M Ű E G Y E T E M 1 7 8 2

# Parameterized Complexity of Graph Modification and Stable Matching Problems

by Ildikó Schlotter

PhD Thesis

supervised by Dr. Dániel Marx

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Computer Science and Information Theory

2010

# Contents

# Acknowledgments

I would like to express my gratitude to my advisor Dániel Marx. The first steps towards this dissertation were motivated by the inspiring talks we had during the last semester of my undergraduate studies. In those months, he introduced me to the exciting world of parameterized complexity, and aroused my interest in theoretical computer science research. I feel lucky that he shared his thoughts with me, and enabled me to benefit from his great ideas and wide knowledge.

I would like to thank Katalin Friedl for her continuous support throughout these years. Not only did she undertake the duties of being my formal advisor during my PhD studies, but she was the person whom I could always turn to with any of my problems. Her kind advice helped me to overcome many difficulties.

I am grateful to my colleagues at the Department of Computer Science and Information Theory, Budapest University of Technology and Economics for their open-hearted attitude and their friendship. Especially, I thank András Recski whose work created a calm and open atmosphere at the department and ensured an environment suitable for scientific research. The financial support of the Hungarian National Research Fund grant OTKA 67651 is kindly appreciated.

In addition, I want to thank all the great teachers who showed me the beauty of mathematics. Katalin Tarcai was the first to encourage my mathematical interests in elementary school, and it were the classes of István Juhász and János Rácz which taught me the solid foundations of mathematics.

I would like to thank Péter Biró and David Manlove for their valuable advice on some of my papers, and also Rolf Niedermeier and Tibor Jordán for reading a preliminary version of the dissertation and providing me with helpful comments. I am also grateful for the wonderful and inspiring time we spent with Britta Dorn and Rolf Niedermeier in and around Tübingen.

And most importantly, I want to thank the immense love and patience of my family and my husband, Karesz.

# CHAPTER 1

# Introduction

Hard problems exist. Although this has always been common knowledge, the systematic study of computability only started in the 1930s. Numerous great mathematicians of that time such as Gödel [68], Turing [132, 133], Church [31, 32], and Post [116] worked out different mathematical foundations to capture the concept of computability and decidability, based on formal systems in logic, $\lambda$-calculus and recursion theory. In the forthcoming years, the Turing machine became the most successful theoretical model for computation. Based on this model, in 1965 Hartmanis and Stearns introduced a notion for describing the efficiency of algorithms, by measuring the worst-case time and space complexity required by an algorithm as a function of the input length [76]. In a few years, several important results were proved by researchers focusing on the connections between time and space complexities, and the underlying models of computation [131, 23, 14, 128, 33].

In 1965, Edmonds [49] argued that polynomial-time solvability is a good formalization of efficient computation. He noted that a wide range of problems can be solved by deterministic algorithms that run in polynomial time, many of them requiring only time that is a linear, quadratic, or cubic function of the length of the input. His work based the definitions of the complexity classes P and NP containing problems that are solvable in deterministic and non-deterministic polynomial time, respectively [48]. Since then, the most basic question arising in the study of any computational problem is whether we can solve the given problem by an algorithm whose worst-case running time is a polynomial function of the input length.

In 1971 and 1973, Cook [36] and Levin [91] independently defined the concept of NP-hardness, and showed the existence of NP-hard problems. Their work yielded a powerful tool to prove that some problems are computationally intractable. Following Karp [85], theoretical computer scientists in the 1970s identified a large class of NP-hard problems [62], appearing to be intractable by polynomial-time algorithms. Since these problems are ubiquitous in computer science, researchers have been developing numerous ways to deal with them. In the last two decades, parameterized complexity theory has proved to be successful in finding efficient algorithms for NP-hard problems, by offering a framework for the detailed investigation of hidden structural properties of such problems.

This thesis contains the study of several computationally hard problems in the context of parameterized complexity. In this framework, an integer $k$, called the *parameter*, is attributed to each instance of a given problem. Thanks to this notion, we can express the running time of an algorithm solving a problem as a function depending on both the size $n$ of the input and the parameter $k$, instead of regarding the running time only as a function that solely depends on $n$. This two-dimensional analysis allows us to study the complexity of a given problem in more detail.

The main objective of parameterized complexity is to develop efficient algorithms. We say that an algorithm is *fixed-parameter tractable* or FPT, if it has running time $f(k)n^c$ for some function $f$ and constant $c$. Though such algorithms can have exponential running time in general, they might still be efficient in practice, if the parameter $k$ is small. We say that a parameterized problem is FPT, if it admits such an algorithm. Then again, the parameterized complexity investigation of a problem might also reveal its *W[1]-hardness*, which shows that the problem is unlikely to be solvable by an FPT algorithm.

This thesis contributes to the parameterized complexity study of problems in connection to the following areas:

- **Apex graphs.** Planarity is a central notion in classical graph theory. We say that a graph is $k$-*apex*, if it contains $k$ vertices such that removing these vertices results in a planar graph. The celebrated results of Robertson and Seymour in graph minor theory [120, 121] imply that the problem of recognizing $k$-apex graphs is FPT, if $k$ is the parameter. However, the proof of this result is existential, and does not actually construct an FPT algorithm. Chapter 2 of this thesis presents an FPT algorithm for this problem.

- **Almost isomorphic graphs.** The complexity of deciding whether two graphs are isomorphic is among the most important open questions of complexity theory. Polynomial-time algorithms exist for special cases, such as the case where the input graphs are planar graphs [78] or interval graphs [96]. In this thesis, we study a variant of this problem, which asks whether two graphs can be made isomorphic by deleting $k$ vertices from the larger graph. We investigate the parameterized complexity of this graph modification problem is Chapters 3 and 4, with the parameter being the number $k$ of vertices to delete.

- **Stable matchings.** The classical Stable Matching or Stable Marriage problem deals with the following situation: we are given $n$ men and $n$ women, and each person ranks the members of the opposite sex in order of preference. The task is to find a matching (an assignment between men and women) that is stable in the following sense: there is no man-woman pair that are not matched to each other, and prefer each other to their partners in the matching. This problem is highly motivated by practice, as it models several situations where we face a matching problem in a two-sided market where preferences play an important role. Typically, the Stable Marriage problem has applications connected to admission systems, like the detailing process of the US Navy [118], the NRMP program for assigning medical residents in the USA [124, 127], or many other establishments in education [11, 2, 3].

  Although the Stable Marriage problem can be solved in linear time by the Gale-Shapley algorithm [60], many important extensions of the problem are computationally hard. Here, we consider two generalizations of this problem. In Chapter 5, we examine the Stable Marriage with Ties and Incomplete Lists problem, where the agents need not rank each member of the opposite sex, and the preference lists of the agents might also contain ties. Chapter 6 deals with the Hospitals/Residents with Couples problem, which is a generalization of the Stable Marriage problem that allows some agents (on the same side of the market) to form couples and have joint preferences.

In the cases where the obtained results show the computational hardness of some problem, we also investigate the theoretical possibilities for applying local search methods for the given problem. Local search is a metaheuristic that is widely applied to deal with optimization problems where an exact solution cannot be found in polynomial time. The basic mechanism of local search is to iteratively improve a given solution of the problem instance by using only local modification steps. The key task of this iterative method is to explore the local neighborhood of the actual solution. We can formulate this problem as follows: given an instance $I$ of the problem, a solution $S$ for $I$, and an integer $\ell$, find a solution $S'$ for $I$ in the $\ell$-neighborhood of $S$ such that $S'$ is better than $S$. Clearly, by searching a larger neighborhood, we can hope for a better solution. The efficiency of local search can be significantly improved if this neighborhood exploration task can be carried out effectively, allowing us to search relatively large neighborhoods in moderate time. We investigate this issue using the framework of parameterized complexity.

The main results of this thesis are listed below. For simplicity, we do not state our results in the most precise manner here. Some of the results only hold under some standard complexity theoretic assumptions.

- We give a quadratic FPT algorithm for recognizing $k$-apex graphs in Theorem 2.4.3, where the parameter is $k$.
- We investigate the complexity of the following problem: given two graphs $H$ and $G$, decide whether removing $k$ vertices from $G$ can result in a graph isomorphic to $H$. More precisely, we obtain the following results:
  - Theorem 3.1.6 yields an FPT algorithm with parameter $k$ for the case where $H$ and $G$ are both planar graphs, and $H$ is 3-connected.
  - Theorem 3.2.3 presents an FPT algorithm with parameter $k$ for the case where $H$ is a tree and $G$ can be arbitrary.
  - Theorem 4.3.1 gives an FPT algorithm with parameter $k$ for the case where $H$ and $G$ are both interval graphs.
  - Theorems 3.1.1 and 4.2.1 show NP-completeness for the cases where $H$ and $G$ are either both 3-connected planar graphs, or they are both interval graphs, respectively.
- We study the Stable Marriage with Ties and Incomplete Lists (or SMTI) problem, obtaining the following results:
  - Theorem 5.1.1 shows that finding the maximum stable matching is FPT, if the parameter is the total length of the ties.
  - Theorem 5.1.2 shows that finding the maximum stable matching is W[1]-hard, if the parameter is the number of ties.
  - Theorems 5.2.1 and 5.2.2 prove that no local search algorithm for this problem runs in FPT time where the radius of the explored neighborhood is considered as a parameter, and either all ties have length two, or we regard the number of ties as a parameter as well.
  - Theorems 5.3.1, 5.3.2, and 5.3.3 investigate the parameterized complexity of two optimization problems connected to SMTI, namely the Minimum Regret SMTI and the and the Egalitarian SMTI problems.

    Theorem 5.3.1 shows that both these problems are FPT if the parameter is the total length of the ties, but Theorems 5.3.2 and 5.3.3 present strong inapproximability results, stating that no approximation algorithms (with certain ratio guarantee) can have FPT running time for these problems, if the parameter is the number of ties.

- We study the Hospitals/Residents with Couples (or HRC) problem, obtaining the following results:
  - Theorem 6.1.1 states that deciding whether a stable assignment exists for an instance of the Hospitals/Residents with Couples problem is W[1]-hard, where the parameter is the number of couples.
  - Theorem 6.2.2 presents a local search algorithm for the maximization version of the HRC problem that runs in FPT time, if we regard both the number of couples and the radius of the explored neighborhood as a parameter. By contrast, Theorem 6.2.1 proves that no such algorithm can have FPT running time, if the only parameter is the radius of the neighborhood explored.
  - In Theorem 6.3.1, we prove that a simplified version of HRC where no preferences are involved, called the Maximum Matching with Couples problem, can be solved in randomized FPT time if the parameter is the number of couples.
  - Theorem 6.3.5 shows that no permissive local search algorithm for Maximum Matching with Couples runs in FPT time, if the parameter is the radius of the explored neighborhood.

The results in Chapters 2, 3, 4, 5, and 6 appear in the papers [104], [106], [108] [105], and [107], respectively.

The organization of the remaining sections of this chapter is the following. After fixing our notation in Section 1.1, we provide a brief introduction to parameterized complexity theory in Section 1.2. Then, we discuss the algorithmic aspects of local search methods in Section 1.3, and we give a short overview of the area of stable assignments in Section 1.4.

## 1.1 Notation

Let us introduce the notation we use throughout the thesis. We assume that the reader is familiar with the standard definitions of graph theory and classical complexity theory. For an overview on these topics, refer to [41] and [62]. Here we only introduce formal notation for some basic concepts, more complex definitions will be convenient to give later when they are used.

We write $[n]$ for the set $\{1,\dots,n\}$ and $\binom{[n]}{2}$ for the set $\{(i,j) \mid 1 \leq i < j \leq n\}$. The vertex set and the edge set of a graph $G$ are denoted by $V(G)$ and $E(G)$, respectively. We consider the edges of a graph as unordered pairs of its vertices. The set of the neighbors of $x \in V(G)$ is denoted by $N_G(x)$. For some $X \subseteq V(G)$ we let $N_G(X) = \bigcup_{x\in X} N_G(x) \setminus X$ and $N_G[X] = N_G(X) \cup X$. The degree of $x$ in $G$ is $d_G(x) = |N_G(x)|$. If $Z \subseteq V(G)$ and $G$ is clear from the context, then we let $N_Z(x) = N_G(x) \cap Z$ and $N_Z(X) = N_G(X) \cap Z$.

For some $X \subseteq V(G)$, $G - X$ is obtained from $G$ by deleting the vertices $X$, and the subgraph of $G$ induced by $X$ is $G[X] = G - (V(G) \setminus X)$. For a subgraph $G'$ of $G$, we let $G - G'$ be the graph $G - V(G')$. We say that two distinct subsets of $V(G)$ are *independent* in $G$, if no edge of $G$ runs between them. Otherwise, they are *neighboring*. For some vertex $x$, sometimes we will use only $x$ instead of $\{x\}$, but this will not cause any confusion.

A *matching* in $G$ is a set $M$ of edges such that no two edges in $M$ share an endpoint. If $x$ is an endpoint of some edge in $M$, then $x$ is *covered* by $M$. For some $x$ covered by $M$, we write $M(x) = y$ if $xy \in M$.

An *isomorphism* from a graph $G$ into a graph $G'$ is a bijection $\varphi : V(G) \cup E(G) \to V(G') \cup E(G')$ that maps vertices of $G$ to vertices of $G'$, edges of $G$ to edges of $G'$, and preserves incidence. For a subgraph $H$ of $G$, $\varphi(H)$ is the subgraph of $G'$ that consists of the images of the vertices and edges of $H$.

A graph is called *planar*, if it can be drawn in the plane such that its edges do not cross each other. For a more formal definition on planarity, see [41]. A planar graph together with a planar embedding is called a *plane* graph.

A *decision problem* in classical complexity theory can be described as follows. Given a finite alphabet $\Sigma$ and a set $Q \subseteq \Sigma^*$, the task of the decision problem corresponding to $Q$ is to decide whether a given input $w \in \Sigma^*$ is contained in $Q$ or not. When defining decision problems, we often omit $\Sigma$ and $Q$ from the definition, and instead we only describe the task of the problem. For example, the well known Clique problem can be defined as follows.

| Clique | |
|---|---|
| Input: | A graph $G$ and an integer $k$. |
| Task: | Decide whether $G$ has a clique of size $k$. |

Besides decision problems, we will often deal with *optimization problems* as well. In such problems, for each input $w$ we define the set $S(w)$ of feasible solutions and an objective function $T$ assigning a real value to each solution in $S(w)$. Given the input $w$, the task of the corresponding minimization or maximization problem is to find a feasible solution $S \in S(w)$ such that the value $T(S)$ is the smallest or largest possible, respectively. For example, the optimization version of Clique, called Maximum Clique, is the following.

| Maximum Clique | |
|---|---|
| Input: | A graph $G$. |
| Task: | Find a clique of $G$ that has maximum size. |

## 1.2 Parameterized complexity

In this section, we give a brief overview of the framework of parameterized complexity, which is the research methodology of this thesis. We start with describing the reasons that motivate the use of this framework in Section 1.2.1. After introducing the fundamental definition of fixed-parameter tractability in Section 1.2.2, Section 1.2.3 shows the various possibilities that can be applied when choosing the parameterization of a given problem. In Section 1.2.4, we present the toolkit of parameterized hardness theory which can be used to prove negative results.

All the definitions and arguments in this section are part of the standard framework of parameterized complexity. For a comprehensive introduction to this area, refer to the monograph of Downey and Fellows [44]. For more details and a newer perspective, see also [114] and [57].

### 1.2.1 Motivation

In the classical complexity investigation of a decision problem $Q$, we are usually interested in the question whether we can give an algorithm for $Q$ with running time polynomial in the size of the input, or show that $Q$ is NP-complete. This latter means that a polynomial-time algorithm for $Q$ would yield a polynomial-time algorithm for every decision problem in the class NP. Since this class contains many problems that are considered computationally hard, the NP-completeness of $Q$ is considered as a strong evidence suggesting that $Q$ does not admit a polynomial-time algorithm. Therefore, usually we only hope for an exponential time algorithm for such problems.

However, an NP-completeness result is never sufficient from a practical point of view. First, it does not give any suggestions to handle the given problem in practice. Clearly, even if a polynomial-time algorithm is out of reach, finding a moderately exponential-time algorithm might be of practical use. Comparing exponential-time algorithms is therefore an important goal.

Second, knowing that a problem is NP-complete does not yield too much insight into the problem. The classical complexity analysis only examines the running time of an algorithm as a function of the input size. However, there may exist other properties of the input that influence the running time in a crucial way. A more detailed, multidimensional analysis of the running time can help the understanding of the given problem, and can help finding more efficient solutions for it. In many cases, such an analysis shows that the hardness of the problem mainly depends on some decisive property of the input. In such a case, we can try to make restrictions on this property of the input on order to obtain algorithms with tractable running times.

