Nœuds P : ses enfants peuvent apparaître dans un ordre arbitraire.
- Nœuds Q : ses enfants doivent apparaître dans l'ordre donné (jusqu'à inversion).

Les nœuds P sont représentés par un cercle et les nœuds Q par un carré. Nous pouvons représenter un graphe d'intervalles comme un arbre PQ où l'ensemble $X$ est l'ensemble des cliques maximales (les lignes de la matrice de cliques), et la famille $\mathcal{I}$ est composée des ensembles de toutes les cliques partageant un même sommet $v$ (les sous-ensembles de $X$ constitués des lignes contenant un 1 dans la même colonne), pour chaque sommet $v \in V$.

**Arbres PQ modifiés** Le modèle d'arbre PQ modifié (arbre MPQ) est une simplification du modèle standard d'arbre PQ. Il assigne un ensemble (éventuellement vide) de sommets à chaque nœud de l'arbre PQ : les nœuds P reçoivent un ensemble tandis que les nœuds Q reçoivent un ensemble pour chacun de ses enfants (et chaque ensemble est ordonné de gauche à droite selon l'ordre des enfants, lequel est unique jusqu'à renversement).

Plus concrètement, à un nœud P appelé $P$, nous assignons l'ensemble des sommets de $G$ contenus dans toutes les cliques maximales représentées par le sous-arbre enraciné en $P$ mais qui n'apparaissent dans aucune autre clique. D'autre part, à un nœud Q

appelé $Q$ avec des enfants $Q_1, \ldots, Q_k$, (ordonnés de gauche à droite), nous assignons un ensemble $S_i$, appelé section, pour chaque $Q_i$. La section $S_i$ contient tous les sommets qui sont contenus dans toutes les cliques maximales du sous-arbre enraciné en $Q_i$ et dans au moins un autre sous-arbre enraciné en un certain $Q_j$ ($j \neq i$), mais qui n'apparaissent dans aucune clique appartenant à un autre sous-arbre qui n'est pas sous $Q$.

**Sous-classes des graphes d'intervalles** Deux des sous-classes les plus importantes des graphes d'intervalles sont les *graphes d'intervalles propres* et les *graphes d'intervalles unitaires*.

**Définition.** *Un graphe d'intervalles est* propre *s'il possède une représentation d'intervalles où aucun intervalle n'est proprement contenu dans un autre, et il est* unitaire *s'il possède une représentation où tous les intervalles ont une longueur unitaire.*

Roberts a donné une caractérisation célèbre des graphes d'intervalles unitaires :

**Théorème** ([131])**.** *Soit $G = (V, E)$ un graphe non orienté. Alors, les propriétés suivantes sont équivalentes :*

1. *$G$ est un graphe d'intervalles sans $K_{1,3}$ induit (étoile avec trois feuilles).*
2. *$G$ est un graphe d'intervalles propre.*
3. *$G$ est un graphe d'intervalles unitaire.*

## Graphes d'intervalles multiples

Puisque tous les graphes ne sont pas des graphes d'intervalles, il est naturel de se demander si nous pouvons étendre l'idée de représenter des graphes par des intersections d'intervalles à tout graphe. Cela a motivé l'introduction d'un nouveau paramètre de graphe : l'*interval number*, qui est en relation directe avec la classe des graphes d'intervalles multiples.

**Définition.** *Pour tout entier naturel $d > 0$, un* $d$-intervalle (disjoint) *est l'union de $d$ intervalles (disjoints) sur la ligne réelle.*

**Définition.** *Pour tout entier naturel $d > 0$, un graphe $G$ est un* graphe de $d$-intervalles (disjoints) *s'il existe une bijection entre les sommets de $G$ et un multiensemble de (disjoints) $d$-intervalles, $f : V \to \mathcal{I}$, telle qu'il existe une arête entre deux sommets si et seulement si leurs $d$-intervals correspondants s'intersectent. Le multiensemble $\mathcal{I}$ de $d$-intervalles est appelé une représentation de $d$-intervalles de $G$, et la famille de tous les intervalles qui composent les $d$-intervalles dans $\mathcal{I}$ est appelée la* famille sous-jacente d'intervalles *de $\mathcal{I}$.*

Une fois ces concepts définis, nous pouvons simplement définir l'*interval number* $i(G)$ d'un graphe $G$ comme le plus petit entier $d$ tel que $G$ soit un graphe de $d$-intervalles. De plus, étant donné un graphe d'intervalles (multiples), nous pouvons également définir sa profondeur, qui est une propriété structurelle de ses représentations :

**Définition.** *La* profondeur *d'une famille d'intervalles est le nombre maximum d'intervalles qui partagent un point commun, et pour tout entier naturel $d \geqslant 1$, la* profondeur de représentation *d'un graphe de d-intervalles est la profondeur minimale de toute représentation de d-intervalles du graphe. Nous utilisons le terme profondeur-r comme abréviation de profondeur de représentation au plus r.*

West et Shmoys ont prouvé que déterminer si un graphe a un *interval number* au plus 2 (ou de manière équivalente, reconnaître les graphes de 2-intervalles) est un problème NP-difficile, et ont obtenu le théorème suivant :

**Théorème** ([148])**.** *Pour tout $d \geqslant 2$ et tout $r \geqslant 3$, déterminer si un graphe $G$ admet une représentation par d-intervalles de profondeur au plus r est un problème* NP*-complet.*

Comme pour les graphes d'intervalles classiques, l'étude des graphes de $d$-intervalles a ensuite été restreinte à certaines de ses sous-classes, telles que les graphes à intervalles unitaires, propres, équilibrés ou $(x_1, \ldots, x_d)$ $d$-intervalles.

**Définition.** *Un graphe de d-intervalles (disjoints) est* propre *s'il existe une représentation où aucun intervalle de la famille sous-jacente n'est proprement contenu dans un autre, et il est* unitaire *s'il existe une représentation de d-intervalles (disjoints) où tous les intervalles de la famille sous-jacente ont une longueur unitaire.*

**Définition.** *Un graphe de d-intervalles (disjoints) est* équilibré *s'il existe une représentation de d-intervalles (disjoints) où les d intervalles d'un même d-intervalle ont la même longueur, mais les intervalles de différents d-intervalles peuvent différer en longueur.*

**Définition.** *Un d-intervalle disjoint est un d-intervalle $(x_1, \ldots, x_d)$ si les d intervalles disjoints sont ouverts, ont des extrémités entières, et ont respectivement des longueurs $x_1, \ldots, x_d$.*

Gambette et Vialette ont remarqué que la preuve de la NP-complétude donnée par West et Shmoys pour la reconnaissance des graphes de 2-intervalles pouvait être adaptée pour la reconnaissance des graphes de 2-intervalles équilibrés. Ils ont également montré que la classe des graphes de 2-intervalles équilibrés est proprement contenue dans la classe des graphes de 2-intervalles, et ont initié l'étude de la complexité de la reconnaissance des graphes à 2-intervalles unitaires [67]. Plus tard, Jiang a réussi à prouver que les graphes de $d$-intervalles unitaires de profondeur deux peuvent être reconnus en temps linéaire, et a donné une approximation pour la reconnaissance des graphes de $d$-intervalles non restreints de profondeur deux avec une erreur additive d'un [96]. Cependant, savoir si les graphes de $d$-intervalles unitaires arbitraires et les graphes de $d$-intervalles $(x, \ldots, x)$ peuvent être reconnus en temps polynomial est restée une question ouverte importante. Un résumé des complexités de reconnaissance des différentes sous-classes de graphes de $d$-intervalles est présenté dans Table 1, ainsi qu'un résumé des relations d'inclusion connues entre certaines de ces sous-classes dans Figure 2.

| | graphe de $d$-intervalles |
|---|---|
| sans restriction | NP-complet |
| équilibré | NP-complet ($d = 2$) |
| unitaire | ? |
| (2,...,2) | ? |
| profondeur 2 | ? (+1 approximation) |
| profondeur 2, unitaire | temps linéaire |

Table 1.: Complexité de la reconnaissance de différentes sous-classes de graphes d-intervalles avant les résultats présentés dans ce manuscrit [67, 96].

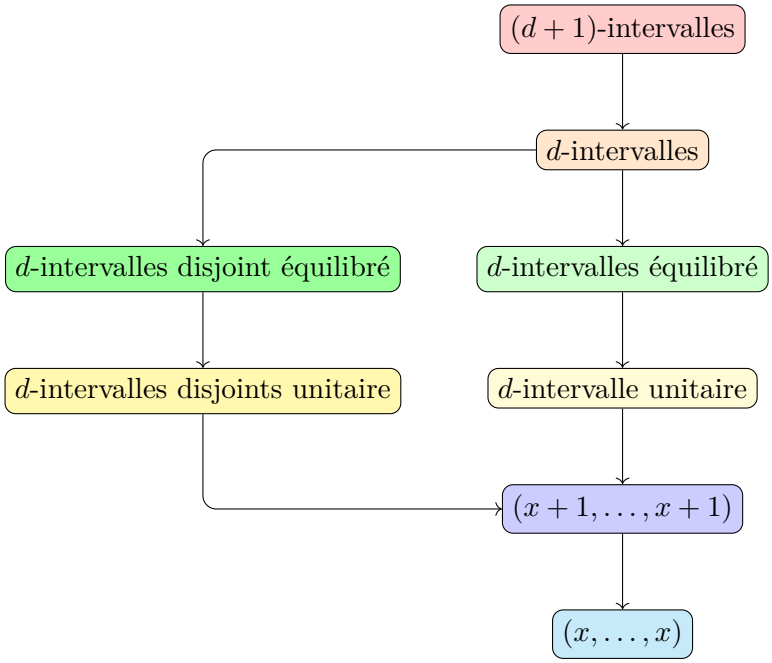



Figure 2.: Relations de contenance entre certaines sous-classes de graphes d'intervalles $d$ pour tout $d \geqslant 2$ (avant les résultats présentés dans cette thèse). Une flèche de la classe $\mathcal{C}_1$ à la classe $\mathcal{C}_2$ indique que $\mathcal{C}_2 \subsetneq \mathcal{C}_1$.

**Discussion sur la définition des $d$-intervalles** Dans la littérature, les graphes de $d$-intervalles ont été définis à la fois comme l'union de $d$ intervalles disjoints [13, 31, 148], comme l'union de $d$ intervalles *pas nécessairement disjoints* [145], et simplement comme l'union de $d$ intervalles, sans spécifier s'ils sont disjoints ou non [58, 138].

Lorsqu'il n'y a pas de restrictions sur la longueur des intervalles, cette ambiguïté n'est pas pertinente, car les deux définitions conduisent à la même classe de graphes (puisqu'il est possible d'étirer les intervalles associés à un même sommet qui s'intersectent pour les rendre disjoints sans changer aucune des autres intersections).

**Observation.** *Les classes des graphes de $d$-intervalles disjoints et des graphes de $d$-intervalles sont équivalentes.*

Cependant, si des restrictions de longueur sont imposées, l'observation précédente ne tient pas. Pour les intervalles unitaires, on ne peut pas remplacer deux intervalles qui s'intersectent $[a, b]$ et $[c, d]$, avec $a < c < b < d$, par $[a, d]$, car l'intervalle résultant ne serait pas de longueur unitaire, et l'étirer pour le rendre unitaire pourrait perturber les autres intersections. Ainsi, dans ce cas, il ne peut être déduit que les deux définitions des intervalles multiples conduisent à la même classe de graphes. En fait, dans cette thèse, nous prouverons qu'elles ne le font pas. Par conséquent, nous distinguerons entre les graphes de $d$-intervalles et les graphes de $d$-intervalles disjoints.

## Un certificat pour être un graphe de 2-intervalles unitaire

Dans le Chapitre 4, nous présentons un certificat combinatoire pour déterminer si un graphe est un graphe de 2-intervalles (disjoints) unitaire. Bien que la manière la plus directe de prouver qu'un graphe donné $G$ appartient à cette classe de graphes soit de fournir une représentation de de 2-intervalles (disjoints) unitaire du graphe, ce n'est pas toujours la plus efficace. Nous prouvons qu'un graphe $G$ est un graphe de 2-intervalles (disjoints) unitaire si et seulement si nous pouvons obtenir un graphe d'intervalles unitaire à partir de $G$ en divisant chaque sommet $v$ en deux représentants de sorte que le voisinage de $v$ soit contenu dans l'union des voisinages des représentants. Nous définissons formellement les divisions (disjoint) d'un graphe et l'ensemble des divisions qui mènent à des graphes d'intervalles unitaires comme suit.

**Définition.** *Étant donné un graphe $G$, une paire $(S, f)$ formée d'un graphe $S$ et d'une fonction $f : V(S) \mapsto V(G)$ est une* division *de $G$ si $f$ satisfait les conditions suivantes :*

- $1 \leqslant |f^{-1}(v)| \leqslant 2$ *pour tout $v \in V(G)$.*
- *Pour chaque arête $(s, t)$ de $S$, $(f(s), f(t))$ est une arête de $G$.*
- *Pour chaque arête $(u, v)$ de $G$, il existe deux sommets $s$ et $t$ dans $f^{-1}(\{u, v\})$ tels que $(s, t)$ soit une arête de $S$.*

**Définition.** *Étant donné un graphe $G$, une paire $(S, f)$ formée d'un graphe $S$ et d'une fonction $f : V(S) \mapsto V(G)$ est une* division disjointe *de $G$ si $f$ satisfait les conditions de la définition précédente ainsi que la condition supplémentaire suivante :*

- *Pour chaque sommet $v$ de $G$, $f^{-1}(v)$ est un ensemble indépendant dans $S$.*

Étant donnés une division $(S, f)$ de $G$ et un sommet $v \in V(G)$, nous appelons l'ensemble $f^{-1}(v)$ l'ensemble des *représentants* du sommet $v$. Si $|f^{-1}(v)| = 1$, nous disons que le sommet $v$ n'a pas été divisé. De même, étant donnée une arête $(u, v) \in E(G)$, l'arête $(s, t)$ de $S$ avec $s$ et $t$ dans $f^{-1}(\{u, v\})$ est appelée un *représentant* de $(u, v)$.

Parmi tous les divisions d'un graphe $G$, nous nous intéressons à ceux qui produisent un graphe d'intervalles unitaire :

**Définition.** *La famille des divisions de $G$ qui conduisent à un graphe d'intervalles unitaire est $\mathcal{S}^*_{\mathcal{U}}(G) := \{(S, f) \mid (S, f)$ est une division de $G$ et $S$ est un graphe d'intervalles unitaire$\}$.*

**Définition.** *La famille des divisions disjointes de $G$ qui conduisent à un graphe d'intervalles unitaire est $\mathcal{S}_{\mathcal{U}}(G) := \{(S, f) \mid (S, f)$ est une division disjoint de $G$ et $S$ est un graphe d'intervalles unitaires$\}$.*

Le lemme suivant montre comment une division (disjointe) $(S, f)$ d'un graphe $G$ peut être utilisé pour certifier que $G$ est un graphe d'intervalles 2-unitaires (disjoint).