The aim of parameterized complexity, a framework developed mainly by Downey and Fellows, is to address these problems. In this framework, a *parameter* in $\mathbb{N}$ is assigned to each input of a given problem $Q \subseteq \Sigma^*$. Hence a *parameterized problem* can be considered as a problem $Q \subseteq \Sigma^*$ together with a *parameterization function* $\kappa : \Sigma^* \to \mathbb{N}$ that assigns a parameter to each possible input. We study the running time of an algorithm solving $Q$ as a function of both the input size and the parameter value. Although we define the parameter as a nonnegative integer, the framework can easily handle those cases as well where the parameter is a tuple of integers.

To see an example for a parameterized problem, let us define the parameterized CLIQUE problem as follows.

CLIQUE (standard parameterization)
Input: a graph $G$ and an integer $k$.
Parameter: the integer $k$.
Task: decide whether $G$ has a clique of size $k$.

In CLIQUE, the parameter is the size of the clique we are looking for. In the DOMINATING SET problem, we are given a graph $G$ and an integer $k$, and the task is to find a *dominating set* of a graph $G$, i.e. a set of vertices $D \subseteq V(G)$ such that each vertex in $V(D) \setminus D$ is adjacent to some vertex in $D$. If we consider $k$ as the parameter, then this problem turns out to be hard from a parameterized viewpoint. However, when the parameter is not the solution size, but a crucial property of the input graph, called its *treewidth*, then the obtained parameterized problem becomes easier to deal with. For the definition of treewidth, see Section 2.1, for more on the topic refer to [18, 41].

DOMINATING SET (parameterized by treewidth)
Input: a graph $G$ and an integer $k$.
Parameter: the treewidth $t$ of $G$.
Task: decide whether $G$ has a dominating set of size $k$.

We can also investigate the dependence of the complexity of a problem on more than one parameters, like in the following example.

| | $n = 10^2$ | $n = 10^4$ | $n = 10^6$ | $n = 10^9$ |
|---|---|---|---|---|
| $k = 10$ | 3; 5; 22 | 5; 7; 44 | 7; 9; 66 | 10; 12; 99 |
| $k = 20$ | 3; 8; 42 | 5; 10; 84 | 7; 12; 126 | 10; 15; 189 |
| $k = 50$ | 5; 11; 102 | 5; 13; 204 | 7; 15; 306 | 10; 18; 459 |
| $k = 100$ | 10; 32; 202 | 10; 34; 404 | 10; 36; 606 | 11; 39; 909 |

Table 1.1: A table comparing different values of the functions $f_1(n,k) = 1.2738^k + kn$, $f_2(n,k) = 2^k n$, and $f_3(n,k) = n^{k+1}$. Each table entry contains the values $\lfloor \log_{10}(f_i(n,k)) \rfloor$ for $i = 1, 2, 3$, separated by semicolons.

$d$-Hitting Set (parameterized by $d$ and the solution size)
Input: a collection $\mathcal{C}$ of subsets of size at most $d$ of a set $S$, and an integer $k$.
Parameter: the integers $d$ and $k$.
Task: decide whether there is a set $S' \subseteq S$ with $|S'| \leq k$ such that no element in $\mathcal{C}$ is disjoint from $S'$.

As we will see in the next section, the parameterized complexity framework offers us a useful tool to handle computationally hard problems, by using the notion of fixed-parameter tractability.

### 1.2.2 FPT algorithms

In this section, we introduce the central concept of parameterized complexity, which is the definition of an FPT algorithm. But first, let us have a look on the parameterized version of the NP-hard Vertex Cover problem. Here the input is a graph $G$ and some integer $k$, with $k$ being the parameter, and the task is to decide whether $G$ has a vertex cover of size $k$. (A vertex cover of a graph is a set of vertices $S$ such that each edge in the graph has at least one endpoint in $S$.)

It is easy to see that Vertex Cover can be solved in $n^{k+1}$ time where $n$ is the number of vertices in the input graph, by trying all possibilities to choose $k$ vertices and check whether they form a vertex cover. Fortunately, more efficient algorithms are also known: one can show a very simple $O(2^k n)$ algorithm for it, but even a running time of $O(1.2738^k + kn)$ can be achieved [27].

Observe that if $k$ is relatively small compared to $n$, then these algorithms are more efficient than the trivial $n^{k+1}$ algorithm. To illustrate this, Table 1.1 shows a comparison of such running times for different values of $n$ and $k$. The key idea that leads to the efficiency of these algorithms is to restrict the combinatorial explosion only to the parameter, so that the exponential part of the running time is a function of $k$ only. This leads us to the following definition.

Given a parameterized problem, we say that an algorithm is *fixed-parameter tractable* or *FPT*, if its running time on an input $I$ having parameter $k$ can be upper bounded by $f(k)|I|^{O(1)}$ for some computable function $f$. Observe, that since the exponent of $|I|$ is a constant, the running time can only depend on the input size in a polynomial way. However, the function $f$ can be exponential or an even faster growing function, but note that it only depends on the parameter. We say that a parameterized problem is FPT, if it admits an FPT algorithm.

The usual aim of parameterized complexity is to find FPT algorithms for parameterized problems. The research for such algorithms has led to a wide range of fruitful techniques that are commonly used in designing both parameterized and classical (non-parameterized) algorithms. These techniques include bounded search tree methods, color-coding, kernelization, the application of graph minor theory, treewidth-based approaches and many others. Besides, parameterized complexity also includes a hardness theory, offering a useful negative tool for showing that some parameterized problem is unlikely to be FPT. Analogously to NP-hardness in classic complexity theory, there exist W[1]-hard problems which can be considered as problems that are hard from a parameterized viewpoint. We will give a short introduction to this area in Section 1.2.4.

In a typical NP-hard problem, there can be many relevant parts of the input that influence the complexity of the given problem. Thus, if a parameterized problem turns out to be intractable with a given parameterization, then considering another parameterization may lead to an efficient algorithm. We can also investigate whether considering more than one parameters might result in a fixed-parameter tractable problem. We discuss the topic of choosing the parameterization in Section 1.2.3.

### 1.2.3 Choosing the parameter

When parameterizing a problem, we have many possibilities to choose the parameter. It can be any part or property of the input that has relevant effects on the complexity of the given problem. Throughout the thesis we will present different types of parameterization. To name a few of the numerous possibilities, we present some examples.

One of the most frequently used parameterizations is the following "standard" parameterization of optimization problems. Given some optimization problem $Q$ with an objective function $T$ to, say, maximize, we can transform it into a decision problem by asking for a given an input $w$ of $Q$ and some integer $k$, whether there is a feasible solution $S$ for $w$ with $T(S) \geq k$. Now, the standard parameterization is to consider the objective value $k$ as the parameter. In many situations, the objective value is the size of a solution, so the standard parameterization in such cases this is to choose the parameter of the given problem to be the size of the solution.

The standard parameterization of Clique, where the parameter is the size of the clique to be found, is a simple example for this. A similar example, which has already been mentioned, is the standard parameterization of Vertex Cover where the parameter is the size of the desired vertex cover. The parameterized $k$-Apex Graph problem studied in this thesis also belongs to this class. In this problem, the task is to decide whether a given graph can be made planar by deleting $k$ vertices from it, and we consider $k$ as the parameter.

Another simple way to choose the parameter for a given problem is to consider such properties of the input as parameter that usually have crucial consequences on computational complexity. For example, if the input of a problem is a graph, then we can consider its maximum degree, its density, or its treewidth as the parameter. The general observation underlying such parameterizations is that graphs with small maximum degree, few edges, small treewidth, etc. tend to be tractable instances for many combinatorial problems. Considering geometrical problems, a similar parameter can be the dimension of the considered problem.

Another commonly used parameterization is to define the parameter of a given instance as its distance from some trivially solvable case. This parameterization catches the expectation that tractable instances should be close to some easy case of the problem. For instance, the Stable Marriage with Ties and Incomplete Lists problem can be solved in polynomial time in the case when there are no ties contained in the preference lists. Thus, the number of ties in an instance describes its distance from this easily solvable case. Hence, taking the number of ties to be the parameter, we can study how the complexity of this problem depends on this distance from triviality. Notice that the $k$-Apex Graph problem can also be considered as an example of this type of parameterization: since we can decide in linear time whether a graph is planar [78], we can regard the case when the parameter $k$ is zero as trivial.

Yet another intuitive parameterization is to pick a part of the problem that tends to be small in the practical instances. Since FPT algorithms are efficient when the parameter value is moderate, such a parameterization results in algorithms that are fast in practice. The Hospitals/Residents Assignment with Couples problem yields a good illustration for this parameterization. This problem can be solved in linear time if no couples are involved [60, 72], but the presence of couples makes it NP-hard. Now, since the number of couples is typically small compared to the total number of residents (and thus to the total size of the instance), taking the number of couples to be the parameter of an instance yields a parameterization that is useful in practice.

We can also mention the concept of dual parameterization, which leads to interesting problems in many cases. This idea can be best presented through an example: the dual of the standard parameterization of Vertex Cover asks whether a graph $G$ contains a vertex cover of size $n-k$, where $n = |V(G)|$ and $k$ is the parameter. Thus, the parameter is not the size of the vertex cover to be found, but the number of the remaining vertices. Since the complement of a vertex cover is always an independent set and vice versa, this problem is exactly the standard parameterization of the Independent Set problem, where given a graph $G$ and a parameter $k$, we ask whether $G$ has an independent set of size $k$.

Such a parameterization also appears in this thesis, arising in the study of the Induced Subgraph Isomorphism problem. The task of this problem is to determine whether for two given graphs $G$ and $H$ it holds that $H$ is an induced subgraph of $G$. The standard parameter of this problem is $|V(H)|$, the number of vertices of the smaller graph. The dual of this parameterization is to regard $|V(G)| - |V(H)|$ as the parameter, which makes sense in situations where the two graphs are close to each other in size.

Although we defined the parameter as a nonnegative integer, the framework can be extended in a straightforward way to handle cases where we have several nonnegative integers as parameters. For example, if $k_1, k_2, \dots, k_n \in \mathbb{N}$ are each regarded as parameters, then an FPT algorithm must have running time at most $f(k_1, k_2, \dots, k_n)|I|^{O(1)}$ for some function $f$, where $I$ denotes the given input of the algorithm. It can be easily seen that all definitions can be modified in a straightforward way in order to handle cases with more parameters.

However, there is also a simple trick that allows us to handle cases where we have more than one parameters from $\mathbb{N}$, without extending the framework directly. Clearly, given nonnegative integers $k_1$, $k_2$, and $n$, a function $T(n, k_1, k_2)$ can be bounded by $f(k_1, k_2)$ for some $f$ if and only if it can be bounded by $f'(k_1 + k_2)$ for some $f'$. Therefore, regarding $k_1, k_2, \dots, k_n \in \mathbb{N}$ as parameters is equivalent to setting $k = \sum_{i=1}^{n} k_i$ to be the unique parameter.

To see an example where it is natural to examine more than one parameters, consider the Stable Marriage with Ties and Incomplete Lists problem. This problem can be solved in linear time if no ties are contained in the given instance [60, 72], but in general it is NP-hard. Hence, ties play a key role in the complexity analysis of the problem. Therefore, a natural parameter is the number of ties appearing in the input. But as we will see in Chapter 5, the length of the ties is also important. This leads us to the parameterization where both the number of ties and the maximum length of ties in an instance are regarded as parameters.

### 1.2.4 Parameterized hardness theory

In this section, we describe the hardness theory of parameterized complexity, widely used for showing that some problem is not likely to admit an FPT algorithm. The theory of parameterized intractability is analogous to the theory of NP-hardness in the classical complexity theory. Just as polynomial-time reductions play a key role in the definition of NP-hardness, the theory of parameterized hardness also relies on the concept of reductions.

Let us give the definition of a parameterized reduction. Let $(Q, \kappa)$ and $(Q', \kappa')$ be two parameterized problems, meaning that $Q, Q' \subseteq \Sigma^*$ are two decision problems and $\kappa, \kappa' : \Sigma^* \to \mathbb{N}$ are the corresponding parameterization functions. We say that $L : \Sigma^* \to \Sigma^*$ is a *parameterized* or *FPT reduction* from $(Q, \kappa)$ to $(Q', \kappa')$ if there exist computable functions $f$ and $g$ such that the followings are true for every $x \in \Sigma^*$:

- $x \in Q$ if and only if $L(x) \in Q'$,
- if $\kappa(x) = k$ and $\kappa'(L(x)) = k'$, then $k' \leq g(k)$, and
- $L(x)$ can be computed in $f(k)|x|^{O(1)}$ time, where $k = \kappa(x)$.

Using this definition we can define the class of W[1]-hard problems. We say that a parameterized problem $(Q, \kappa)$ is *W[1]-hard* if there exists a parameterized reduction from the fundamental Short Turing Machine Acceptance problem to $(Q, \kappa)$. We will not need the exact definition of this problem, but basically its task is to decide whether a given nondeterministic Turing-machine can accept a given word in $k$ steps, where $k$ is the parameter. If any W[1]-hard problem would admit an FPT algorithm, then this would result in the collapse of the W-hierarchy. Namely, we would obtain fixed-parameter tractability for a whole class of problems, resulting in the equivalence of the classes FPT and W[1], which is considered highly unlikely. Thus, W[1]-hardness can be thought of a strong evidence showing that we cannot expect an FPT algorithm for the given problem.

The general technique to show that a problem $Q$ is W[1]-hard with parameterization $\kappa$ is to give an FPT reduction from an already known W[1]-hard problem to $(Q, \kappa)$. In particular, all W[1]-hardness proofs in this thesis contain a reduction from the W[1]-hard parameterized Clique problem (with the standard parameterization).

## 1.3 Local search and parameterized complexity

This section outlines the idea of local search and its connection to parameterized complexity. Sections 1.3.1 and 1.3.2 present standard knowledge in this field, but in Section 1.3.3 we present a new contribution that differentiates between two types of local search algorithms.

### 1.3.1 Local search

Local search is a simple and extremely useful metaheuristic that is widely applied in optimization problems [1]. Its basic idea is to start with a feasible solution of the problem, and then iteratively improve it by searching for better solutions in the local neighborhood of the actual solution. Thus, local search algorithms explore the space of feasible solutions by moving from solution to solution. In each step, the algorithm searches the neighborhood of the actual solution, and chooses the next solution by using only local information. The efficiency of such a method depends both on the neighborhood relation defined in the space of feasible solutions, and the strategy which determines the next solution in a given neighborhood.

In the most simple variant, the algorithm always chooses the best solution found in some small local neighborhood of the actual solution, and iterates this step until it either finds a solution that is better enough, or until some time bound elapses. Although this method, also called "hill-climbing", usually does not necessarily lead to an optimal solution, it is considered as a scalable heuristic.

In the last thirty years, many efforts have been made to improve the efficiency of this basic heuristic. Simulated annealing [88, 134] enhances the chance of finding an optimal solution by using a probabilistic method. Here, the next solution is chosen randomly according to a distribution that depends on the quality of the given solution (and also on a global variable describing "temperature"). Tabu search tries to increase the size of the solution space explored by maintaining a tabu list that contains solutions that have already been investigated [67]. Greedy randomized adaptive search procedure (GRASP) combines the local search technique with a greedy constructive approach [54].

Although all these approaches use the local search heuristic in a different way, each of them needs to explore the local neighborhood of a given solution efficiently. Clearly, there is a trade off between the expected quality of the solution to be found in a step and the size of the neighborhood we search. (See also the literature on very large-scale neighborhood search (VLNS) algorithms [7]). Thus, to speed up local search algorithms, we need to find algorithms that explore the neighborhood of a given solution efficiently. In the followings, we define this core step of local search more formally.

Let $Q$ be an optimization problem with an objective function $T$ which we want to, say, maximize. To define the concept of neighborhoods, we suppose there is some *distance* $d(S, S_0)$ defined for each pair $(S, S_0)$ of solutions for some instance $I$ of $Q$. We say that $S$ is *$\ell$-close* to $S_0$ if $d(S, S_0) \leq \ell$. Now, we can describe the task of a *local search algorithm for $Q$*: given an instance $I$ of $Q$, a solution $S_0$ for $I$, and some $\ell \in \mathbb{N}$, decide whether $I$ has a solution $S$ such that $T(S) > T(S_0)$ and $d(S, S_0) \leq \ell$. We will refer to this problem as the *local search* or *neighborhood search* problem for $Q$. Section 1.3.2 discusses the parameterized complexity aspects of this issue.

### 1.3.2 The parameterized neighborhood search problem

Although local search is a popular technique to handle computationally hard problems, investigations considering the connection of local search and parameterized algorithms have only been started a few years ago. However, research in this area has been gaining increasing attention lately [99].

In the previous section we formalized the neighborhood search problem that has to be solved many times when applying local search methods to deal with an optimization problem $Q$. Its task is the following: given an input instance $I$ of $Q$, an initial solution $S_0$ and an integer $\ell$, find a solution for $I$ that is better than $S_0$ but is $\ell$-close to $S_0$.

Typically, the $\ell$-neighborhood of a solution $S_0$ can be explored in $n^{O(\ell)}$ time by examining all possibilities to find those parts of $S_0$ that should be modified. Here, $n$ denotes the size of the input. However, in some cases the dependency on $\ell$ can be improved by getting $\ell$ out of the exponent of $n$, resulting in a running time of the form $f(\ell)n^{O(1)}$ for some function $f$. Observe that this means that the neighborhood search problem is fixed-parameter tractable with parameter $\ell$. Consequentially, it is natural to ask whether we can give an FPT algorithm for the local search problem with this parameterization.

This question has already been studied in connection with different optimization problems [86, 129]. Krokhin and Marx [90] investigated both the classical and the parameterized complexity of this neighborhood exploration problem for Boolean constraint satisfaction problems. They found a considerable amount of CSP problems that are NP-hard, but fixed-parameter tractable when parameterized by the radius of the neighborhood to be searched. As a negative example, Marx [103] showed that the neighborhood exploration problem is

W[1]-hard for the Metric Traveling Salesperson Problem. Fellows et al. [52] considered the applicability of local search for numerous classical problems such as Dominating Set, Vertex Cover, Odd Cycle Transversal, Maximum Cut and Minimum Bisection. They presented FPT algorithms solving the corresponding neighborhood exploration problems on sparse graphs (like graphs of bounded local treewidth and graphs excluding a fixed graph as a minor), and provided hardness results indicating that brute force search is unavoidable in more general classes of sparse graphs, like 3-degenerate graphs.

### 1.3.3 Strict and permissive local search

In this section, we contribute to the framework of the theory of local search by making a minor distinction between a "strict" and a "permissive" version of neighborhood search algorithms. We will refer to the standard formulation of the neighborhood search problem for some optimization problem $Q$ as the *strict local search* problem for $Q$. For simplicity, we assume that $Q$ is a maximization problem, with $T$ denoting its objective functions.

Strict local search for $Q$
Input: $(I, S_0, \ell)$ where $I$ is an instance of $Q$, $S_0$ is a solution for $I$, and $\ell \in \mathbb{N}$.
Task: If there exists a solution $S$ for $I$ such that $d(S, S_0) \leq \ell$ and $T(S) > T(S_0)$, then output such an $S$.

In contrast, a *permissive local search* algorithm for $Q$ is allowed to output a solution that is not close to $S_0$, provided that it is better than $S_0$. In local search methods, such an algorithm is as useful as its strict version. Formally, its task is as follows:

Permissive local search for $Q$
Input: $(I, S_0, \ell)$ where $I$ is an instance of $Q$, $S_0$ is a solution for $I$, and $\ell \in \mathbb{N}$.
Task: If there exists a solution $S$ for $I$ such that $d(S, S_0) \leq \ell$ and $T(S) > T(S_0)$, then output *any* solution $S'$ for $I$ with $T(S') > T(S)$.

Note that if an optimal solution can be found by some algorithm, then this yields a permissive local search algorithm for the given problem. Yet, finding a strict local search algorithm might be hard even if an optimal solution is easily found. An example for such a case is the Vertex Cover problem for bipartite graphs [90]. Besides, proving that no permissive local search algorithm exists for some problem is clearly harder than it is for strict local search algorithms. We also present results of this kind in this thesis.

## 1.4 Stable matchings

This section gives a brief overview of the area of stable matchings. First, we introduce the classical formulation of the Stable Marriage problem in Section 1.4.1. After that, we describe the extensions of this problem which allow the presence of ties or couples in Sections 1.4.2 and 1.4.3, respectively.

### 1.4.1 Classical stable matching problems

The Stable Marriage or Stable Matching problem was introduced by Gale and Shapley [60]. In the classical problem setting, we are given a set of women, a set of men, and a preference list for each person, containing a strict ordering of the members of the opposite sex. The task

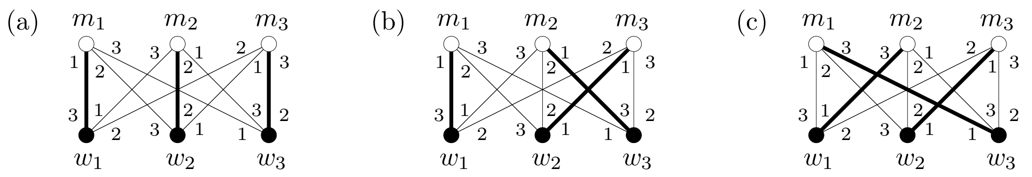


Figure 1.1: Illustration of a Stable Marriage instance. White circles represent men, black circles represent women. The small numbers on the edges represent the preferences. The matching represented by bold edges is unstable in Figure (a), but stable in Figures (b) and (c).

is to find a matching between men and women that is stable in the following sense: there are no man $m$ and woman $w$ both preferring each other to their partners in the given matching.

Figure 1.1 shows an example where there are three men $m_1, m_2, m_3$ and three women $w_1$, $w_2$, $w_3$ with preference lists as follows. We denote by $L(x)$ the preference list for a person $x$.

| | |
|---|---|
| $L(m_1)\colon w_1, w_2, w_3$ | $L(w_1)\colon m_2, m_3, m_1$ |
| $L(m_2)\colon w_3, w_2, w_1$ | $L(w_2)\colon m_3, m_2, m_1$ |
| $L(m_3)\colon w_2, w_1, w_3$ | $L(w_3)\colon m_1, m_3, m_2$ |

In this instance, the matching $\{m_1w_1, m_2w_2, m_3w_3\}$ shown in (a) of Figure 1.1 is not stable, because $m_3$ and $w_1$ both prefer each other to their partner in the matching. In such a case, we say that $m_3w_1$ is a blocking pair. Observe that $m_3$ and $w_2$ form a blocking pair as well. This instance can be easily seen to admit two stable matchings, which are depicted in (b) and (c) of Figure 1.1.

A slightly more general formulation of the Stable Marriage problem is the Hospitals/Residents problem, which was introduced by Gale and Shapley [60] to model the following situation. We are given a set of hospitals, each having a number of open positions, and a set of residents applying for jobs in the hospitals. Each resident has a ranking over the hospitals, and conversely, each hospital has a ranking over the residents. The task of the Hospitals/Residents problem is to assign as many residents to some hospital as possible, with the restrictions that the capacities of the hospitals are not exceeded and the resulting assignment is stable. We consider an assignment as unstable, if there is a hospital $h$ and a resident $r$ such that $r$ is not assigned to $h$, but both $h$ and $r$ would benefit from contracting with each other instead of accepting the given assignment. The number of residents that are assigned to some hospital is called the size of the assignment. Note that if each hospital has capacity one, then we obtain an instance of the Stable Marriage problem.

Both of these problems are linear-time solvable by the classical algorithm of Gale and Shapley [60, 72]. Given an instance of the Hospitals/Residents problem, their algorithm always finds a stable assignment. Moreover, the Gale-Shapely algorithm can even handle the case when the preference lists of the hospitals and the residents may be "incomplete", meaning that residents can refuse to be applied in some of the hospitals, and vice versa. It is also true that every stable assignment must have the same size, hence any stable assignment has maximum size [72], even in the case of incomplete preference lists.

The area of stable matchings has several practical application. Among others, we can mention the NRMP program for assigning medical residents in the USA [124, 127], the detailing process of the US Navy [118], or many application in the field of education like the admission system of higher education in Hungary [11]. Although understanding the original version of the Stable Marriage problem is of crucial importance, most of the applications motivate some kind of extension or modification of this problem. In the recent decade various versions have been investigated. For example, we can extend the model so that it allows

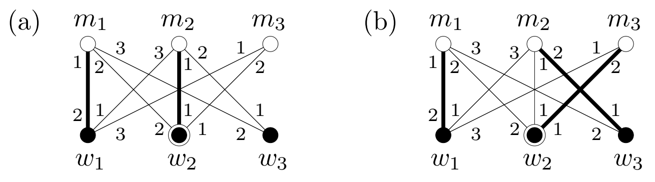


Figure 1.2: Illustration of an SMTI instance admitting two stable matchings of different sizes. Double black circles represent women who have ties in their preference lists.

preference lists to contain ties, or residents to form couples. The case when the market of the agents is one-sided, called the Stable Roommates problem [79, 82], and the case when the assignment may be a many-to-many matching [125, 13, 47] are also among the most frequently studied variants.

### 1.4.2 Ties

One of the most important generalizations of the classical Stable Marriage problem is the Stable Marriage with Ties and Incomplete Lists (or SMTI) problem, where we not only allow for incomplete preference lists, but we also allow ties in the preference list, meaning that the ordering represented by the preference lists may not be strict. This extension is highly motivated by practice, see e.g. the various applications in educational admission systems [11, 2, 3].

When ties are involved, the concept of stability has to be redefined. We consider a matching to be stable, if there are no man $m$ and woman $w$ both *strictly* preferring each other to their partners in the given matching. We remark that other definitions of stability are also in use, such as strong and super-stability [80]. The definition of stability used by us, which received the most attention, is sometimes referred to as weak stability.

It is easy to see that, by breaking ties in an arbitrary way and applying the Gale-Shapley algorithm, a stable matching can always be found. But as opposed to the case when no ties are allowed, stable matchings of various size may exist. To see an example, let us consider the following instance, containing men $m_1, m_2, m_3$ and women $w_1, w_2, w_3$ with the preference lists below. This instance contains one tie, present in the preference list of $w_2$, and both the lists $L(w_3)$ and $L(m_3)$ are incomplete, as they are unacceptable for each other. (Throughout this thesis, we denote ties in the preference lists of some SMTI instance with rounded parenthesis.)

| | |
|---|---|
| $L(m_1)\colon w_1, w_2, w_3$ | $L(w_1)\colon m_2, m_1, m_3$ |
| $L(m_2)\colon w_2, w_3, w_1$ | $L(w_2)\colon (m_2, m_3), m_1$ |
| $L(m_3)\colon w_1, w_2$ | $L(w_3)\colon m_2, m_1$ |

It is easy to verify that this instance admits two stable matchings. As Figure 1.2 shows, one of them has size two, and the other has size three.

When ties are present in an instance, the usual aim is to maximize the size of the stable matching. The resulting problem is called the Maximum Stable Marriage with Ties and Incomplete Lists (or MaxSMTI) problem. It has been proven that finding a stable matching of maximum size in this situation is NP-hard [84]. Since then, several researchers have attacked the problem, most of them presenting approximation algorithms [83, 87]. We study this problem in more detail in Chapter 5,

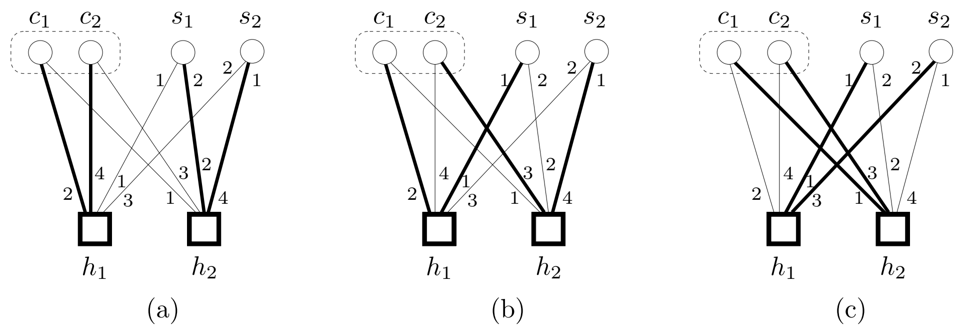


Figure 1.3: Illustration of a HRC instance. Circles represent residents and rectangles represent hospitals. Bold edges show the assignments $M_1$, $M_2$, and $M_3$ on figures (a), (b), and (c), respectively.

### 1.4.3 Couples

Here we introduce an extension of the Hospitals/Residents problem called Hospitals/Residents with Couples or HRC. In this problem, residents may form couples and thus have joint rankings over the hospitals. This means that instead of ranking the hospitals individually, couples rank *pairs of hospitals* according to their preferences. This allows them to express intentions such as being applied in the same hospital, or in hospitals that are close to each other. If we allow joint preferences, the notion of stability has to be adopted to fit this context as well. Although we only give the details later, together with the formal definition of the problem in Chapter 6, we show an intuitive example.

We define an instance of HRC that contains residents $s_1, s_2, c_1, c_2$ and hospitals $h_1$, $h_2$. Let $s_1$ and $s_2$ be singles, and let $(c_1, c_2)$ form a couple $c$. The capacity of both $h_1$ and $h_2$ is 2. The preference lists of the agents are the following (see Figure 1.3 for an illustration):

| | |
|---|---|
| $L(s_1)\colon h_1, h_2$ | $L(h_1)\colon s_1, c_1, s_2, c_2$ |
| $L(s_2)\colon h_2, h_1$ | $L(h_2)\colon c_1, s_1, c_2, s_2$ |
| $L(c)\colon (h_1, h_1), (h_2, h_2), (h_1, h_2)$ | |

Part (a) of 1.3 depicts an assignment $M_1$ that assigns both members of the couples $c$ to $h_1$ and both singles to $h_2$. This assignment is not stable, since the single $s_1$ and the hospital $h_1$ would both benefit from contracting each other, so they form a blocking pair for $M_1$. Part (b) of the figure shows an assignment $M_2$ where $s_1$ and $c_1$ are assigned to $h_1$, and $s_2$ and $c_2$ are assigned to $h_2$. Note that both the couples $c$ and the hospital $h_2$ would benefit from contracting each other. Therefore, they block the assignment, yielding that $M_2$ is unstable as well. Finally, we illustrate a stable assignment $M_3$ in (c), where $M_3$ assigns the singles to $h_1$, and both members of $c$ to $h_2$.

The task of the Hospitals/Residents with Couples problem is to decide whether a stable assignment exists in an instance where couples are involved. This problem was first introduced by Roth [124] who also discovered that a stable assignment need not exist. Later, Ronn [123] proved that the problem is NP-hard.

In Chapter 6 we will give an example showing that an instance of the Hospitals/Residents with Couples problem may admit stable assignments of different sizes. We denote by Maximum Hospitals/Residents with Couples the optimization problem where the task is to determine a stable assignment of maximum size for a given instance. Note that this problem is trivially NP-hard, as it contains the Hospitals/Residents with Couples problem.

We remark that HRC models a situation that arises in many real world applications

[127, 126]. In the last decade, various approaches have been investigated to deal with its intractability, but most researchers examined different assumptions on the preferences of couples that guarantee some kind of tractability [45, 26, 89, 112]. We examine this problem from different viewpoints in Chapter 6.

CHAPTER 2

# Recognizing $k$-apex graphs

In this chapter, we propose a newly developed FPT algorithm for the following problem: given a graph $G$, find a set of $k$ vertices whose deletion makes $G$ planar. We parameterize this problem by $k$, the number of vertices which we are allowed to delete.

Planar graphs are subject of wide research interest in graph theory. There are many generally hard problems which can be solved in polynomial time when considering planar graphs, e.g., Maximum Clique, Maximum Cut, and Subgraph Isomorphism [50, 73]. For problems that remain NP-hard on planar graphs, we often have efficient approximation algorithms. For example, the problems Independent Set, Vertex Cover, and Dominating Set admit an efficient linear-time approximation scheme [10, 94]. The research for efficient algorithms for problems on planar graphs is still very intensive.

Many results on planar graphs can be extended to almost planar graphs, which can be defined in various ways. For example, we can consider possible embeddings of a graph in a surface other than the plane. The genus of a graph is the minimum number of handles that must be added to the plane to embed the graph without any crossings. Although determining the genus of a graph is NP-hard [130], the graphs with bounded genus are subjects of wide research. A similar property of graphs is their crossing number, i.e., the minimum possible number of crossings with which the graph can be drawn in the plane. Determining the crossing number is also NP-hard [63].

Cai [24] introduced another notation to capture the distance of a graph $G$ from a graph class $\mathcal{F}$, based on the number of certain elementary modification steps. He defines the distance of $G$ from $\mathcal{F}$ as the minimum number of modifying steps needed to make $G$ a member of $\mathcal{F}$. Here, modification can mean the deletion or addition of edges or vertices. We consider the following question: given a graph $G$ and an integer $k$, is there a set of at most $k$ vertices in $G$, whose deletion makes $G$ planar?