**Lemme.** *Nous pouvons caractériser les graphes de 2-intervalles (disjoints) unitaires comme suit :*

1. *Un graphe $G$ est un graphe de 2-intervalles unitaire si et seulement si la famille $\mathcal{S}^*_{\mathcal{U}}(G)$ n'est pas vide.*

2. *Un graphe $G$ est un graphe de 2-intervalles disjoints unitaire si et seulement si la famille $\mathcal{S}_{\mathcal{U}}(G)$ n'est pas vide.*

Les caractérisations précédentes peuvent être généralisées à des dimensions supérieures, c'est-à-dire aux graphes de $d$-intervalles (disjoints) unitaires pour $d > 2$.

Enfin, nous expliquons comment obtenir un logiciel efficace pour reconnaître les graphes de 2-intervalles disjoints unitaires en encodant cette modélisation du problème dans le langage de programmation *Answer Set Programming (ASP)*, une forme de programmation déclarative orientée vers les problèmes de recherche difficiles.

## Généralisation de la caractérisation des graphes d'intervalles unitaires de Roberts

Dans le Chapitre 5, nous cherchons à généraliser la caractérisation des graphes d'intervalles unitaires de Roberts aux graphes de intervalles multiples. Il est clair que la condition nécessaire de la caractérisation de Roberts, à savoir être sans $K_{1,3}$ induit, s'étend naturellement aux graphes d'intervalles multiples : un graphe de 2-intervalles unitaire ne peut contenir un $K_{1,5}$ comme sous-graphe induit ; et plus généralement, un graphe de $d$-intervalles unitaire ne peut contenir un $K_{1,2d+1}$ comme sous-graphe induit.

Pour obtenir une caractérisation complète, la généralisation la plus simple consisterait peut-être à caractériser les graphes de $d$-intervalles unitaires comme des graphes de $d$-intervalles sans $K_{1,2d+1}$ induit, mais cela a déjà été prouvé faux dans [141] : il existe un graphe qui est de 2-intervalles et sans $K_{1,5}$ induit, mais qui n'est pas de 2-intervalles unitaire. Ainsi, nous étudions des cas plus restreints.

Déjà en 2016, Durán et al. ont décidé de se concentrer sur les graphes de $d$-intervalles qui sont également *d'intervalle* [55]. Dans une présentation au VII LAWCG, ils ont affirmé que si $G$ est un graphe d'intervalle, alors $G$ est un graphe de $d$-intervalles disjoints unitaire si et seulement s'il est sans $K_{1,2d+1}$ induit[2].

Nous montrons ici que cette affirmation est en réalité fausse, et que, de manière peut-être surprenante, la caractérisation de Roberts ne peut être généralisée que selon la définition choisie des graphes de $d$-intervalles ! (Voir Figure 3 pour un résumé des principaux résultats).

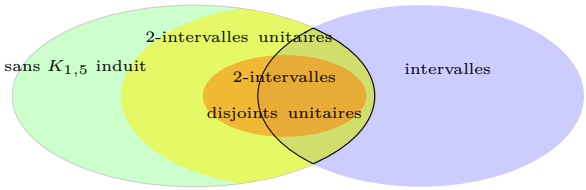



Figure 3.: Les graphes d'intervalles sans $K_{1,5}$ induit ne sont pas contenus dans la classe des graphes de 2-intervalles disjoints unitaires. La classe des graphes de 2-intervalles unitaires est une superclasse des graphes de 2-intervalles disjoints unitaires, et s'étend à toute l'intersection des graphes sans $K_{1,5}$ induit et des graphes d'intervalles.

Nous commençons par prouver que la caractérisation de Roberts des graphes d'intervalles unitaires peut être généralisée aux graphes de $d$-intervalles. Rappelons que par graphes de $d$-intervalles, nous faisons référence aux graphes d'intersection de $d$-intervalles où les $d$ intervalles ne sont *pas* nécessairement disjoints.

**Théorème.** *Soit $G$ un graphe d'intervalles. Alors, pour tout nombre naturel $d \geqslant 2$, $G$ est un graphe de $d$-intervalles unitaire si et seulement si $G$ ne contient pas une copie de $K_{1,2d+1}$ comme sous-graphe induit. De plus, étant donné un graphe d'intervalles sans $K_{1,2d+1}$ induit, une représentation de $d$-intervalles unitaire peut être construite en temps $\mathcal{O}(n+m)$, où $n$ et $m$ sont respectivement le nombre de sommets et d'arêtes du graphe.*

Pour prouver le théorème, nous présentons un algorithme en temps polynomial qui, étant donnée une représentation d'intervalles $\mathcal{I}$ d'un graphe sans $K_{1,2d+1}$ induit, renvoie une représentation de $d$-intervalles $\mathcal{I}'$ du graphe où aucun intervalle de la famille sous-jacente de $\mathcal{I}'$ n'intersecte trois intervalles disjoints ou plus. Cela garantit que la famille sous-jacente d'intervalles retournée correspond à une représentation d'un graphe

[2]Notez qu'ils se réfèrent aux $d$-intervalles disjoints unitaires simplement comme des $d$-intervalles unitaires, mais ils sont explicitement définis au préalable comme l'union de $d$ intervalles disjoints.

sans $K_{1,3}$ induit, donc nous pouvons utiliser l'algorithme décrit dans [21] pour le transformer en une représentation correcte (et ensuite en une représentation unitaire en temps linéaire [69]).

**Algorithme** Soit la famille d'intervalles $\mathcal{I}$ une représentation d'intervalles de $G$. Pour chaque intervalle $I \in \mathcal{I}$, notons $l(I)$ et $r(I)$ ses extrémités gauche et droite, respectivement. De plus, définissons un ordre partiel comme suit : étant donné deux intervalles $I, J \in \mathcal{I}$, notons $I \prec J$ si et seulement si $r(I) < l(J)$ (c'est-à-dire que l'intervalle $J$ est entièrement à droite de l'intervalle $I$). Deux intervalles sont incomparables s'ils s'intersectent.

**Étape 1** Initialiser un ensemble d'intervalles $\mathcal{C}$ avec tous les intervalles de $\mathcal{I}$, définir $\mathcal{I}' := \emptyset$, et passer à **Étape 2**.

**Étape 2** Choisir un intervalle $I$ de $\mathcal{C}$, le retirer de l'ensemble et définir son voisinage $\mathcal{N}(I) = \{J \in \mathcal{I} \colon J \cap I \neq \emptyset\}$. Soit $m$ le nombre maximum d'intervalles deux à deux disjoints que $I$ intersecte. Si $m \leqslant 2$, passer à **Étape 3** ; si $m = 3$, passer à **Étape 4** ; et si $m > 3$, passer à **Étape 5**.

**Étape 3** Si $m \leqslant 2$, ajouter l'intervalle $I_1 = I$ à la famille $\mathcal{I}'$ et appeler $I_1$ un intervalle original. Puis passer à **Étape 6**.

**Étape 4** Si $m = 3$, définir quatre intervalles auxiliaires :

$$A_1 = \arg\min_{J \in \mathcal{N}(I)} \{r(J)\} \qquad A_2 = \arg\min_{\{J \in \mathcal{N}(I) \colon A_1 \prec J\}} \{r(J)\}$$
$$A_4 = \arg\max_{J \in \mathcal{N}(I)} \{l(J)\} \qquad A_3 = \arg\max_{\{J \in \mathcal{N}(I) \colon J \prec A_4\}} \{l(J)\}$$

Ensuite, ajouter à $\mathcal{I}'$ le 2-intervalle $I_1 \cup I_2$, avec $I_1 = [l(I), r(A_2)]$ et $I_2 = [l(A_3), r(I)]$. Notez que $A_2$ et $A_3$ s'intersectent nécessairement, sinon nous aurions $m \geqslant 4$, donc $I_1 \cup I_2$ n'est pas un 2-intervalle disjoint. Après l'avoir ajouté à $\mathcal{I}'$, passer à **Étape 6**.