It was proven by Lewis and Yannakakis [92] that the node-deletion problem is NP-complete for every non-trivial hereditary graph property decidable in polynomial time. As planarity is such a property, the problem of finding a maximum induced planar subgraph is NP-complete, so we cannot hope to find a polynomial-time algorithm that answers the above question. Therefore, the parameterized complexity framework seems suitable for the analysis of this problem.

The standard parameterized version of our problem is the following:

$k$-Apex
Input: a graph $G$ and an integer $k$.
Parameter: the integer $k$.
Task: decide whether deleting at most $k$ vertices from $G$ can result in a planar graph.

We refer to this parameterized problem as the $k$-Apex problem, because a set of vertices whose deletion makes the graph planar is sometimes called *apex vertices* or *apices*. We will denote the class of graphs for which the answer of the problem is 'yes' by Apex($k$). Observe that the parameter $k$ indeed expresses the distance of an input instance from planarity. We note that Cai, who also used the parameterized complexity framework for his examinations, used the notation Planar $+ kv$ to denote this class [24].

In the parameterized complexity literature, numerous similar node-deletion problems have been studied. A classical result of this type by Bodlaender [16] and Downey and Fellows [43] states that the Feedback Vertex Set problem, asking whether a graph can be made acyclic by the deletion of at most $k$ vertices, is FPT. The parameterized complexity of the directed version of this problem has been a long-standing open question, and it has only been proved recently that it is FPT as well [28]. Fixed-parameter tractability has also been proved for the problem of finding $k$ vertices whose deletion results in a bipartite graph [117], or in a chordal graph [101]. On the negative side, the corresponding node-deletion problem for wheel-free graphs was proved to be W[2]-hard [95].

Considering the graph class Apex($k$), we can observe that this family of graphs is closed under taking minors. The celebrated graph minor theorem by Robertson and Seymour states that such families can be characterized by a set of excluded minors [121]. They also showed that for each graph $H$ it can be tested in cubic time whether a graph contains $H$ as a minor [120]. As a consequence, membership for such graph classes can be decided in cubic time. In particular, we know that there exists an algorithm with running time $f(k)n^3$ that can decide whether a graph belongs to Apex($k$). However, the proof of the graph minor theorem is non-constructive in the following sense. It proves the existence of an algorithm for the membership test that uses the excluded minor characterization of the given graph class, but does not provide any algorithm for determining this characterization. In 2008, an algorithm was presented by Adler, Grohe, and Kreutzer [5] for constructing the set of excluded minors for a given graph class closed under taking minors, which yields a way to explicitly construct the algorithm whose existence was proved by Robertson and Seymour. We remark that it follows also from a paper by Fellows and Langston [53] that an algorithm for testing membership in Apex($k$) can be constructed explicitly.

Although these results provide a general tool that can be applied to our specific problem, so far no direct FPT algorithm has been proposed for it[1] In this chapter, we present a new algorithm which solves Apex($k$) in $f(k)n^2$ time. Note that the presented algorithm runs in quadratic time, and hence yields a better running time than any algorithm using the minor testing algorithm that is applied in the above mentioned approaches. Moreover, if $G \in$ Apex($k$) then our algorithm also returns a solution, i.e., a set $S \in V(G)$, $|S| \leq k$ such that $G - S$ is planar.

The presented algorithm is strongly based on the ideas used by Grohe [69] for computing crossing number. Grohe uses the fact that the crossing number of a graph is an upper bound for its genus. Since the genus of a graph in Apex($k$) cannot be bounded by a function of $k$, we need some other ideas. As in [69], we exploit the fact that in a graph with large treewidth we can always find a large grid minor [122]. Examining the structure of the graph with such a

[1]Recently, a paper by Ken-ichi Kawarabayashi with title *Planarity allowing few error vertices in linear time* has been accepted to FOCS 2009 proposing an algorithm for this problem.

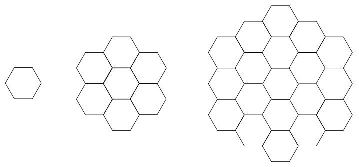


Figure 2.1: The hexagonal grids $H_1$, $H_2$, and $H_3$.

grid minor, we can reduce our problem to a smaller instance. Applying this reduction several times, we finally get an instance with bounded treewidth. Then we make use of Courcelle's Theorem [38], which states that every graph property that is expressible in monadic second-order logic can be decided in linear time on graphs of bounded treewidth.

In Section 2.1 we summarize some useful definitions used in this chapter. Section 2.2 outlines the FPT algorithm solving the $k$-Apex problem, and Sections 2.3 and 2.4 describe the two phases of the algorithm. The results of this chapter were published in [104].

## 2.1 Preliminaries

In this section, graphs are assumed to be simple, since both loops and multiple edges are irrelevant in the $k$-Apex problem.

A graph $H$ is a *minor* of a graph $G$ if it can be obtained from a subgraph of $G$ by contracting some of its edges. Here *contracting an edge* $e$ with endpoints $a$ and $b$ means deleting $e$, and then identifying vertices $a$ and $b$.

A graph $H$ is a *subdivision* of a graph $G$ if $G$ can be obtained from $H$ by contracting some of its edges that have at least one endpoint of degree two. Or, equivalently, $H$ is a subdivision of $G$ if $H$ can be obtained from $G$ by replacing some of its edges with newly introduced paths such that the inner vertices of these paths have degree two in $H$. We refer to these paths in $H$ corresponding to edges of $G$ as *edge-paths*. A graph $H$ is a *topological minor* of $G$ if $G$ has a subgraph that is a subdivision of $H$. We say that $G$ and $G'$ are *topologically isomorphic* if they both are subdivisions of a graph $H$.

The $g \times g$ *grid* is the graph $G_{g\times g}$ where $V(G_{g\times g}) = \{v_{ij} \mid 1 \le i,j \le g\}$ and $E(G_{g\times g}) = \{v_{ij}v_{i'j'} \mid |i-i'| + |j-j'| = 1\}$. Instead of giving a formal definition for the *hexagonal grid* of radius $r$, which we will denote by $H_r$, we refer to the illustration shown in Figure 2.1. A *cell* of a hexagonal grid is one of its cycles of length 6.

A *tree decomposition* of a graph $G$ is a pair $(T, (V_t)_{t\in V(T)})$ where $T$ is a tree, $V_t \subseteq V(G)$ for all $t \in V(T)$, and the following are true:

- for all $v \in V(G)$ there exists a $t \in V(T)$ such that $v \in V_t$,
- for all $xy \in E(G)$ there exists a $t \in V(T)$ such that $x, y \in V_t$,
- if $t$ lies on the path connecting $t'$ and $t''$ in $T$, then $V_t \supseteq V_{t'} \cap V_{t''}$.

The *width* of such a tree decomposition is the maximum of $|V_t|-1$ taken over all $t \in V(T)$. The *treewidth* of a graph $G$, denoted by $\text{tw}(G)$, is the smallest possible width of a tree decomposition of $G$. For an introduction to treewidth see e.g. [18, 41].

## 2.2 Overview of the algorithm

Here we outline an algorithm *Apex* which solves the $k$-Apex problem in time $f(k)n^2$ for some function $f$, where $n$ is the number of vertices in the input graph. Algorithm *Apex* works in two phases. In the first phase (Section 2.3) we compress the given graph repeatedly, and finally either conclude that there is no solution for our problem or construct an equivalent problem instance with a graph having bounded treewidth. The bounded treewidth case is solved in the second phase of the algorithm (Section 2.4) by applying Courcelle's Theorem which gives a linear-time algorithm for the evaluation of MSO-formulas on bounded treewidth graphs.

To describe the first step of our algorithm, we need some deep results from graph minor theory. The following result states that every graph having large treewidth must contain a large grid as a minor.

**Theorem 2.2.1** (Excluded Grid Theorem, [119]). *For every integer $r$ there exists an integer $w(r)$ such that if $\mathrm{tw}(G) > w(r)$ then $G$ contains $G_{r\times r}$ as a minor.*

The grid minor guaranteed by this theorem in the case when the treewidth of the graph $G$ is large can be found in cubic time. However, we need a linear-time algorithm for finding a large grid minor, so we have to make use of the following result, which states that if the graph is planar, then the bound on $w(r)$ is linear:

**Theorem 2.2.2** (Excluded Grid Theorem for Planar Graphs, [122]). *For every integer $r$ and every planar graph $G$, if $\mathrm{tw}(G) > 6r - 5$ then $G$ contains $G_{r\times r}$ as a minor.*

Also, we will use the following algorithmic results:

**Theorem 2.2.3** ([17, 115]). *For every fixed integer $w$ there exists a linear-time algorithm that, given a graph $G$, does the following:*

- *either produces a tree decomposition of $G$ of width at most $w$, or*
- *outputs a subgraph $G'$ of $G$ with $\mathrm{tw}(G') > w$, together with a tree decomposition of $G'$ of width at most $2w$.*

**Theorem 2.2.4** ([9]). *For every fixed graph $H$ and integer $w$ there exists a linear-time algorithm that, given a graph $G$ and a tree decomposition for $G$ of width $w$, returns a minor of $G$ isomorphic to $H$, if this is possible.*

Now, we are ready to state our first lemma, which provides the key structures for the mechanism of our algorithm. In this lemma, we focus on hexagonal grids instead of rectangular grids. The reason for this is the well-known fact that if a graph of maximum degree three is a minor of another graph, then it is also contained in it as a topological minor [41]. This property of the hexagonal grid will be very useful later on.

**Lemma 2.2.5.** *For every pair of fixed integers $r$ and $k$ there is a linear-time algorithm GridStructure, that, given an input graph $G$, does the following:*

- *either produces a tree decomposition of $G$ of width $w(r,k) = 24r - 11 + k$, or*
- *finds a subdivision of $H_r$ in $G$, or*
- *correctly concludes that $G \notin \mathrm{Apex}(k)$.*

*Proof.* Let $r$ and $k$ be arbitrary fixed integers. Run the algorithm provided by Theorem 2.2.3 for $w = w(r,k)$ on graph $G$. If it produces a tree decomposition of width $w(r,k)$ for $G$, then we output it. Otherwise let $G'$ be the subgraph of $G$ with $\mathrm{tw}(G') > w(r,k)$ that has been provided together with a tree decomposition $\mathcal{T}'$ for it having width at most $2w(r,k)$.

On the one hand, if $G' \notin \mathrm{Apex}(k)$, then $G \notin \mathrm{Apex}(k)$ also holds as $G'$ is a subgraph of $G$. On the other hand, if $G' \in \mathrm{Apex}(k)$, then there exists a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G' - S$ is planar. Deleting a vertex of a graph can only decrease its treewidth by at most one, so $\mathrm{tw}(G' - S) > w(r,k) - k = 6(4r-1) - 5$. Now, Theorem 2.2.2 implies that $G' - S$ contains $G_{(4r-1)\times(4r-1)}$ as a minor. Since the hexagonal grid with radius $r$ is a subgraph of the $(4r-1)\times(4r-1)$ grid, we get that $G' - S$ must also contain $H_r$ as a minor, and hence as a topological minor.

Thus, we get that either $G \notin \mathrm{Apex}(k)$, or $G'$ (and hence $G$) contains $H_r$ as a (topological) minor. Now, using the algorithm of Theorem 2.2.4 for $G'$ and $\mathcal{T}'$, we can find a subgraph of $G'$ isomorphic to a subdivision of $H_r$ in linear time, if possible. If the algorithm produces such a subgraph, then we output it, otherwise we can correctly conclude that $G \notin \mathrm{Apex}(k)$. $\square$

In algorithm *Apex* we will run *GridStructure* several times. As long as the result is a hexagonal grid of radius $r$ as topological minor, we will run Phase I of algorithm *Apex*, which compresses the graph $G$. If at some step algorithm *GridStructure* gives us a tree decomposition of width $w(r,k)$, we run Phase II. (The constant $r$ will be fixed later.) And of course if at some step *GridStructure* finds out that $G \notin \mathrm{Apex}(k)$, then algorithm *Apex* can stop with the output "No solution."

Clearly, we can assume without loss of generality that the input graph is simple, and it has at least $k+3$ vertices. So if $G \in \mathrm{Apex}(k)$, then deleting $k$ vertices from $G$ (which means the deletion of at most $k(|V(G)|-1)$ edges) results in a planar graph, which has at most $3|V(G)|-6$ edges. Therefore, if $|E(G)| > (k+3)|V(G)|$ then surely $G \notin \mathrm{Apex}(k)$. Since this can be detected in linear time, we can assume that $|E(G)| \leq (k+3)|V(G)|$.

## 2.3 Phase I of Algorithm *Apex*

In Phase I we assume that after running *GridStructure* on $G$ we get a subgraph $H'_r$ that is a subdivision of $H_r$. Our goal is to find a set of vertices $X$ such that $G-X$ is planar, and $|X| \leq k$. Let $\mathrm{ApexSets}(G,k)$ denote the family of sets of vertices that have these properties, i.e., let $\mathrm{ApexSets}(G,k) = \{X \subseteq V(G) \mid |X| \leq k \text{ and } G - X \text{ is planar}\}$. Since the case $k=1$ is very simple we can assume that $k > 1$.

**Reduction A: Flat zones.** In the following we regard the grid $H'_r$ as a fixed subgraph of $G$. Let us define $z$ zones in it. Here $z$ is a constant depending only on $k$, which we will determine later. A *zone* is a subgraph of $H'_r$ which is topologically isomorphic to the hexagonal grid $H_{2k+5}$. We place such zones next to each other in the well-known radial manner with radius $q$, i.e., we replace each hexagon of $H_q$ with a subdivision of $H_{2k+5}$. It is easy to show that in a hexagonal grid with radius $(q-1)(4k+9)+(2k+5)$ we can define this way $3q(q-1)+1$ zones that only intersect in their outer circles. So let $r = (q-1)(4k+9)+(2k+5)$, where we choose $q$ big enough to get at least $z$ zones, i.e., $q$ is the smallest integer such that $3q(q-1)+1 \geq z$. Let the set of these innerly disjoint zones be $\mathcal{Z}$, and the subgraph of these zones in $H'_r$ be $R$.

Let us define two types of *grid-components*. An edge which is not contained in $R$ is a grid-component if it connects two vertices of $R$. A subgraph of $G$ is a grid-component if it is a (maximal) connected component of $G-R$. A grid-component $K$ is *attached* to a vertex $v$ of the grid $R$ if it has a vertex adjacent to $v$, or (if $K$ is an edge) one of its endpoints is $v$. The *core* of a zone is the (unique) subgraph of the zone which is topologically isomorphic to $H_{2k+3}$ and lies in the middle of the zone. Let us call a zone $Z \in \mathcal{Z}$ *open* if there is a

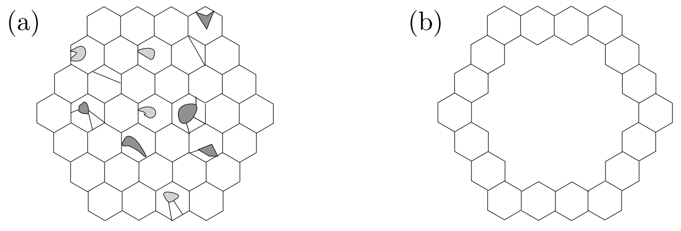


Figure 2.2: (a) An induced subgraph of a flat zone, together with its grid components. Among them, there are two edges, four edge-components (shown in light gray) and five cell-components (dark gray). (b) The ring $R_3$ of $Z$.

vertex in its core that is connected to a vertex $v$ of another zone in $\mathcal{Z}$, $v \notin V(Z)$, through a grid-component. A zone is *closed* if it is not open.

For a subgraph $H$ of $R$ we let $T(H)$ denote the subgraph of $G$ induced by the vertices of $H$ and the vertices of the grid-components which are only attached to $H$. Let us call a zone $Z$ *flat* if it is closed and $T(Z)$ is planar. Let $Z$ be such a flat zone. See Figure 2.2 (a) for an illustration of a flat zone together with its grid-components. A grid-component is an *edge-component* if it is either only attached to one edge-path of $Z$ or only to one vertex of $Z$. Otherwise, it is a *cell-component* if it is only attached to vertices of one cell. As a consequence of the fact that all embeddings of a 3-connected graph are equivalent (see e.g. [41]), and $Z$ is a subdivision of such a graph, every grid-component attached to some vertex in the core of $Z$ must be one of these two types. Note that we can assume that in an embedding of $T(Z)$ in the plane, all edge-components are embedded in an arbitrarily small neighborhood of the edge-path (or vertex) which they belong to.

Let us define the *ring* $R_i$ ($1 \leq i \leq 2k+4$) as the union of those cells in $Z$ that have common vertices both with the $i$-th and the $(i+1)$-th concentric circle of $Z$. Let $R_0$ be the cell of $Z$ that lies in its center. The zone $Z$ can be viewed as the union of $2k+5$ concentric rings, i.e., the union of the subgraphs $R_i$ for $0 \leq i \leq 2k+4$. Figure 2.2 (b) depicts the ring $R_3$.

**Lemma 2.3.1.** *Let $Z$ be a flat zone in $R$, and let $G'$ be the graph $G - T(R_0)$. Then $X \in \mathrm{ApexSets}(G', k)$ implies $X \in \mathrm{ApexSets}(G, k)$.*

*Proof.* Suppose $X \in \mathrm{ApexSets}(G', k)$. Since $G - T(R_0) - X$ is planar, we can fix a planar embedding $\phi$ of it. If $R_i \cap X = \emptyset$ for some $i$ ($2 \leq i \leq 2k+2$) then let $W_i$ denote the maximal subgraph of $G - T(R_0) - X$ for which $\phi(W_i)$ is in the region determined by $\phi(R_i)$ (including $R_i$). If $R_i \cap X$ is not empty then let $W_i$ be the empty graph. Note that if $2 \leq i \leq 2k$ then $W_i$ and $W_{i+2}$ are disjoint. Therefore, there exists an index $i$ for which $W_i \cap X = \emptyset$ and $W_i$ is not empty. Let us fix this $i$.

Let $Q_i$ denote $T(\bigcup_{j=0}^{i} R_j)$. We prove the lemma by giving an embedding for $G - X'$ where $X' = X \setminus V(Q_{i-1})$. The region $\phi(R_i)$ divides the plane in two other regions. As $Z$ is flat, vertices of $Q_{i-1}$ can only be adjacent to vertices of $Q_i$. Thus we can assume that in the finite region only vertices of $Q_{i-1}$ are embedded, so $G - X' - (Q_{i-1} \cup W_i)$ is entirely embedded in the infinite region. Let $U$ denote those vertices in $Q_{i-1}$ which are adjacent to some vertex in $G - Q_{i-1}$. Observe that the vertices of $U$ lie on the $i$-th concentric circle of $Z$, hence, the restriction of $\phi$ to $G - X' - (Q_{i-1} - U)$ has a face whose boundary contains $U$.

Now let $\theta$ be a planar embedding of $T(Z)$, and let us restrict $\theta$ to $Q_{i-1}$. Note that $U$ only contains vertices which are either adjacent to some vertex in $R_i$ or are adjacent to cell-components belonging to a cell of $R_i$. But $\theta$ embeds $R_i$ and its cell-components also, and therefore the restriction of $\theta$ to $Q_{i-1}$ results in a face whose boundary contains $U$. Here we

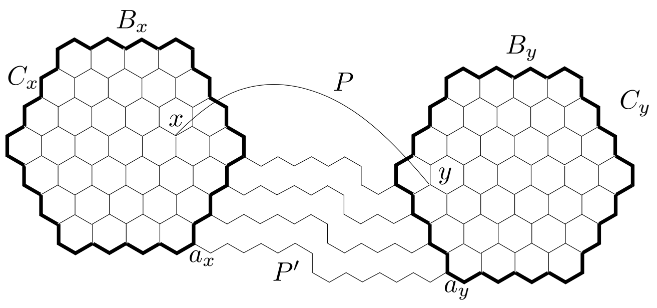


Figure 2.3: Illustration for Lemma 2.3.2. The edges of $C_x$ and $C_y$ are shown in bold.

used also that $R_i$ is a subdivision of a 3-connected graph whose embeddings are equivalent.

Now it is easy to see that we can combine $\theta$ and $\phi$ in such a way that we embed $G - X' - (Q_{i-1} - U)$ according to $\phi$ and, similarly, $Q_{i-1}$ according to $\theta$, and then "connect" them by identifying $\phi(u)$ and $\theta(u)$ for all $u \in U$. This gives the desired embedding of $G - X'$. Finally, we have to observe that $X' \in \text{ApexSets}(G, k)$ implies $X \in \text{ApexSets}(G, k)$, since $X' \subseteq X$ and $|X| \leq k$. □

This lemma has a trivial but crucial consequence: $X \in \text{ApexSets}(G, k)$ if and only if $X \in \text{ApexSets}(G - T(R_0), k)$, so deleting $T(R_0)$ reduces our problem to an equivalent instance. Let us denote this deletion as *Reduction A*.

Note that the closedness of a zone $Z$ can be decided by a simple breadth first search, which can also produce the graph $T(Z)$. Planarity can also be tested in linear time [78]. Therefore we can test whether a zone is flat, and if so, we can apply Reduction A on it in linear time.

Later we will see that unless there are some easily recognizable vertices in our graph which must be included in every solution, a flat zone can always be found (Lemma 2.3.6). This yields an easy way to handle graphs with large treewidth: compressing our graph by repeatedly applying Reduction A we can reduce the problem to an instance with bounded treewidth.

**Reduction B: Well-attached vertices.** A subgraph of $R$ is a *block* if it is topologically isomorphic to $H_{k+3}$. A vertex of a given block is called *inner vertex* if it is not on the outer circle of the block. (We define the outer circle of the block using the "standard" planar embedding of $H_{k+3}$. Instead of a formal definition, we refer to the illustration in Figure 2.3.)

**Lemma 2.3.2.** *Let $X \in \text{ApexSets}(G, k)$. Let $x$ and $y$ be inner vertices of the disjoint blocks $B_x$ and $B_y$, respectively. If $P$ is an $x - y$ path that (except for its endpoints) does not contain any vertex from $B_x$ or $B_y$, then $X$ must contain a vertex from $B_x$, $B_y$ or $P$.*

*Proof.* See Figure 2.3 for the illustration of this proof. Let $C_x$ and $C_y$ denote the outer circle of $B_x$ and $B_y$, respectively. Let us notice that since $B_x$ and $B_y$ are disjoint blocks, there exist at least $k+3$ vertex disjoint paths between their outer circles, which—apart from their endpoints—do not contain vertices from $B_x$ and $B_y$. Moreover, it is easy to see that these paths can be defined in a way such that their endpoints that lie on $C_x$ are on the border of different cells of $B_x$. To see this, note that the number of cells which lie on the border of a given block is $6k + 12$. At least three of these paths must be in $G - X$ also. Since $x$ can lie only on the border of at most two cells having common vertices with $C_x$, we get that there is a path $P'$ in $G - X$ whose endpoints are $a_x$ and $a_y$ (lying on $C_x$ and $C_y$, resp.), and there exist no cell of $B_x$ whose border contains both $a_x$ and $x$.

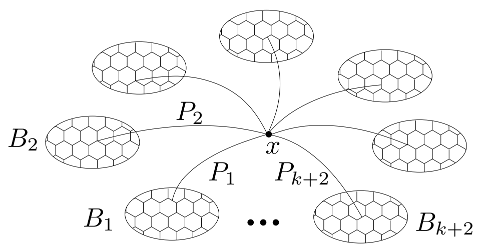


Figure 2.4: A well-attached vertex.

Let us suppose that $B_x \cup B_y \cup P$ is a subgraph of $G - X$. Since all embeddings of a 3-connected planar graph are equivalent, we know that if we restrict an arbitrary planar embedding of $G - X$ to $B_x$, then all faces having $x$ on their border correspond to a cell in $B_x$. Since $x$ and $y$ are connected through $P$ and $V(P) \cap V(B_x) = \{x\}$, we get that $y$ must be embedded in a region $F$ corresponding to a cell $C_F$ of $B_x$. But this implies that $B_y$ must entirely be embedded also in $F$.

Since $V(P' - a_x - a_y) \cap V(B_x) = \emptyset$ and $P'$ connects $a_x \in V(B_x)$ and $a_y \in V(B_y)$ we have that $a_y$ must lie on the border of $F$. But then $C_F$ is a cell of $B_x$ containing both $a_x$ and $x$ on its border, which yields the contradiction. □

Using this lemma we can identify certain vertices that have to be deleted. Let $x$ be a *well-attached* vertex in $G$ if there exist paths $P_1, P_2, \dots, P_{k+2}$ and disjoint blocks $B_1, B_2, \dots, B_{k+2}$ such that $P_i$ connects $x$ with an inner vertex of $B_i$ ($1 \le i \le k+2$), the inner vertices of $P_i$ are not in $R$, and if $i \ne j$ then the only common vertex of $P_i$ and $P_j$ is $x$.

**Lemma 2.3.3.** *Let $X \in \mathrm{ApexSets}(G, k)$. If $x$ is well-attached, then $x \in X$.*

*Proof.* If $x \notin X$, then after deleting $X$ from $G$ (which means deleting at most $k$ vertices) there would exist indices $i \ne j$ such that no vertex from $P_i$, $P_j$, $B_i$, and $B_j$ was deleted. But then the disjoint blocks $B_i$ and $B_j$ were connected by the path $P_i - x - P_j$, and by the previous lemma, this is a contradiction. □

We can decide whether a vertex $v$ is well-attached in time $f'(k)e$ using standard flow techniques, where $e = |E(G)|$. This can be done by simply testing for each possible set of $k+2$ disjoint blocks whether there exist the required disjoint paths that lead from $x$ to these blocks. Since the number of blocks in $R$ depends only on $k$, and we can find $p$ disjoint paths starting from a given vertex of a graph $G$ in time $O(p|E(G)|)$, we can observe that this can be done indeed in time $f'(k)e$.

**Finding flat zones.** Now we show that if there are no well-attached vertices in the graph $G$, then a flat zone exists in our grid.

**Lemma 2.3.4.** *Let $X \in \mathrm{ApexSets}(G, k)$, and let $G$ not include any well-attached vertices. If $K$ is a grid-component, then there cannot exist $(k+1)^2$ disjoint blocks such that $K$ is attached to an inner vertex of each block.*

*Proof.* Let us assume for contradiction that there exist $(k+1)^2$ such blocks. Since $|X| \le k$, at least $(k+1)^2 - k$ of these blocks do not contain any vertex of $X$. So let $x_1, x_2, \dots x_{(k+1)^2-k}$ be adjacent to $K$ and let $B_1, B_2, \dots, B_{(k+1)^2-k}$ be disjoint blocks of $G - X$ such that $x_i$ is an inner vertex of $B_i$.

Since $G - X$ is planar, it follows from Lemma 2.3.2 that a component of $K - X$ cannot be adjacent to different vertices from $\{x_i \mid 1 \le i \le (k+1)^2 - k\}$. So let $K_i$ be the connected component of $K - X$ that is attached to $x_i$ in $G - X$. $K$ is connected in G, hence for every $K_i$ there is a vertex of $T = K \cap X$ that is adjacent to it in $G$. Since there are no well-attached vertices in $G$, every vertex of $T$ can be adjacent to at most $k+1$ of these subgraphs. But then $|T| \geq ((k+1)^2 - k)/(k+1) > k$ which is a contradiction since $T \subseteq X$. □

Let us now fix the constant $d = (k+1)((k+1)^2 - 1)$.

**Lemma 2.3.5.** *Let $X \in \text{ApexSets}(G,k)$, let $G$ not include any well-attached vertices, and let $x$ be a vertex of the grid $R$. Then there cannot exist $B_1, B_2, \ldots, B_{d+1}$ disjoint blocks such that for all $i$ ($1 \leq i \leq d+1$) an inner vertex of $B_i$ and $x$ are both attached to some grid-component $K_i$.*

*Proof.* As a consequence of Lemma 2.3.4, each of the grid-components $K_i$ can be attached to at most $(k+1)^2 - 1$ disjoint blocks. But since $x$ is not a well-attached vertex, there can be only at most $k+1$ different grid-components among the grid-components $K_i$, $1 \leq i \leq d+1$. So the total number of disjoint blocks that are attached to $x$ through a grid-component is at most $(k+1)((k+1)^2 - 1) = d$. □

**Lemma 2.3.6.** *Let $X \in \text{ApexSets}(G,k)$, and let $G$ not include any well-attached vertices. Then there exists a flat zone $Z$ in $G$.*

*Proof.* Let $Z \in \mathcal{Z}$ be an open zone which has a vertex $w$ in its core that is attached to a vertex $v$ of another zone in $\mathcal{Z}$ ($v \notin V(Z)$) through a grid-component $K$. By the choice of the size of the zones and their cores, we have disjoint blocks $B_w$ and $B_v$ containing $w$ and $v$ respectively as inner points. We can also assume that $B_w$ is a subgraph of $Z$ which does not intersect the outer circle of $Z$.

By Lemma 2.3.2 we know that $B_w$, $B_v$ or $K$ contains a vertex from $X$. Let $\mathcal{Z}_1$ denote the set of zones in $\mathcal{Z}$ with an inner vertex in $X$, let $\mathcal{Z}_2$ denote the set of open zones in $\mathcal{Z}$ with a core vertex to which a grid-component, having a common vertex with $X$, is attached, and finally let $\mathcal{Z}_3$ be the set of the remaining open zones in $\mathcal{Z}$. Since $|X| \leq k$ and a grid-component can be attached to inner vertices of at most $(k+1)^2$ disjoint blocks by Lemma 2.3.4, we have that $|\mathcal{Z}_1| \leq k$ and $|\mathcal{Z}_2| \leq k(k+1)^2$.

Let us count the number of zones in $\mathcal{Z}_3$. To each zone $Z$ in $\mathcal{Z}_3$ we assign a vertex $u(Z)$ of the grid not in $Z$, which is connected to the core of $Z$ by a grid-component. First, let us bound the number of zones $Z$ in $\mathcal{Z}_3$ for which $u(Z) \in X$. Lemma 2.3.5 implies that any $v \in X$ can be connected this way to at most $d$ zones, so we can have only at most $kd$ such zones.

Now let $U = \{v \mid v = u(Z), Z \in \mathcal{Z}_3\}$. Let $a$ and $b$ be different members of $U$, and let $a$ be connected through the grid-component $K_a$ with the core vertex $z_a$ of $Z_a \in \mathcal{Z}_3$. Let $B_a$ denote a block which only contains vertices that are inner vertices of $Z_a$, and contains $z_a$ as inner vertex. Such a block can be given due to the size of a zone and its core. Let us define $K_b$, $z_b$, $Z_b$, and $B_b$ similarly. Note that $V(B_a) \cap X = V(B_b) \cap X = \emptyset$ by $Z_a, Z_b \notin \mathcal{Z}_1$.

Now let us assume that $a$ and $b$ are in the same component of $R - X$. Let $P$ be a path connecting them in $R - X$. If $P$ has common vertices with $B_a$ (or $B_b$) then we modify $P$ the following way. If the first and last vertices reached by $P$ in $Z_a$ (or $Z_b$, resp.) are $w$ and $w'$, then we swap the $w - w'$ section of $P$ using the outer circle of $Z_a$ (or $Z_b$, resp.). This way we can fix a path in $R - X$ that connects $a$ and $b$, and does not include any vertex from $B_a$ and $B_b$. But this path together with $K_a$ and $K_b$ would yield a path in $G - X$ that connects two inner vertices of $B_a$ and $B_b$, contradicting Lemma 2.3.2.

Therefore, each vertex of $U$ lies in a different component of $R - X$. But we can only delete at most $k$ vertices, and each vertex in a hexagonal grid has at most 3 neighbors, thus we can conclude that $|U| \leq 3k$. As for different zones $Z_1$ and $Z_2$ in $\mathcal{Z}$ we cannot have $u(Z_1) = u(Z_2)$ (which is also a consequence of Lemma 2.3.2) we have that $|\mathcal{Z}_3| \leq 3k$. So if we choose the

```
Phase I of algorithm Apex
    Input: $G = (V, E)$.
    Let $W = \emptyset$.
      1. Run algorithm GridStructure on $G$, $w(r, k)$, and $r$.
         If it returns a subgraph $H'_r$ topologically isomorphic to $H_r$ then go to
         Step 2. If it returns a tree decomposition $\mathcal{T}$ of $G$, then output $(G, W, \mathcal{T})$.
         Otherwise output "No solution".
      2. For all zones $Z$ do:
         If $Z$ is flat then $G := G - T(R_0)$, and go to Step 1.
      3. Let $U = \emptyset$. For all $x \in V$: if $x$ is well-attached then $U := U \cup \{x\}$.
         If $|U| = \emptyset$ or $|W| + |U| > k$ then output "No solution".
         Otherwise $W := W \cup U$, $G := G - U$ and go to Step 1.
```

Figure 2.5: Phase I of algorithm *Apex*.

number of zones in $\mathcal{Z}$ to be $z = 7k + k(k+1)^2 + kd + 1$ we have that there are at least $3k+1$ zones in $\mathcal{Z}$ which are not contained in $\mathcal{Z}_1 \cup \mathcal{Z}_2 \cup \mathcal{Z}_3$, indicating that they are closed. Since a vertex can be contained by at most 3 zones, $|X| \leq k$ implies that there exist a closed zone $Z^* \in \mathcal{Z}$, which does not contain any vertex from $X$, and all grid-components attached to $Z^*$ are also disjoint from $X$. This immediately implies that $T(Z^*)$ is a subgraph of $G - X$, and thus $T(Z^*)$ is planar. □

**Algorithm for Phase I.** The exact steps of Phase I of the algorithm *Apex* are shown in Figure 2.5. It starts with running algorithm *GridStructure* on the graph $G$ and integers $w(r,k)$ and $r$. If *GridStructure* returns a hexagonal grid as a topological minor, then the algorithm proceeds with the next step. If *GridStructure* returns a tree decomposition $\mathcal{T}$ of width $w(r,k)$, then Phase I returns the triple $(G, W, \mathcal{T})$. Otherwise $G$ does not have $H_r$ as minor and its treewidth is larger than $w(r,k)$, so by Lemma 2.2.5 we can conclude that $G \notin \text{Apex}(k)$.