**Étape 5** Si $m > 3$, définir deux familles d'intervalles auxiliaires. La première famille $\mathcal{A} := \{A_i \,|\, i \in \{1, \ldots, m\}\}$ forme un ensemble maximum d'intervalles deux à deux disjoints intersectant $I$, et elle garantira que toutes les intersections sont préservées. Elle est définie comme suit :

$$A_1 = \arg\min_{J \in \mathcal{N}(I)} \{r(J)\} \qquad A_i = \arg\min_{\{J \in \mathcal{N}(I) \colon A_{i-1} \prec J\}} \{r(J)\} \ , \forall\ i \in \{2, \ldots, m-2\}$$
$$A_m = \arg\max_{J \in \mathcal{N}(I)} \{l(J)\} \quad A_{m-1} = \arg\max_{\{J \in \mathcal{N}(I) \colon J \prec A_m\}} \{l(J)\}$$

La deuxième famille $\mathcal{B} := \{B_i \,|\, i \in \{1, \ldots, m\}\}$ est un outil pour garantir que chaque nouvel intervalle $I_i$ intersecte seulement deux intervalles disjoints dans $\mathcal{I}'$.

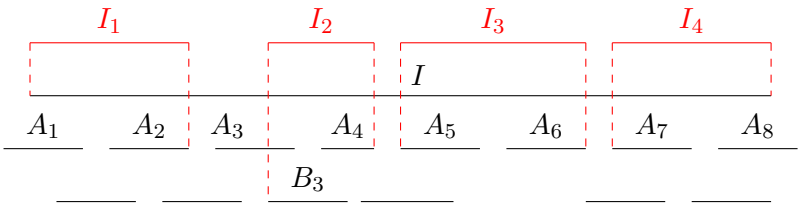



Figure 4.: L'intervalle $I$ intersecte 8 intervalles disjoints. En rouge, le 4-intervalle retourné par l'algorithme.

Ainsi, pour chaque $i \in \{1, \ldots, m\}$, $B_i$ est défini comme suit :

$$B_i = \underset{J \in \mathcal{N}(A_i) \cup A_i}{\arg\max} \{l(J)\}$$

Autrement dit, $B_i$ est l'intervalle dans le voisinage fermé de $A_i$ qui commence le plus tard. Notez que s'il n'existe aucun intervalle intersectant $A_i$ qui commence après $A_i$, alors $B_i = A_i$ car nous considérons le voisinage fermé. Maintenant, ajouter à $\mathcal{I}'$ le $t$-intervalle $I_1 \cup ... \cup I_t$, défini comme suit. Nous distinguons deux cas légèrement différents :

a) Si $m$ est pair, c'est-à-dire $m = 2t$ pour un $t > 1$, définir $I_1 = [l(I), r(A_2)]$, $I_i = [l(B_{2i-1}), r(A_{2i})]$ pour chaque $i \in \{2, \ldots, t-1\}$, et $I_t = [l(A_{2t-1}, r(I)]$.

b) Si $m$ est impair, c'est-à-dire $m = 2t - 1$ pour $t > 2$, définir $I_{t-1}$ et $I_t$ différemment, comme $I_{t-1} = A_{2t-3}$ et $I_t = [l(A_{2t-2}, r(I)]$, et les autres intervalles comme précédemment.

Remarquez qu'en vertu de la définition, les intervalles $I_i, ..., I_t$ sont effectivement deux à deux disjoints, donc si $m > 3$, le $t$-intervalle ajouté à $\mathcal{I}'$ est un $t$-intervalle disjoint. Après avoir ajouté l'intervalle $t$-intervalle, passer à **Étape 6**.

**Étape 6** Si $\mathcal{C} = \emptyset$, retourner $\mathcal{I}'$, sinon passer à **Étape 2**.

Figure 4 illustre l'algorithme appliqué à un exemple.

L'algorithme construit une représentation de $d$-intervalles unitaire, mais elle n'est pas disjointe : dans le cas des $K_{1,3}$ maximaux, les intervalles construits $I_1$ et $I_2$ s'intersectent. Cependant, dans le cas des $K_{1,m}$ maximaux avec $m > 3$, les $t$ intervalles du $t$-intervalle créé sont en fait disjoints deux à deux, donc si $G$ est un graphe d'intervalles et ne contient pas de $K_{1,2d+1}$ et ne contient aucun $K_{1,3}$ maximal, alors c'est également un graphe de $d$-intervalles disjoints unitaire. Avec une analyse plus attentive, nous pouvons inférer un corollaire encore plus fort, qui au lieu d'exiger l'absence totale de 3-griffes maximales, interdit seulement un sous-ensemble d'entre elles. Nous faisons référence à ces griffes interdites, qui sont exactement celles des 3-griffes maximales contenues dans un graphe $E$ induit, comme les griffes $E$. Rappelons qu'un graphe $E$ (ou $\text{star}_{1,2,2}$) est un graphe sur six sommets qui a comme ensemble d'arêtes un chemin $v_1, v_2, v_3, v_4, v_5$ et une arête supplémentaire $(v_3, v_6)$.

**Théorème.** *Soit $G$ un graphe sans $K_{1,2d+1}$ induit qui ne contient aucune griffe $E$. Alors, $G$ est un graphe de $d$-intervalles disjoints unitaire.*

Nous continuons en montrant qu'il n'y a aucun moyen d'étendre l'algorithme pour le faire fonctionner dans le cas général pour les graphes de $d$-intervalles disjoints unitaires. En particulier, nous prouvons le théorème suivant.

**Théorème.** *Il existe un graphe d'intervalles sans $K_{1,5}$ induit qui n'est pas un graphe de 2-intervalles disjoints unitaire.*

Pour prouver le théorème précédent, nous proposons le graphe $G$ dans Figure 5 comme certificat.

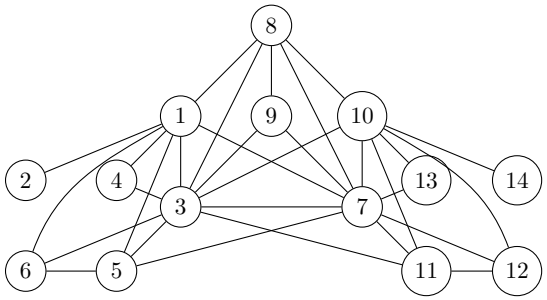



Figure 5.: Un graphe d'intervalles qui est sans $K_{1,5}$ mais pas un graphe de 2-intervalles disjoints unitaire.

Enfin, nous montrons que le théorème précédent peut en fait être généralisé pour les graphes de $d$-intervalles disjoints unitaires pour tout $d > 2$.

**Corollaire.** *Il existe un graphe d'intervalles sans $K_{1,2d+1}$ qui n'est pas un graphe de $d$-intervalles disjoints unitaire.*

Malgré ce résultat négatif, nous donnons une borne sur le nombre $t$ tel que $G$ soit un graphe de $t$-intervalles disjoints unitaires.