In the next step the algorithm tries to find a flat zone $Z$. If such a zone is found, then the algorithm executes a deletion, whose correctness is implied by Lemma 2.3.1. Note that after altering the graph, the algorithm must find the hexagonal grid again and thus has to run *GridStructure* several times.

If no flat zone was found in Step 2, the algorithm removes well-attached vertices from the graph in Step 3. The vertices already removed this way are stored in $W$, and $U$ is the set of vertices to be removed in the actual step. By Lemma 2.3.3, if $X \in \text{ApexSets}(G,k)$ then $W \cup U \subseteq X$, so $|W| + |U| > k$ means that there is no solution. By Lemma 2.3.6, the case $U = \emptyset$ also implies $G \notin \text{Apex}(k)$. In these cases the algorithm stops with the output " No solution." Otherwise it proceeds with updating the variables $W$ and $G$, and continues with Step 1.

The output of the algorithm can be of two types: it either refuses the instance (outputting "No solution.") or it returns an instance for Phase II. For the above mentioned purposes the new instance is equivalent with the original problem instance in the following sense:

**Theorem 2.3.7.** *Let $(G', W, \mathcal{T})$ be the triple returned by Apex at the end of Phase I. Then for all $X \subseteq V(G)$ it is true that $X \in \text{ApexSets}(G,k)$ if and only if $W \subseteq X$ and $(X \setminus W) \in \text{ApexSets}(G', k - |W|)$.*

Now let us examine the running time of this phase. The first step can be done in time $f''(k)n$ according to [122, 17, 115] where $n = |V(G)|$. Since the algorithm only runs algorithm *GridStructure* again after reducing the number of the vertices in $G$, we have that *GridStructure* runs at most $n$ times. This takes $f''(k)n^2$ time. The second step requires only linear time (a breadth first search and a planarity test). Deciding whether a vertex is well-attached can be done in time $f'(k)e$ (where $e = |E(G)|$), so we need $f'(k)ne$ time to check every vertex at a given iteration in Step 3. Note that the third step is executed at most $k+1$ times, since at each iteration $|W|$ increases. Hence, this phase of algorithm *Apex* uses total time $f''(k)n^2 + f'(k)kne = f(k)n^2$, as the number of edges is $O(kn)$.

## 2.4 Phase II of Algorithm *Apex*

At the end of Phase I of algorithm *Apex* we either conclude that $G \notin \text{Apex}(k)$, or we have a triple $(G', W, \mathcal{T})$ for which Theorem 2.3.7 holds. Here $\mathcal{T}$ is a tree decomposition for $G'$ of width at most $w(r,k)$. This bound only depends on $r$ which is a function of $k$. From the choice of the constants $r, q, z$, and $d$ we can derive by a straightforward calculation that $\text{tw}(G') \le w(r,k) \le 100(k+2)^{7/2}$.

In order to solve our problem, we only have to find out if there is a set $Y \in \text{ApexSets}(G', k')$ where $k' = k - |W|$. For such a set, $Y \cup W$ would yield a solution for the original $k$-Apex problem.

A theorem by Courcelle states that every graph property defined by a formula in monadic second-order logic (MSO) can be evaluated in linear time if the input graph has bounded treewidth. Here we consider graphs as relational structures of vocabulary $\{V, E, I\}$, where $V$ and $E$ denote unary relations interpreted as the vertex set and the edge set of the graph, and $I$ is a binary relation interpreted as the incidence relation. For instance, a formula stating that $x$ and $y$ are neighboring vertices is the following: $\exists e : Ixe \wedge Iye$. We will denote by $U^G$ the universe of the graph $G$, i.e., $U^G = V(G) \cup E(G)$. Variables in monadic second-order logic can be element or set variables, and the containment relation between an element variable $x$ and a set variable $X$ is simply expressed by the formula $Xx$. For the complete description of MSO logic refer to [46], and for a survey on MSO logic on graphs see [39].

Following Grohe [69], we use a strengthened version of Courcelle's Theorem:

**Theorem 2.4.1.** ([56]) *Let $\varphi(x_1, \ldots, x_i, X_1, \ldots, X_j, y_1, \ldots, y_p, Y_1, \ldots, Y_q)$ denote an MSO-formula and let $w \ge 1$. Then there is a linear-time algorithm that, given a graph $G$ with $\text{tw}(G) \le w$ and $b_1, \ldots, b_p \in U^G, B_1, \ldots, B_q \subseteq U^G$, decides whether there exist $a_1, \ldots, a_i \in U^G, A_1, \ldots, A_j \subseteq U^G$ such that*

$$G \vDash \varphi(a_1, \ldots, a_i, A_1, \ldots, A_j, b_1, \ldots, b_p, B_1, \ldots, B_q),$$

*and, if this is the case, computes such elements $a_1, \ldots, a_i$ and sets $A_1, \ldots, A_j$.*

It is well-known that there is an MSO-formula $\varphi_{\text{planar}}$ that describes the planarity of graphs, i.e., for every graph $G$ the statement $G \vDash \varphi_{\text{planar}}$ holds if and only if $G$ is planar. The following simple claim shows that we can also create a formula describing the Apex($k$) graph class.

**Theorem 2.4.2.** *For every integer $k'$, there exists an MSO-formula* $\text{apex}(x_1, \ldots, x_{k'})$ *for which $G \vDash \text{apex}(v_1, \ldots, v_{k'})$ holds if and only if $\{v_1, \ldots, v_{k'}\} \in \text{ApexSets}(G, k')$.*

*Proof.* We will use the simple characterization of planar graphs by Kuratowski's Theorem: a graph is planar if and only if it does not contain any subgraph topologically isomorphic to $K_5$ or $K_{3,3}$. To formulate the existence of these subgraphs as an MSO-formula, we need some more simple formulas.

First, it is easy to see that the following formula expresses the property that $(X, Y)$ is a partition of the set $Z$:

$$\text{partition}(X, Y, Z) := \forall z : (Zz \to ((Xz \to \neg Yz) \wedge (\neg Xz \to Yz)))$$

Using this, we can express that the vertex set $Z$ contains a path connecting $a$ and $b$, by saying that every partition of $Z$ that separates $a$ and $b$ has to separate two neighboring vertices:

$$\begin{aligned}&\text{connected}(a, b, Z) := Za \wedge Zb \wedge \forall X \forall Y :\\ &\quad ((\text{partition}(X, Y, Z) \wedge Xa \wedge Yb) \to (\exists c \exists d \exists e : Xc \wedge Yd \wedge Ice \wedge Ide))\end{aligned}$$

The following two formulas express that two sets are disjoint, or their intersection is some given unique vertex.

$$\begin{aligned}&\text{disjoint}(X, Y) := \forall z : (Xz \to \neg Yz)\\ &\text{almost-disjoint}(X, Y, a) := \forall z : (Xz \to (\neg Yz \vee (z = a)))\end{aligned}$$

Now, we can state formulas expressing that a given subgraph has $K_5$ or $K_{3,3}$ as a topological minor. For brevity, we only give the formula which states that there is a subdivision of $K_5$ in the graph such that the vertices $v_1, v_2, \ldots, v_5$ correspond to the vertices of the $K_5$, and the vertex sets $P_{12}, P_{13}, \ldots, P_{45}$ contain the subdivisions of the corresponding edges of $K_5$.

$$\begin{aligned}&K_5\text{-top-minor } (v_1, v_2, \ldots, v_5, P_{12}, P_{13}, \ldots, P_{45}) :=\\ &\quad \text{connected}(v_1, v_2, P_{12}) \wedge \cdots \wedge \text{connected}(v_4, v_5, P_{45}) \wedge\\ &\quad \text{almost-disjoint}(P_{12}, P_{13}, v_1) \wedge \cdots \wedge \text{almost-disjoint}(P_{35}, P_{45}, v_5) \wedge\\ &\quad \text{disjoint}(P_{12}, P_{34}) \wedge \cdots \wedge \text{disjoint}(P_{23}, P_{45})\end{aligned}$$

The formula $K_{3,3}$-top-minor can be similarly created. Now, we are ready to give the apex formula having $k'$ free variables and fulfilling the property that $G \vDash \text{apex}(v_1, \ldots, v_{k'})$ holds if and only if $\{v_1, \ldots, v_{k'}\} \in \text{ApexSets}(G, k')$. The formula makes use of the fact that $G - \{v_1, v_2, \ldots, v_{k'}\}$ is planar if and only if every subdivision of $K_5$ or $K_{3,3}$ in $G$ involves at least one vertex from $\{v_1, v_2, \ldots, v_{k'}\}$.

$$\begin{aligned}&\text{apex}(v_1, v_2, \ldots, v_{k'}) :=\\ &\quad \forall x_1 \forall x_2 \ldots \forall x_5 \forall X_1 \forall X_2 \ldots \forall X_{10} : (K_5\text{-top-minor}(x_1, x_2, \ldots, x_5, X_1, X_2, \ldots, X_{10})\\ &\qquad \to \left( \bigvee_{\substack{1 \le i \le 5,\\ 1 \le j \le k'}} (x_i = v_j) \vee \bigvee_{\substack{1 \le i \le 10,\\ 1 \le j \le k'}} (X_i v_j) \right) \wedge\\ &\quad \forall x_1 \forall x_2 \ldots \forall x_6 \forall X_1 \forall X_2 \ldots \forall X_9 : (K_{3,3}\text{-top-minor}(x_1, x_2, \ldots, x_6, X_1, X_2, \ldots, X_9)\\ &\qquad \to \left( \bigvee_{\substack{1 \le i \le 6,\\ 1 \le j \le k'}} (x_i = v_j) \vee \bigvee_{\substack{1 \le i \le 9,\\ 1 \le j \le k'}} (X_i v_j) \right)\end{aligned}$$

□

Now let us apply Theorem 2.4.1. Let $\mathcal{C}$ be the algorithm which, given a graph $G$ of bounded treewidth, decides whether there exist $v_1, \ldots, v_{k'} \in U^G$ such that $G \vDash \text{apex}(v_1, \ldots, v_{k'})$ is true, and if possible, also produces such variables. By Theorem 2.4.2, running $\mathcal{C}$ on $G'$ either returns a set of vertices $U \in \mathrm{ApexSets}(G', k')$, or reports that this is not possible. Hence, we can finish algorithm *Apex* in the following way: if $\mathcal{C}$ returns $U$ then output($U \cup W$), otherwise output("No solution").

The running time of Phase II is $g(k)n$ for some function $g$.

*Remark.* Phase II of the algorithm can also be done by applying dynamic programming, using the tree decomposition $\mathcal{T}$ returned by *GridStructure*. This also yields a linear-time algorithm, with a double exponential dependence on $\mathrm{tw}(G')$ (and hence on $k$). Since the proof is quite technical and detailed, we omit it.

Finally, we can summarize the main theorem of this chapter as follows.

**Theorem 2.4.3.** *Algorithm Apex presented here solves the $k$-Apex problem in time $f(k)n^2$ for some function $f$, where $n$ is the number of vertices in the input graph.*

CHAPTER 3

# Recognizing almost isomorphic graphs

In this chapter, we investigate the parameterized complexity of the following problem, called the INDUCED SUBGRAPH ISOMORPHISM problem: given two graphs $G$ and $H$, decide whether we can delete some vertices of $G$ to obtain a graph isomorphic to $H$. We parameterize this problem by $k = |V(G)| - |V(H)|$, the number of vertices which we have to delete from $G$ to obtain a graph isomorphic to $H$. We propose newly developed FPT algorithms for this problem in the following three cases:

- $G$ is an arbitrary graph but $H$ is a tree, or
- both $G$ and $H$ are planar graphs and $H$ is 3-connected, or
- both $G$ and $H$ are interval graphs.

In the first and the third case, we also prove that the given problem is NP-hard (this was already known in the second case). We discuss the first two cases in this chapter, but we dedicate a new chapter to the case of interval graphs, due to the involved techniques we used to attack it.

Problems related to graph isomorphisms play a significant role in algorithmic graph theory. The INDUCED SUBGRAPH ISOMORPHISM problem is one of the basic problems of this area: given two graphs $H$ and $G$, find an induced subgraph of $G$ isomorphic to $H$, if this is possible. In this general form, INDUCED SUBGRAPH ISOMORPHISM is NP-hard, since it contains several well-known NP-hard problems, such as CLIQUE, INDEPENDENT SET and INDUCED PATH.

As INDUCED SUBGRAPH ISOMORPHISM has a wide range of important applications, polynomial-time algorithms have been given for numerous special cases, such as the case when both input graphs are trees [111] or 2-connected outerplanar graphs [93]. However, INDUCED SUBGRAPH ISOMORPHISM remains NP-hard even if $H$ is a forest and $G$ is a tree, or if $H$ is a path and $G$ is a cubic planar graph [62].

Note that INDUCED SUBGRAPH ISOMORPHISM is solvable in time $O(|V(G)|^{|V(H)|}|V(H)|^2)$ on input graphs $H$ and $G$ by trying every possible subgraph of $G$, or more precisely, by checking for every possible injective mapping from $V(H)$ to $V(G)$ whether it is an isomorphism. As $H$ is typically much smaller than $G$ in applications related to pattern matching, the usual parameterization of INDUCED SUBGRAPH ISOMORPHISM is to define the parameter $k$

to be $|V(H)|$. FPT algorithms with this parameterization are known if $G$ is planar [50], has bounded degree [25], or if $H$ is a log-bounded fragmentation graph and $G$ has bounded treewidth [74]. We note that this parameterization yields a W[1]-complete problem when $G$ can be arbitrary and $H \in \mathcal{H}$ for some graph class $\mathcal{H}$ of infinite cardinality [30].

In this thesis, we consider another parameterization of Induced Subgraph Isomorphism, where the parameter is the difference $|V(G)| - |V(H)|$. Considering the presence of extra vertices as some kind of error or noise, the problem of finding the original graph $H$ in the "dirty" graph $G$ containing errors is clearly meaningful. In other words, the task is to "clean" the graph $G$ containing errors in order to obtain $H$. For two graph classes $\mathcal{H}$ and $\mathcal{G}$ we define the Cleaning$(\mathcal{H}, \mathcal{G})$ problem: given a pair of graphs $(H, G)$ with $H \in \mathcal{H}$ and $G \in \mathcal{G}$, find a set of vertices $S$ in $G$ such that $G - S$ is isomorphic to $H$. The parameter associated with the input $(H, G)$ is $|V(G)| - |V(H)|$. Clearly, the size of the set $S$ to be found has to be equal to the parameter. For the case when $\mathcal{G}$ or $\mathcal{H}$ is the class of all graphs, we will use the notation Cleaning$(\mathcal{H}, -)$ or Cleaning$(-, \mathcal{G})$, respectively.

In the special case when the parameter is 0, the problem is equivalent to the Graph Isomorphism problem, so we cannot hope to give an FPT algorithm for the general problem Cleaning$(-, -)$. Thus, we consider special graph classes where the Graph Isomorphism problem is solvable in polynomial time. The most important graph classes on which Graph Isomorphism is polynomial-time solvable are planar graphs [78], interval graphs [96], permutation graphs [34], graphs having bounded treewidth [15], bounded genus [113, 55], or bounded degree [97]. We focus on the class of planar graphs and interval graphs, denoted by *Planar* and *Interval*, respectively. Since we were not able to solve the Cleaning(*Planar*, *Planar*) problem in general, we consider special graph classes of planar graphs such as trees and 3-connected planar graphs, denoted by *Tree* and *3-Connected-Planar*, respectively.

We give FPT algorithms for the problems Cleaning(*Tree*,−), Cleaning(*3-Connected-Planar, Planar*) and Cleaning(*Interval, Interval*). Note that these problems differ from the Feedback Vertex Set, the $k$-Apex problems, and the Interval Deletion problems, where the task is to delete a minimum number of vertices from the input graph to get an *arbitrary* acyclic, planar, or interval graph, respectively.