**Théorème.** *Soit $G$ un graphe d'intervalles ne contenant aucun $K_{1,2d+1}$ induit. Alors, $G$ est un graphe de $(d+1)$-intervalles disjoints unitaires.*

Pour conclure le Chapitre 5, nous résumons les relations d'inclusion entre les graphes de 2-intervalles unitaires, les graphes de 2-intervalles disjoints unitaires, les graphes de 2-intervalles équilibrés et les graphes de 2-intervalles disjoints équilibrés (voir Figure 6 pour une illustration graphique); et montrons qu'ils ne se généralisent pas pour les sous-classes de graphes de $d$-intervalles, car la classe des graphes de $d$-intervalles équilibrés n'est pas équivalente à la classe des graphes de $d$-intervalles disjoints équilibrés pour $d > 2$.

**Théorème.** *1. Les classes des* graphes de 2-intervalles *et des* graphes de 2-intervalles disjoints *sont* équivalentes.

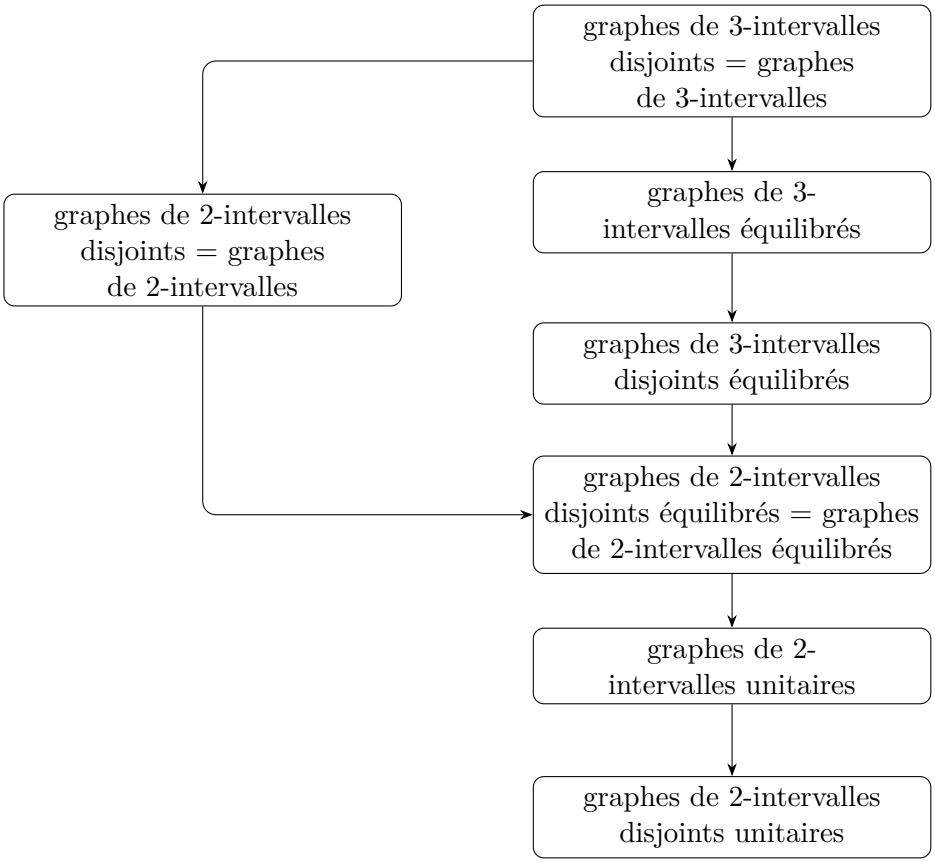



Figure 6.: Paysage des sous-classes de graphes de 3-intervalles. Une flèche allant d'une classe de graphes $\mathcal{C}$ à une classe $\mathcal{C}'$ indique que $\mathcal{C}' \subset \mathcal{C}$.

2. *Les classes des* graphes de 2-intervalles équilibrés *et des* graphes de 2-intervalles disjoints équilibrés *sont* équivalentes.

3. *La classe des* graphes de 2-intervalles unitaires *est* proprement contenue *dans la classe des* graphes de 2-intervalles disjoints équilibrés .

4. *La classe des* graphes de 2-intervalles disjoints unitaires *est* proprement contenue *dans la classe des* graphes de 2-intervalles unitaires.

**Théorème.** *La classe des graphes de 3-intervalles disjoints équilibrés est proprement contenue dans la classe des graphes de 3-intervalles équilibrés.*

**Corollaire.** *La classe des graphes de d-intervalles disjoints équilibrés est proprement contenue dans la classe des graphes de d-intervalles équilibrés pour tout nombre naturel* $d \geqslant 3$.

## Complexité de la Reconnaissance

Dans le Chapitre 6, nous prouvons d'abord que Disjoint Unit 2-Interval Recognition, c'est-à-dire, la reconnaissance des graphes de 2-intervalles disjoints unitaires, est NP-complet, et nous utilisons cela pour prouver la difficulté de reconnaître les graphes de $d$-intervalles disjoints unitaires pour tout $d \geqslant 2$. Le résultat pour $d = 2$ est obtenu en deux étapes. Nous définissons d'abord une version plus générale du problème, que nous appelons Colored disjoint unit 2-interval recognition, et prouvons qu'il est NP-complet. Ensuite, nous réduisons ce nouveau problème à la Disjoint Unit 2-Interval Recognition. Le problème plus général, Colored disjoint unit 2-interval recognition, est défini comme suit :

Colored disjoint unit 2-interval recognition

***Instance:*** Un graphe $G = (V, E)$ et une coloration $\gamma : V \to \{\texttt{white}, \texttt{black}\}$.
***Goal:*** Décidez si $G$ possède une représentation de 2-intervalles disjoints unitaire où :

- chaque sommet blanc est représenté par un 2-intervalle unitaire,
- chaque sommet noir est représenté par un intervalle unitaire.

Nous faisons référence à cette représentation comme une *représentation de 2-intervalles coloré unitaire.*

Pour prouver que la Colored disjoint unit 2-interval recognition est NP-complet, nous procédons par réduction à partir d'une variante du problème $SAT$ (dans ce qui suit, le terme $j$-clause fait référence à une clause qui contient exactement $j$ littéraux) :

**Lemme** ([60])**.** Satisfiability *est* NP*-complet même lorsqu'elle est restreinte aux formules CNF telles que :*

1. *Chaque clause contient soit 3 littéraux (3-clause), soit 2 littéraux (2-clause).*
2. *Chaque variable apparaît dans exactement une 3-clause.*
3. *Chaque 3-clause est monotone positive, c'est-à-dire qu'elle est composée de trois littéraux positifs.*
4. *Chaque variable apparaît exactement dans trois clauses, une fois négative et deux fois positive.*

**Théorème.** *Colored disjoint unit 2-interval recognition est* NP*-complet, même pour les graphes où les sommets blancs ont un degré au plus 6 et les sommets noirs ont un degré au plus 5.*

Étant donné une instance $\Psi$ de la variante de $SAT$ précédemment introduite, formée par un ensemble de variables booléennes $x_1, \ldots, x_n$ et un ensemble de clauses $C_1, \ldots, C_m$, nous construisons une instance équivalente $(G_\Psi, \gamma_\Psi)$ du problème Colored disjoint unit 2-interval recognition comme suit.