Without parameterization, Cleaning(*Tree*,−) is NP-hard because it contains Induced Path. We show NP-hardness for Cleaning(*3-Connected-Planar, 3-Connected-Planar*) and Cleaning(*Interval, Interval*) too. A polynomial-time algorithm is known for Cleaning(*Tree, Tree*) [111], and an FPT algorithm is known for Cleaning(*Grid*,−) where *Grid* is the class of rectangular grids [40].

In Table 3.1 we summarize the complexity of the Cleaning$(\mathcal{H}, \mathcal{G})$ problem for different graph classes $\mathcal{H}$ and $\mathcal{G}$. Table 3.1 contains results for three versions of the problem: without parameterization, with the standard parameter $|V(H)|$, and finally with the parameter $|V(G)| - |V(H)|$ yielding the parameterization discussed in this chapter.

In Section 3.1, we show that Cleaning(*3-Connected-Planar, 3-Connected-Planar*) is NP-complete, and we also present an FPT algorithm for Cleaning(*3-Connected-Planar, Planar*). We discuss the Cleaning(*Tree*,−) problem in Section 3.2. Our results for the Cleaning(*Interval, Interval*), including an NP-hardness proof and an FPT algorithm for the problem, are contained in Chapter 4. (Although this problem logically belongs to the issue of Chapter 3, we moved its discussion to Chapter 4 because of its length and the differences of the techniques applied.)

The results of this chapter were published in [106], and the results of Chapter 4 can be found in [108].

| | Parameter | | |
|---|---|---|---|
| Graph classes $(\mathcal{H}, \mathcal{G})$ | None | $\lvert V(H)\rvert$ | $\lvert V(G)\rvert - \lvert V(H)\rvert$ |
| (*Tree*, *Tree*) | P [111] | FPT (trivial) | FPT (trivial) |
| (*Tree*,−) | NP-complete [62] | W[1]-complete [29] | FPT* |
| (*3-Connected-Planar*, *Planar*) | NP-complete* | FPT [50] | FPT* |
| (−, *Planar*) | NP-complete [62] | FPT [50] | Open |
| (*Interval*, *Interval*) | NP-complete* | W[1]-hard* | FPT* |
| (*Grid*,−) | NP-complete [62] | W[1]-complete [30] | FPT [40] |

Table 3.1: Summary of some known results for the Cleaning$(\mathcal{H}, \mathcal{G})$ problem. The new results obtained by us are marked with an asterisk.

## 3.1 3-connected planar graphs

In this section, we present an algorithm for Cleaning(*3-Connected-Planar, Planar*). Since 3-connected planar graphs can be considered as "rigid" graphs in the sense that they cannot be embedded in the plane in essentially different ways, this problem seems to be easy. However, Theorem 3.1.1 shows that it is NP-hard.

**Theorem 3.1.1.** Cleaning(*3-connected-Planar, 3-Connected-Planar*) *is* NP*-hard.*

*Proof.* We give a reduction from the NP-complete Planar 3-Colorability problem [62]. Let $F$ be the planar input graph given. W.l.o.g. we assume that $F$ is connected. We construct 3-connected planar graphs $H$ and $G$ such that Cleaning(*3-Connected-Planar, 3-Connected-Planar*) with input $(H, G)$ is solvable if and only if $F$ is 3-colorable.

The high-level idea of the reduction is the following. For each vertex in $F$, we construct a wheel-like gadget in $G$ and a similar gadget in $H$ with the property that the gadget of $H$ can be obtained as the induced subgraph of the gadget in $G$ in three different ways (as illustrated by Figure 3.1). These three different ways correspond to the three possible colorings of the given vertex of $F$. The hard task is to force that $H$ can only be an induced subgraph of $G$ if the coloring indicated by the deleted vertices of the gadgets in $G$ is a proper coloring of $F$. This will be ensured using connection gadgets for each edge of $F$.

The gadgets we construct are shown in Figure 3.1. For every $x \in V(F)$ we set an integer $9|V(F)| \leq p(x) \leq 10|V(F)|$ such that $p(x) \neq p(y)$ for any $x \neq y \in V(F)$. For every vertex $x \in V(F)$ we build a *node-gadget* $N_x$ in $G$ as follows. We introduce a central vertex $a^x$, together with a cycle consisting of the vertices $b_0^x, b_1^x, \ldots, b_{6p(x)-1}^x$ with each $b_i^x$ being connected to $a^x$, and finally the vertices $c_0^x, \ldots, c_{3p(x)-1}^x$ with each $c_i^x$ being connected to three consecutive vertices from the cycle $b_0^x b_1^x \ldots b_{6p(x)-1}^x$, as illustrated in Figure 3.1. Formally, the edge set of the node-gadget $N_x$ is $\{a^x b_j^x, b_{j-1}^x b_j^x \mid j \in [6p(x)]\} \cup \{c_j^x b_{2j}^x, c_j^x b_{2j+1}^x, c_j^x b_{2j+2}^x \mid 0 \leq j < 3p(x)\}$ where $b_{6p(x)}^x = b_0^x$. The node-gadget $N_x$ can be considered as a plane graph, supposing that the vertices $b_0^x, b_1^x, \ldots, b_{6p(x)-1}^x$ (and thus $c_0^x, c_1^x, \ldots, c_{3p(x)-1}^x$) are embedded in a clockwise order around $a^x$. We define the $j$-th *block* $B_j^x$ of $N_x$ to be $(c_{3j}^x, c_{3j+1}^x, c_{3j+2}^x)$, for every $0 \leq j < p(x)$. The *type* of $c_j^x$ can be 0, 1, or 2, according to the value of $j$ modulo 3. We set $C_x = \{c_j^x \mid 0 \leq j < 3p(x)\}$.

Let us fix an arbitrary ordering of the vertices of $F$. For each $x < y$ with $xy \in E(F)$ we build a *connection* $E_{xy}$ in $G$ that uses 9 consecutive blocks from each of $N_x$ and $N_y$, say $B_i^x, \ldots, B_{i+8}^x$ and $B_j^y, \ldots, B_{j+8}^y$. These blocks are the *base blocks* for $E_{xy}$, and we also define $q(x, y) = (i, j)$. Note that since $p(x) \geq 9|V(F)| > 9d_F(x)$, we can define connections such that no two connections share a common base block. To build $E_{xy}$ with $q(x, y) = (i, j)$,

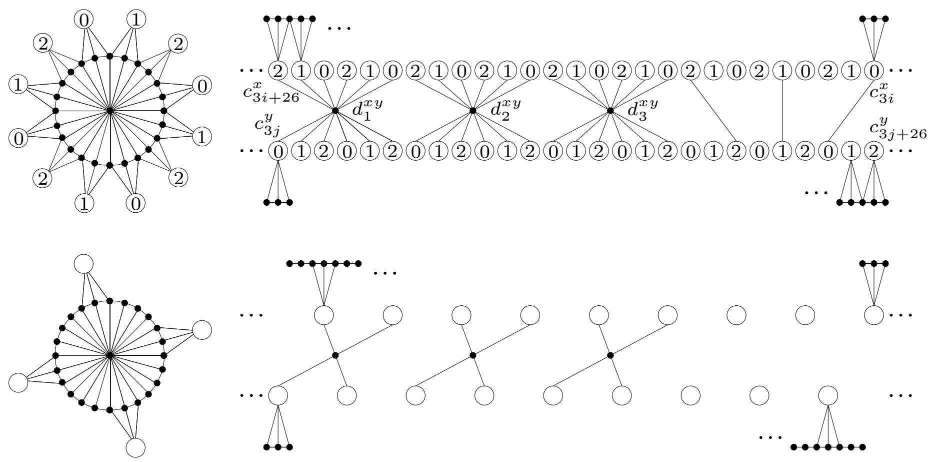


Figure 3.1: A node-gadget and a connection in $G$, and the corresponding subgraphs of $H$ used in the proof of Theorem 3.1.1.

we introduce new vertices $d_1^{xy}, d_2^{xy}, d_3^{xy}$ and edges $\{c^x_{3i+26-6m+\ell}d^{xy}_m, c^y_{3j+6m-\ell}d^{xy}_m \mid m \in [3], \ell \in [6]\} \cup \{c^x_{3i}c^y_{3j+24}, c^x_{3i+4}c^y_{3j+22}, c^x_{3i+8}c^y_{3j+20}\}$ (see Figure 3.1). By choosing the base blocks for each connection in a way that the order of the connections around a node-gadget is the same as the order of the corresponding edges around the corresponding vertex for some fixed planar embedding of $F$, we can give a planar embedding of $G$. Moreover, it is easy to see that $G$ is also 3-connected.

To construct $H$, we make a disjoint copy $\bar{G}$ of $G$, and delete some edges and vertices from it as follows. For the copy of $c^x_j$ ($a^x$, $C_x$, etc.) we write $\bar{c}^x_j$ ($\bar{a}^x$, $\bar{C}^x$, etc. respectively). To get $H$, we delete from $\bar{G}$ the three edges connecting vertices of $\bar{C}_x$ and $\bar{C}_y$ for every $x < y$ and $xy \in E(F)$, and also the vertices $\bar{c}^x_{3j+1}$ and $\bar{c}^x_{3j+2}$ for every $x \in V(F), 0 \leq j < p(x)$. Clearly, $H$ is planar, and observe that it remains 3-connected.

Now, we prove that if Cleaning(*3-Connected-Planar, 3-Connected-Planar*) has a solution $S$ for the input $(H, G)$, then $F$ is 3-colorable. Let $\varphi$ be an isomorphism from $H$ to $G-S$. First, observe that since $p(x) \neq p(y)$ if $x \neq y$, and the integers $\{p(x) \mid x \in V(F)\}$ are large enough, $\varphi$ must map $\bar{a}^x$ to $a^x$ because of its degree. For each $x \in V(F)$, the vertices in $C_x \setminus S$ must have the same type, so let the color of $x$ be this type. If $xy \in E(F)$, then the color of $x$ and $y$ must differ, otherwise one of the edges $c^x_{3i}c^y_{3j+24}, c^x_{3i+4}c^y_{3j+22}, c^x_{3i+8}c^y_{3j+20}$ would be in $G-S$ where $q(x,y) = (i,j)$, as for every type $t$, one of these edges connects two vertices of type $t$. Thus the coloring is proper.

For the other direction, let $t : V(F) \to \{0,1,2\}$ be a coloring of $F$. For each $x \in V(F)$, let $S$ contain those vertices in $C_x$ whose type is not $t(x)$. Let $\varphi$ map $\bar{a}^x$ and $\bar{d}^{xy}_m$ (for every meaningful $x, y, m$) to $a^x$ and $d^{xy}_m$, respectively, and let $\varphi$ map $\bar{c}^x_j$ to $c^x_{j+t(x)}$. By adjusting $\varphi$ on the vertices $\bar{b}^x_i$ in the natural way, we can prove that $\varphi$ is an isomorphism. It is clear that the restriction of $\varphi$ on $\bar{N}_x$ is an isomorphism. Note that the only vertex of $B^x_j$ present in $G-S$ is $c^x_{3j+t(x)} = \varphi(\bar{c}^x_{3j})$, so independently from $t(x)$ and $t(y)$, the neighborhood of $\bar{d}^{xy}_m$ is also preserved. We only have to check that the edges connecting $C_x$ and $C_y$ are not present in $G-S$. This is implied by the properness of the coloring, as all such edges connect vertices of the same type, but for $xy \in E(F)$ the types of the vertices in $C_x \setminus S$ and $C_y \setminus S$ differ. □

We present an FPT algorithm for Cleaning(*3-Connected-Planar, Planar*) where the parameter is $k = |V(G)| - |V(H)|$ for input $(H, G)$. We assume $n = |V(H)| > k+2$ and $n \geq 4$

as otherwise we can solve the problem by brute force. We also assume that $H$ and $G$ are simple graphs.

Let $S$ be a solution. First observe that if $C$ is a set of at most 2 vertices such that $G-C$ is not connected, then there is a component $K$ of $G-C$ such that the 3-connected graph $G-S$ is contained in $G[V(K)\cup C]$. Clearly, $|V(K)|\geq n-2$, so $K$ is unique by $n>k+2$. Since such a separating set $C$ can be found in linear time [77], $K$ can also be found in linear time. If no component of $G-C$ has size at least $n-2$, the algorithm outputs 'No', otherwise it proceeds with $G[V(K)\cup C]$ as input.

So we can assume that $G$ is 3-connected. First, the algorithm determines a planar embedding of $H$ and $G$. Every planar embedding determines a circular order of the edges incident to a given vertex. Two embeddings are equivalent, if these orderings are the same for each vertex in both of the embeddings. It is well-known that a 3-connected planar graph has exactly two planar embeddings, and these are reflections of each other (see e.g. [41]). Let us fix an arbitrary embedding $\theta$ of $H$. By the 3-connectivity of $G$, one of the two possible embeddings of $G$ yields an embedding of $G-S$ that is equivalent to $\theta$. The algorithm checks both possibilities. From now on, we regard $H$ and $G$ as plane graphs, and we are looking for an isomorphism $\varphi$ from $H$ into $G-S$ which preserves the embedding.

Before going into the details, we need two definitions concerning plane graphs. For a subgraph $H$ of a plane graph $G$, an edge $e\in E(H)$ is called an *outer edge* of $(H,G)$ if $G$ has a face $F_e$ incident to $e$ which is not in $H$. In this case, $F_e$ is an *outer face* of $e$ w.r.t. $(H,G)$. The *border* of $H$ in $G$ is the subgraph formed by the outer edges of $(H,G)$.

In a general step of the algorithm, we grow a partial mapping, which is a restriction of $\varphi$. We assume that $\varphi$ is already determined on a connected subgraph $D$ of $H$ having at least one edge. The definition of $D$ implies $\varphi(V(D))\cap S=\emptyset$, so if at some point the algorithm would have to delete vertices from $\varphi(D)$, it outputs 'No'.

The algorithm grows the subgraph $D$ on which $\varphi$ is determined step by step. At each step, it chooses an outer edge $e$ of $(D,H)$, and either deletes some vertices of $G-\varphi(D)$ that must be in $S$, or adds to $D$ an outer face $F$ of $e$ w.r.t. $(D,H)$. The algorithm chooses $e$ and $F$ in a way such that after the first step the following property will always hold:

> *Invariant 1*: the outer edges of $(D,H)$ form a cycle.

We refer to this as choosing a suitable face. Formally, a face $F$ is *suitable* for $(D,H)$ if it is an outer face w.r.t. $(D,H)$ and Invariant 1 holds after adding $F$ to $D$. Lemma 3.1.2 argues that a suitable face can always be found. We will see that the algorithm can only add a face $F$ to $D$ if $\varphi(F)$ is a face of $G$ as well (that is, the interior of $\varphi(F)$ does not contain vertices from $S$). Hence, this method ensures that all vertices of $\varphi(V(H-D))$ and $S$ are embedded on the same side of the border of $\varphi(D)$, allowing us to assume the following:

> *Invariant 2*: the vertices of $V(G)\setminus\varphi(V(D))$ are embedded in the unique unbounded region determined by the border of $\varphi(D)$ in $G$.

The most important consequence of Invariant 2 is that $\varphi$ yields a bijection between the outer edges of $(D,H)$ and the outer edges of $(\varphi(D),G)$.

**Lemma 3.1.2.** *If $D$ is a subgraph of a 3-connected graph $H$ such that $|V(D)|<|V(H)|$ and the border of $D$ in $H$ is a cycle $C$, then there exists a suitable face for $(D,H)$.*

*Proof.* By $|V(D)|<|V(H)|$, each edge of $C$ has an outer face w.r.t. $(D,H)$. The planarity of $H$ implies that if $a,b,c,d$ are four vertices appearing in this order on $C$, then there cannot exist two outer faces $F_1,F_2$ of $(D,H)$ such that $F_1$ contains $a$ and $c$, and $F_2$ contains $b$ and $d$. Given an outer face $F$ of $(D,H)$, let the *gap* of $F$ be the maximum length of a subpath of $C$ whose endpoints are in $V(F)$ but has no internal vertices in $V(F)$.

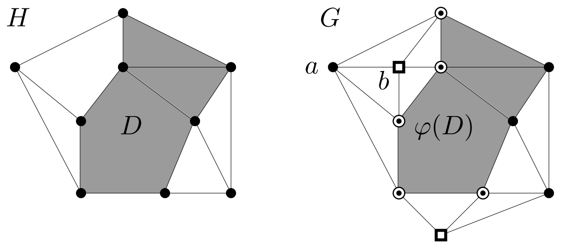


Figure 3.2: Common neighbors test. Vertices of $M$ are indicated by double circles, vertices of $S$ by squares. By Lemma 3.1.3, we obtain that $b \in S$ but $a \notin S$.