Pour chaque variable $x_i$, nous introduisons le gadget de variable $\hat{V}_i$, qui est le graphe coloré composé de trois sommets noirs $A_i$, $B_i$, $C_i$ et trois sommets blancs $x_i^1$, $x_i^2$ et $x_i^N$, avec toutes les arêtes entre un sommet noir et un sommet blanc, ainsi que les arêtes $(x_i^1, x_i^2)$, $(C_i, A_i)$ et $(C_i, B_i)$ (voir Figure 7)

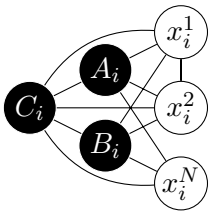



Figure 7.: Gadget de variable $\hat{V}_i$

Soit $C_\alpha$ une clause, pour $\alpha = 1, \ldots, m$. Si $C_\alpha$ est une clause à 3 littéraux, alors elle est positive monotone, c'est-à-dire que $C_\alpha = (x_i \vee x_j \vee x_k)$ pour certains $i, j, k \in \{1, \ldots, n\}$. Dans ce cas, nous introduisons les trois arêtes $(x_i^1, x_j^1)$, $(x_j^1, x_k^1)$, $(x_k^1, x_i^1)$, qui composent le gadget de la clause (voir Figure 8).

Si $C_\alpha$ est une clause à 2 littéraux, disons $C_\alpha = (x_i^r \vee x_j^s)$ avec $i, j \in \{1, \ldots, n\}$ et $r, s \in \{2, N\}$, alors nous introduisons un sommet noir public $L_{i,j}^\alpha$ avec un voisin noir privé $p_{i,j}^\alpha$ et nous ajoutons les quatre arêtes $(x_i^r, x_j^s)$, $(x_i^r, L_{i,j}^\alpha)$, $(x_j^s, L_{i,j}^\alpha)$ et $(L_{i,j}^\alpha, p_{i,j}^\alpha)$. Ces quatre arêtes ainsi que les deux sommets ajoutés composent le gadget de la clause.

Nous prouvons que $\Psi$ est satisfiable si et seulement si le graphe construit $G_\Psi = (V, E)$, avec $V = V_{\texttt{white}} \cup V_{\texttt{black}}$, admet une représentation de 2-intervalles coloré unitaire.

Nous montrons ensuite que Colored disjoint unit 2-interval recognition est réductible en temps polynomial à la Disjoint Unit 2-Interval Recognition (le problème qui consiste en déterminer si un graphe est un graphe de 2-intervalles disjoints unitaire), ce qui conduit au résultat principal du chapitre :

**Théorème.** *Disjoint Unit 2-Interval Recognition est* NP*-complet, même pour des graphes de degré au plus 7.*

Enfin, nous généralisons le résultat pour les graphes de $d$-intervalles disjoints unitaires, avec $d \geqslant 2$, qui n'est pas directement impliqué dans les problèmes de reconnaissance de graphes, et nous étendons le résultat de difficulté pour certains cas spécifiques de graphes à $d$-intervalles disjoints unitaires.

**Corollaire.** *La reconnaissance des graphes de d-intervalles disjoints unitaires de profondeur $r$ est* NP*-complet pour tout $r \geqslant 4$ et tout $d \geqslant 2$.*

**Corollaire.** *La reconnaissance des graphes de d-intervalles $(x, \ldots, x)$ est* NP*-complet pour tout $x \geqslant 11$ et tout $d \geqslant 2$.*

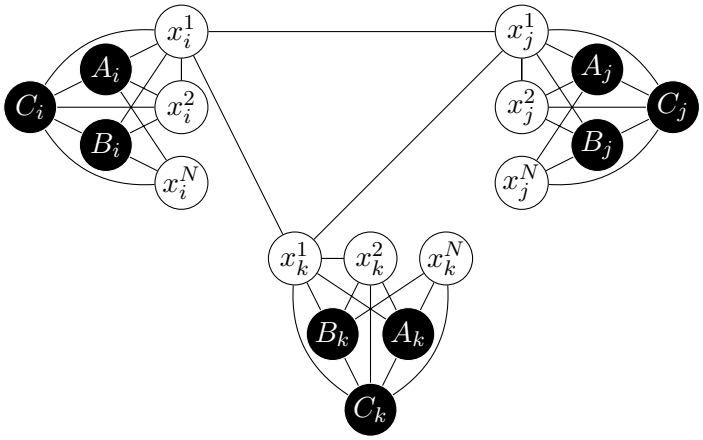



Figure 8.: Gadget de la clause à trois littéraux.

**Corollaire.** *À moins que l'ETH ne soit invalidé,* Disjoint Unit $d$-Interval Recognition *n'admet pas d'algorithme avec un temps d'exécution* $2^{o(|V|+|E|)}$.

Le dernier corollaire implique que nous ne pouvons espérer un algorithme significativement meilleur que celui par force brute pour la reconnaissance des graphes de $d$-intervalles disjoints unitaires, car cela est optimal sous l'ETH pour un $d$ fixé.

**Théorème.** *Disjoint Unit $d$-Interval Recognition peut être résolue en temps* $O(2^{d^2|E|})$.

Finalement, nous prouvons également que la reconnaissance des graphes de $d$-intervalles (non nécessairement disjoints) unitaires est également NP-complet, en adaptant la réduction précédente.

**Théorème.** *Unit $d$-interval recognition est* NP*-complet pour tout* $d \geqslant 2$*, même pour les graphes de profondeur au plus* 4.

## PIG-*completion*

Dans le Chapitre 7, nous étudions le problème d'édition suivant :

*Proper Interval Graph Completion (PIG-completion)*

| | |
|---|---|
| ***Instance:*** | Un graphe $G = (V, E)$. |
| ***Goal:*** | Retourner un ensemble d'arêtes $F$ (appelés *arêtes de remplissage*) tel que $G = (V, E \cup F)$ soit un graphe d'intervalles propre, avec $F \cap E = \emptyset$ et la cardinalité de $F$ soit minimale parmi tous les ensembles d'arêtes possibles. |

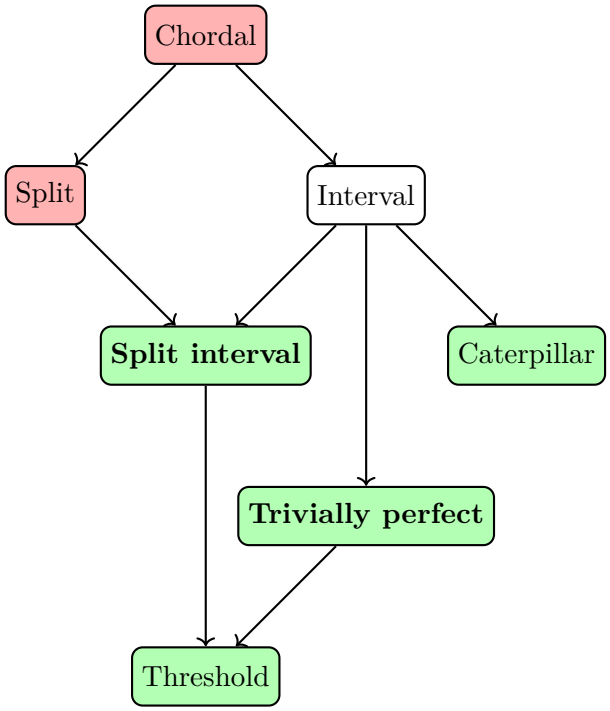



Figure 9.: Complexité de *PIG-completion* sur différentes sous-classes de graphes chordaux.

L'une des applications les plus connues de *PIG-completion* est la cartographie physique de l'ADN. Étant donné un ensemble de clones (intervalles contigus de la chaîne d'ADN) et des informations sur leurs chevauchements par paires, l'objectif de ce problème est de construire une carte décrivant la position relative des clones. Étant donné que ces informations sont obtenues expérimentalement, il peut y avoir des erreurs, c'est-à-dire des chevauchements non identifiés. Si nous modélisons cela comme un graphe où les clones sont représentés par des sommets et les chevauchements par des arêtes, le problème de construire une carte en supposant le moins d'erreurs possibles lorsque tous les clones ont une longueur égale est équivalent à *PIG-completion*.

Dross et al. ont montré que le problème *PIG-completion* reste NP-complet lorsque le graphe d'entrée est un graphe split, et devient polynomial s'il s'agit d'un graphe threshold ou d'un arbre caterpillar [54]. Étant donné que le problème est difficile sur les graphes chordaux (comme il l'est sur les graphes *split*) et polynomial sur certaines sous-classes de graphes d'intervalles (*threshold* et *caterpillar*), une question très naturelle à poser est de savoir si *PIG-completion* reste difficile lorsque le graphe d'entrée est un graphe d'intervalles. Le chapitre est consacré à la réponse à cette question. La Figure 9 illustre la complexité de *PIG-completion)* sur différentes sous-classes de graphes chordaux.

Nous étudions d'abord le problème de *PIG-completion* lorsque le graphe d'entrée est restreint aux graphes d'intervalles ayant un sommet universel (un sommet adjacent à tous les autres sommets du graphe). Dans ce cas, nous savons que la solution a une

forme très spécifique.

**Observation.** *Soit $G$ un graphe d'intervalles avec un sommet universel et soit $H$ une* PIG-completion *de $G$. Alors, les sommets de $H$ peuvent être partitionnés en deux ensembles $C_1$ et $C_2$ tels que chaque ensemble induit une clique dans $H$.*

**Lemme.** *Soit $G = (V, E)$ un graphe d'intervalles avec un sommet universel et soit $T$ son arbre MPQ associé. Soit $(C_1, C_2)$ une partition de $V$. Soit $w$ un nœud $P$ arbitraire de $T$ et soient $b_1, \ldots, b_k$ les enfants de $w$ et $T_B := \{T_{b_1}, \ldots, T_{b_k}\}$ l'ensemble des sous-arbres enracinés à chaque enfant. Alors, la partition $(C_1, C_2)$ induit une complétion sans cycles induits de longueur supérieure à 3 si et seulement si les trois conditions suivantes sont satisfaites :*

- *Pour chaque nœud $P$ $w$, il existe au plus un sous-arbre dans l'ensemble $T_B$ qui contient une paire de sommets $v_1, v_2$ telle que $v_1$ appartienne à $C_1$ et $v_2$ à $C_2$.*
- *Pour chaque nœud $Q$ $q$, il n'existe pas deux sections $S_i$ et $S_j$ avec des enfants $t_i$ et $t_j$ tels que les deux enfants soient non vides et que $T_{t_i}$ contienne une paire de sommets $v_1, v_2$ avec $v_1 \in C_1$ et $v_2 \in C_2$, tandis que le sous-arbre $T_{t_j}$ contienne une paire de sommets $u_1, u_2$ avec $u_1 \in C_1$ et $u_2 \in C_2$.*
- *Pour chaque nœud $Q$ $q$, il n'existe pas deux sections $S_i$ et $S_j$ telles que le sous-arbre $T_{S_i}$ contienne une paire de sommets $v_1, v_2$ avec $v_1 \in C_1$ et $v_2 \in C_2$, et que le sous-arbre $T_{S_j}$ contienne une paire de sommets $u_1, u_2$ avec $u_1 \in C_1$ et $u_2 \in C_2$, avec $v_1 \prec u_2$ et $v_2 \prec u_1$. Remarquez que si aucun des sommets n'est contenu dans l'une des sections, cette condition est équivalente à la deuxième.*

En utilisant le lemme précédent, nous concevons un algorithme de programmation dynamique qui exploite la structure du graphe d'intervalles d'entrée. L'optimalité de l'algorithme donne le résultat suivant.

**Théorème.** *Soit $G$ un graphe d'intervalles avec un sommet universel. Alors, il existe un algorithme polynomial qui calcule la* PIG-completion *de $G$ en temps $\mathcal{O}(n^3)$.*

**Algorithm**

Pour chaque nœud $v$ de l'arbre MPQ et chaque valeur $r \in [\lfloor |T_v|/2 \rfloor + 1]$, nous définissons :

$M_{v,r} = \min\{|F| : F$ est un ensemble d'arêtes de remplissage de $G[V(T_v)]$ qui donne une partition du sous-graphe en deux cliques avec la plus petite de taille $r\}$

- **Cas de base**: Si $v$ est une feuille,

$$M_{v,r} = \begin{cases} 0 & r \leqslant |v| \\ +\infty & \text{sinon} \end{cases}$$

- **Étape si $v$ est un nœud P**. Soit $v$ un nœud P avec $k+1$ enfants. Pour chaque $i \in [k]$, et chaque enfant $u$ de $v$ avec des frères et sœurs $b_1, \ldots, b_k$, nous définissons des sous-problèmes auxiliaires :

$$W_{u,i,r} = \begin{cases} M_{u,r} & \text{si } i = 0, \\ \min\Big\{ W_{u,i-1,r} + (|T_{B_{i-1}}| - r) \cdot |T_{b_i}| + M_{b_i,0}, & \\ \qquad W_{u,i-1,r-|T_{b_i}|} + (r - |T_{b_i}|) \cdot |T_{b_i}| + M_{b_i,0} \Big\} & \text{si } i > 0. \end{cases}$$

  La récurrence pour le problème original est alors donnée par :

$$\begin{aligned} M_{v*,r} &= \min\{W_{u,k,r} \mid u \text{ enfant de } v\} \\ M_{v,r} &= \min\{M_{v*,r-i} \mid i \in [|v|]\} \end{aligned}$$

- **Étape si $v$ est un nœud Q**. Soit $v$ un nœud Q avec des enfants $t_1, \ldots, t_m$ et des sections $S_1, \ldots, S_m$. Pour chaque $r$ dans $[|T_v|]$, nous définissons un sous-problème auxiliaire :
  - Cas de base :

$$\begin{cases} W_0 = \sum_i M_{t_i,0} + r(T_{t_i}) \cdot |T_{t_i}| + \sum_{x^l \in S_i} r(x) \\ c_{0,x} = \mathit{update}(x, 0),\ \forall x \\ u_{0,x} = 0\ \forall x \end{cases}$$

  - Étape :

$$\begin{cases} x & = \arg\min_x c_{r-1,x} \\ W_r & = W_{r-1} + c_{r-1,x},\ r > 0 \\ u_{r,x} & = u_{r-1,x} + 1 \text{ et } u_{r,y} = u_{r-1,y},\ \forall y \neq x \\ c_{r,x} & = \mathit{update}(x, u_{r,x}) \text{ et } c_{r,y} = c_{r-1,y},\ \forall y \neq x \end{cases}$$

  Ensuite, la récurrence pour le problème original est donnée par :

$$M_{v,r} = \min\{W_r, W_{|T_v|-r}\}$$

**Corollaire.** *Soit $G = (V, E)$ un graphe d'intervalles et soit $T$ son arbre MPQ associé. Supposons que pour chaque paire de sommets $u, v \in V$ tels que $u$ et $v$ soient tous deux des centres maximaux dans $G$, les centres $u$ et $v$ soient disjoints. Alors, nous pouvons calculer une* PIG-completion *de $G$ en temps $O(n^3)$.*

Nous terminons en montrant que l'algorithme peut être étendu pour donner une solution optimale lorsque le graphe d'intervalles d'entrée est arbitraire, mais la complexité devient exponentielle par rapport au nombre de centres des griffes maximales non disjoints. En particulier, cela implique que la *PIG-completion* est dans XP paramétré par

le nombre de centres d'intersection des griffes maximales. Cela implique un algorithme polynomial pour la classe des graphes d'intervalles *split*.

**Théorème.** *Soit $G$ un graphe d'intervalles* split*. Alors, nous pouvons calculer sa PIG-completion en temps polynomial.*

**Théorème.** *La* PIG-completion *est dans* XP *paramétré par le nombre de centres des griffes maximales non disjoints du graphe d'entrée.*

## Conclusion

Dans ce manuscrit, nous avons étudié les graphes d'intervalles (multiples) sous un angle structurel et algorithmique, ainsi qu'un problème de modification de graphes sur les graphes d'intervalles.

Nous avons commencé le Chapitre 4 en fournissant une caractérisation utile des graphes de 2-intervalles disjoints unitaires et un logiciel efficace pour tester si un graphe appartient à cette classe de graphes. Ensuite, dans le Chapitre 5, nous avons montré que les classes des graphes de $d$-intervalles disjoints unitaires et des graphes de $d$-intervalles unitaires ne sont pas équivalentes et nous avons obtenu une caractérisation complète des graphes de $d$-intervalles unitaires qui sont également des graphes d'intervalles. Cependant, pour la classe des graphes de $d$-intervalles disjoints unitaires qui sont également des graphes d'intervalles, seule une caractérisation partielle a été obtenue. Cela soulève la question de savoir si nous pouvons obtenir un résultat analogue, ou si nous pouvons même reconnaître cette classe en temps polynomial. Nous avons également étudié les relations entre les classes des graphes de $d$-intervalles équilibrés et des graphes de $d$-intervalles disjoints équilibrés. Néanmoins, certaines relations entre les sous-classes obtenues avec les deux définitions différentes de $d$-intervalle restent ouvertes, comme la question de savoir si la classe des graphes de $d$-intervalles unitaires est contenue dans la classe des graphes de $d$-intervalles disjoints équilibrés pour $d > 2$.

Dans le Chapitre 6, nous avons prouvé que la reconnaissance des graphes de $d$-intervalles unitaires et des graphes de $d$-intervalles disjoints unitaires de profondeur de représentation $r$ est NP-complet pour chaque $d \geqslant 2$ et chaque $r \geqslant 4$, et nous avons obtenu comme corollaire que la reconnaissance des graphes $(x, \ldots, x)$ $d$-intervalles est NP-complete pour $x \geqslant 11$. En plus de combler le fossé pour régler la complexité de la reconnaissance des graphes de $d$-intervalles $(x, \ldots, x)$ pour $2 \geqslant x \geqslant 10$, ou des graphes de $d$-intervalles de profondeur 3, une autre piste de recherche intéressante serait d'étudier le problème paramétré suivant :

Parameterized (disjoint) unit $d$-interval recognition

| | |
|---|---|
| ***Instance:*** | Un graphe arbitraire $G$ et un paramètre $k$. |
| ***Goal:*** | Déterminer si nous pouvons transformer $G$ en un graphe d'intervalles unitaire en divisant au plus $k$ de ses sommets. |

Une autre piste de recherche pertinente serait de savoir si, étant donné un graphe, nous pouvons obtenir une approximation à ratio constant de son *unit interval number*.

Enfin, dans le Chapitre 7, nous étudions le problème de modification de graphes *Proper interval graph-completion* sur les graphes d'intervalles. Bien que nous obtenions un algorithme en temps polynomial pour divers cas, y compris les graphes d'intervalles avec un sommet universel, les graphes d'intervalles split, ou les graphes d'intervalles sans centres intersectants de griffes maximales, la complexité dans le cas général reste ouverte. Néanmoins, nous fournissons un algorithme XP pour le cas général, paramétré par le nombre maximum de centres non disjoints de griffes maximales. Étant donné que le problème est FPT paramétré par le nombre d'arêtes à ajouter, il serait intéressant d'étudier s'il est également FPT par ce paramètre structurel. Une recherche supplémentaire pourrait se concentrer sur l'étude de la possibilité d'adapter l'algorithme basé sur la programmation dynamique sur l'arbre MPQ du graphe d'intervalles que nous avons développé ici pour la version de suppression d'arêtes du problème, ou pour des variantes qui admettent également l'ajout/suppression de sommets.

## RÉSUMÉ

Les graphes d'intervalles multiples sont une généralisation bien connue des graphes d'intervalles, où chaque sommet d'un graphe est représenté par un $d$-intervalle (l'union de $d$ intervalles) pour un certain nombre naturel $d > 1$, et il existe une arête entre deux sommets si et seulement si leurs $d$-intervalles correspondants se croisent. En particulier, un graphe de $d$-intervalles est unitaire si tous les intervalles de la représentation ont une longueur unitaire.
Dans cette thèse, nous étudions les graphes de $d$-intervalles unitaires d'un point de vue structurel et algorithmique. Dans la première partie, nous essayons de généraliser la caractérisation de Roberts des graphes d'intervalles unitaires, qui affirme qu'un graphe est un graphe d'intervalles unitaire si et seulement s'il est un graphe d'intervalles et ne contient pas le graphe biparti complet $K_{1,3}$ comme sous-graphe induit. Ensuite, nous passons à l'étude de la complexité de la reconnaissance des graphes d'intervalles multiples unitaires. Nous prouvons que, étant donné un graphe $G$, il est NP-difficile de déterminer si $G$ est un graphe de $d$-intervalles unitaires, et nous étendons ensuite ce résultat de difficulté à d'autres sous-classes de graphes de $d$-intervalles unitaires. Dans la dernière partie de ce manuscrit, nous nous concentrons sur le problème de PIG-completion, qui étant donné un graphe d'intervalles $G$, demande de trouver le nombre minimum d'arêtes à ajouter à $G$ pour qu'il devienne un graphe d'intervalles unitaire. Nous obtenons un algorithme polynomial lorsque $G$ contient un sommet adjacent à tous les autres sommets du graphe, et un algorithme XP paramétré par une propriété structurelle du graphe.



## MOTS CLÉS

Graphes d'intervalles, graphes d'intervalles multiples, graphe d'intervalles multiples unitaires, complexité, algorithmes



## ABSTRACT

Multiple interval graphs are a well-known generalization of interval graphs, where each vertex of a graph is represented by a $d$-interval (the union of $d$ intervals) for some natural number $d > 1$, and there exists an edge between two vertices if and only if their corresponding $d$-intervals intersect. In particular, a $d$-interval graph is unit if all the intervals on the representation have unit length.
In this thesis, we study unit $d$-interval graphs from a structural and an algorithmic perspective. In the first part, we try to generalize Roberts characterization of unit interval graphs, which states that a graph is unit interval if and only if it is interval and it does not contain the complete bipartite graph $K_{1,3}$ as an induced subgraph. Then, we move on to study the complexity of recognizing unit multiple interval graphs. We prove that given a graph $G$ it is NP-hard to determine whether $G$ is a unit $d$-interval graph, and then extend this hardness result to other subclasses of unit $d$-interval graphs. In the last part of this manuscript, we focus on the PIG-completion problem, where given an interval graph $G$, we are asked to find the minimum number of edges that we need to add to $G$ so that it becomes a proper interval graph. We obtain a polynomial algorithm when $G$ contains a vertex that is adjacent to every other vertex of the graph, and an XP algorithm parameterized by a structural property of the graph.



## KEYWORDS

Interval graphs, multiple interval graphs, unit multiple interval graphs, complexity, algorithms