Now, consider an outer face $F^*$ of $(D, H)$ that has minimum gap. If the gap of $F^*$ is at least 2, then there is a subpath $Q$ of $C$ having at least two edges such that $V(F^*) \cap V(C)$ contains exactly the endpoints of $Q$. Consider any outer face $F_Q$ of $(D, H)$ that is incident to an edge of $Q$. By the observation of the previous paragraph, such a face cannot be incident to a vertex of $C$ that is not in $Q$. Thus, $F_Q$ must have smaller gap than $F^*$, which contradicts to the minimality of $F^*$. Therefore, $F^*$ must have gap 1. Hence, the vertices of $V(F^*) \cap V(C)$ are consecutive vertices of $C$, implying that $F^*$ is suitable. □

To find an initial partial mapping, we try to find a pair of edges $ab$ and $a'b'$ in $H$ and $G$, respectively, such that $\varphi(a) = a'$ and $\varphi(b) = b'$. To do that, the algorithm fixes an arbitrary edge $ab$ in $H$ and guesses $\varphi(a)$ and $\varphi(b)$. This yields $2|E(G)|$ possibilities. After this, the algorithm applies one of the following steps.

**3-connectivity test.** Although in the beginning $G$ is assumed to be 3-connected, the algorithm may delete vertices from $G$ throughout its running, and thus it can happen that $G$ ceases to be 3-connected. This can be handled as described above, by finding a separating set $C$ of size at most 2, and determining the component $K$ of $G - C$ with at least $|V(H)| - 2$ vertices. If no such component exists, or if it does not include $\varphi(D)$, then the algorithm outputs 'No', otherwise it deletes $V(G - C - K)$.

**Common neighbors test.** Let $M = \{\varphi(v) \mid v \in V(D), d_H(v) < d_G(\varphi(v))\}$. First, note that every vertex in $M$ must have a neighbor in $S$, thus if $|M| > 2k$, then some vertex in $S$ is adjacent to at least three vertices in $M$. By Invariant 2, the vertices of $S \subseteq V(G) \setminus \varphi(V(D))$ are embedded in the unbounded region determined by the border of $\varphi(D)$ in $G$, the vertices of $M$ lie on this border. The algorithm checks every vertex $q \in V(G) \setminus \varphi(V(D))$ having at least three neighbors on the border of $\varphi(D)$, and determines whether $q \in S$, using Lemma 3.1.3 below. If no such vertex of $S$ can be found in spite of $|M| > 2k$, then the algorithm outputs 'No'. Figure 3.2 shows an example.

**Lemma 3.1.3.** *Let $q \in V(G) \setminus \varphi(V(D))$ be adjacent to different vertices $x, y$ and $z$ on the border of $\varphi(D)$ in $G$. Then $q \in S$ if and only if there is no vertex $p \in V(H) \setminus V(D)$ which is a common neighbor of $\varphi^{-1}(x), \varphi^{-1}(y)$ and $\varphi^{-1}(z)$.*

*Proof.* For contradiction, let us assume $q \in S$ and suppose that a vertex $p$ exist as described. As Invariant 1 holds for $(D, H)$ and $\varphi$ preserves the embedding, the outer edges of $(\varphi(D), G)$ and the edges $\varphi(p)x, \varphi(p)y$ and $\varphi(p)z$ cut the plane into four regions, and the only region among these containing all three of $x, y$ and $z$ is the bounded region determined by the outer edges of $(\varphi(D), G)$. But as no vertex in $S$ can be embedded in this region by Invariant 2, $q$ cannot be adjacent to all of $x, y$ and $z$, a contradiction. In the case where there is no vertex in $V(H) \setminus V(D)$ adjacent to $\varphi^{-1}(x), \varphi^{-1}(y)$ and $\varphi^{-1}(z)$, then $q \in S$ is trivial. □

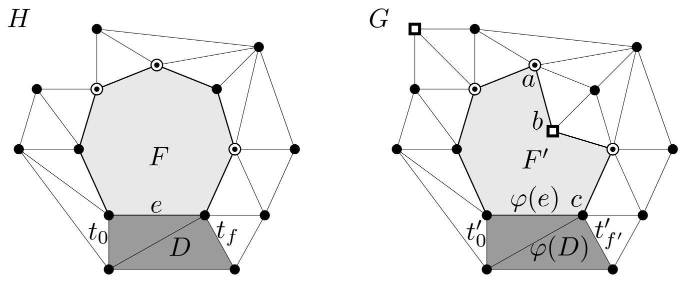


Figure 3.3: Examining an outer face. Vertices of $\{t_j \mid j \in R\}$ and $\{t'_j \mid j \in R\}$ are indicated by double circles, vertices of $S$ by squares. By Lemma 3.1.4, we obtain that $\{a, b, c\} \cap S \neq \emptyset$. Note that in general $F$ might have more than one common edge with $D$.

**Examining an outer face.** In this step, the algorithm takes an outer edge $e = xy$ of $(D, H)$ with a suitable outer face $F$ in $H$, and the corresponding outer face $F'$ of $\varphi(e)$ w.r.t. $(\varphi(D), G)$. Lemma 3.1.2 shows that a suitable face can always be found. If the algorithm finds that $V(F') \cap S = \emptyset$ must hold because of a sufficient condition given in Lemma 3.1.4 below, then it extends $\varphi$ by adding $F$ to $D$. Otherwise, $V(F')$ may contain vertices in $S$, so the algorithm branches into a bounded number of directions.

In the branch assuming $V(F') \cap S = \emptyset$, the extension of $\varphi$ is performed. In the branches when $V(F') \cap S \neq \emptyset$ is assumed, the algorithm tries to find and delete the first vertex $s$ on the border of $F'$ in $S$, and branches according to the choice of $s$. Lemma 3.1.4 bounds the possibilities to choose $s$.

Intuitively, if there is a vertex of $S$ on the border of $F'$, then its deletion decreases the degree of at least two other vertices in $V(F')$. Also, the 3-connectedness of $G-S$ implies that deleting a vertex of $S$ not in $V(F')$ can decrease the degree of at most two vertices on the border of $F'$. Lemma 3.1.4 states the consequences of these observations in a precise manner. See Figure 3.3 for an illustration.

**Lemma 3.1.4.** *Let $e = t_0t_f$ be an outer edge of $(D, H)$ and $F$ its outer face w.r.t. $(D, H)$ such that its vertices in clockwise ordering are $t_0, t_1, \dots, t_f$. Similarly, let $F'$ be the outer face of $\varphi(e)$ w.r.t. $(\varphi(D), G)$, where the vertices of $F'$ are $t'_0 = \varphi(t_0), t'_1, \dots, t'_{f'-1}$ in clockwise ordering and $t'_{f'} = \varphi(t_f)$. Let also $R = \{j \mid 0 \leq j \leq \min(f, f'), d_H(t_j) \neq d_G(t'_j)\}$ and let the indices in $R$ be $r_1 < \cdots < r_{|R|}$.*

*(1) If $|R| \leq 1$ and $f = f'$, then $V(F') \cap S = \emptyset$ and $\varphi(t_i) = t'_i$ for every $i \in [f]$.*

*(2) If $V(F') \cap S \neq \emptyset$ and $t'_{i^*}$ is the first vertex on the border of $F'$ that is in $S$, then $i^* - 1 \in \{r_j \mid j \in [\min(|R|, 2k+1)]\}$.*

*Proof.* Let $e_0 = t_0t_f$ and let $e_i = t_it_{i-1}$ for every $i \in [f]$, so $e_i$ is followed by $e_{i-1}$ in the clockwise circular order of the edges incident to $t_i$ for every $i \in [f]$. Similarly, we define $e'_i = t'_it'_{i-1}$ for every $i \in [f']$. Supposing $V(F') \cap S = \emptyset$, we prove by induction that $\varphi(t_i) = t'_i$ for every $i \in \{0, 1, \dots, f\}$. This is clear for $i = 0$, so assume that it holds for each index smaller than $i$. Since $\varphi$ is an isomorphism, we have $\varphi(e_{i-1}) = e'_{i-1}$. Now, $e'_i \in E(G - S)$ implies $\varphi(e_i) = e'_i$ as well, since $\varphi$ preserves the embedding. This proves the claim.

Now, if $V(F') \cap S \neq \emptyset$ and $t'_{i^*}$ is the first vertex on the border of $F'$ that is in $S$, then the vertices $t'_0, \dots, t'_{i^*-1}$ are not in $S$, so by applying the above argument we get $\varphi(t_\ell) = t'_\ell$ for all $\ell < i^*$. But $t'_{i^*-1}$ has a neighbor in $S$, hence $d_G(t'_{i^*-1}) > d_{G-S}(t'_{i^*-1}) = d_{G-S}(\varphi(t_{i^*-1})) = d_H(t_{i^*-1})$. This implies $i^*-1 \in R$. Letting $j^*$ be the last vertex on the border of $F'$ that is in $S$,

```
GrowSolution$(H, G, D, \varphi)$

  1. If $k = |V(G)| - |V(H)| < 0$ then output('No').
  2. Perform the 3-connectivity test.
  3. If $D$ equals $H$ then output('Yes').
  4. Perform the common neighbors test.
  5. Examine an outer face. If for the chosen pair $(F, F')$ of faces $|V(F)| = |V(F')|$
     and $|R| \leq 1$, then extend $\varphi$ on $F$, and go to Step 3. Otherwise branch as follows:
       • for all $j \in [2k+1]$: let $i^* = r_j + 1$ and call GrowSolution$(H, G - t'_{i^*}, D, \varphi)$.
       • if $|V(F)| = |V(F')|$ then extend $\varphi$ on $F$ and call GrowSolution$(H, G, D, \varphi)$.
```

Figure 3.4: The algorithm *GrowSolution*.

and using $f = f'$ and the same argument as above, we get $j^* + 1 \in R$. Clearly $i^* - 1 < j^* + 1$, so $V(F') \cap S \neq \emptyset$ would imply $|R| \geq 2$. Hence, the conditions of (1) imply $V(F') \cap S = \emptyset$, proving also $\varphi(t_i) = t'_i$ for every $i \in [f]$.

To prove (2), suppose $V(F') \cap S \neq \emptyset$. As $i^* - 1 \in R$, $i^* - 1 = r_{\ell^*}$ for some $\ell^*$. We claim $\ell^* \leq 2k + 1$, which clearly implies $i^* - 1 \in \{r_j \mid j \in [\min(|R|, 2k+1)]\}$. To see the claim, first observe that the definition of $i^*$ implies $S \cap \{t_i \mid i < i^*\} = \emptyset$, so if $\ell < \ell^*$ then $\varphi(t_{r_\ell}) = t'_{r_\ell}$. But from the definition of $R$ we know $d_H(t_{r_\ell}) \neq d_G(t'_{r_\ell})$, so we get that $t'_{r_\ell}$ must be adjacent to a vertex $s \in S$. As $r_\ell < i^* - 1$, this vertex $s$ cannot be in the region of $G$ corresponding to the face $F$ of $H$. Note that in a 3-connected graph with at least four vertices no three vertices on the border of a single face can also lie on the border of another face, so no three vertices in $V(F')$ can be adjacent to the same $s \in S$. Therefore, each vertex of $s$ can be adjacent to at most two vertices from $\{t_{r_\ell} \mid \ell \in [\ell^* - 1]\}$, so we obtain $\ell^* - 1 \leq 2|S| = 2k$. □

Now let us describe the key mechanism of our algorithm. The essential work is done by a recursive algorithm that we call *GrowSolution*, described in Figure 3.4. The input of *GrowSolution* is a 4-tuple $(H, G, D, \varphi)$, where $H$ and $G$ are plane graphs, $H$ is 3-connected, $D$ is a subgraph of $H$ which is either an edge (in the first step) or the union of faces whose border in $H$ is a cycle (Invariant 1), and $\varphi$ is an embedding preserving isomorphism from $D$ to an induced subgraph of $G$, such that the vertices of $V(G) \setminus \varphi(V(D))$ are embedded in the unique unbounded region determined by the border of $\varphi(D)$ in $G$ (Invariant 2). The algorithm finds out whether there is an $S \subseteq V(G) \setminus \varphi(V(D))$ such that $\varphi$ can be extended to map $H$ to $G - S$ while remaining an isomorphism that preserves embedding. In each call, *GrowSolution* may stop or branch into a few directions. According to this, we will speak of *terminal* and *branching calls*. In each branch of a branching call, *GrowSolution* either deletes a vertex from $G$, or extends $\varphi$ by adding a new face to $D$. If at the end of a branch a vertex is deleted, then this is a *deletion branch*, otherwise it is an *extension branch*. (Actually, the algorithm may extend $\varphi$ also in the deletion branches before performing the deletion.) At the end of each branch, *GrowSolution* calls itself recursively with the modified input.

In a single call, the algorithm first checks whether $|V(G)| < |V(H)|$, and if so, then correctly outputs 'No'. Next, it handles the case when $G$ is not 3-connected. If $D$ equals $H$, then Step 3 outputs 'Yes'. Note that this step allows $V(G) \setminus \varphi(V(H)) = S \neq \emptyset$ as well. To proceed, the algorithm searches for common neighbors, as described above. Recall that the algorithm might output 'No' or delete vertices from $G$ at this step according to Lemma 3.1.3. If the algorithm deletes vertices $S' \subseteq S$ in Step 2 or Step 4, then this means that it calls *GrowSolution*$(H, G - S', D, \varphi)$. Now, if the algorithm does not stop or delete vertices, it examines an outer face. If for the chosen pair of faces $(F, F')$ the conditions of (1) in Lemma 3.1.4 are fulfilled, then we know $V(F') \cap S = \emptyset$, so the algorithm proceeds by extending $\varphi$ on $F$ according to the lemma. When *GrowSolution* performs this extension, it also adds $F$ to $D$, and checks whether $\varphi$ is still an isomorphism on $D$, and if not, outputs 'No'. This is correct by Lemma 3.1.4. This extension step is iterated until either a vertex is deleted or the algorithm stops in Step 3, 4 or 5, or the conditions of (1) in Lemma 3.1.4 do not hold.

In the last case, we do not know whether $V(F') \cap S$ is empty or not, so the algorithm branches into at most $2k + 2$ directions. First we assume $V(F') \cap S \neq \emptyset$, in this case statement (2) of Lemma 3.1.4 implies that $i^* \in \{r_j + 1 \mid j \in [\min(2k + 1, |R|)]\}$ where $t'_{i^*}$ is the first vertex on the border of $F'$ being in $S$. The algorithm branches on these at most $2k + 1$ possibilities to delete $t'_{i^*}$. The last branch is an extension branch corresponding to the case $V(F') \cap S = \emptyset$. Here, *GrowSolution* performs the extension of $\varphi$ on $F$ as described above. Note that this branch is only necessary if $|V(F)| = |V(F')|$.

Observe that Lemmas 3.1.3 and 3.1.4 directly imply the correctness of the algorithm. Although *GrowSolution* only answers the decision problem, it is straightforward to modify it in order to output the set $S$ and the mapping $\varphi$.

To analyze the running time of the algorithm, we assign a search tree $T(I)$ to a run of *GrowSolution* with a given input $I$. The nodes of this tree correspond to the calls of *GrowSolution*. The leaves represent the terminal calls and the internal nodes represent branching calls. The edge(s) leaving a node represent the branch(es) of the corresponding call of *GrowSolution*, so $e$ heads from $x$ to $y$ if $y$ is called in the branch represented by $e$ in the call corresponding to $x$. The parameter of a node with input $I = (H, G, D, \varphi)$ is $k_I = |V(G)| - |V(H)|$. The parameter clearly decreases in a deletion branch, which cannot happen more than $k+1$ times. However, in the extension branches this is not true, which seems to make it problematic to bound the size of the search tree. The following lemma shows that this problem does not arise, thanks to Step 4 of the algorithm.

**Lemma 3.1.5.** *The size of $T(I)$ is bounded by $f(k) = 2(2k + 2)^{13k^2-1}$ where $k = k_I$.*

*Proof.* Let $E^*$ denote the edges in $T(I)$ that correspond to extension branches. The value of the parameter decreases in each deletion branch, and it can only be negative in a leaf. Thus a path $P$ leading from the root to a leaf in $T(I)$ can include at most $k + 1$ edges which are not in $E^*$. Let $Q = v_0 v_1 \dots v_q$ be a subpath of $P$ containing only edges in $E^*$.

First, we observe the fact that given a set $L$ of vertices in a simple 3-connected planar graph $G$ and a set $\mathcal{F}$ of faces each having at least 2 vertices from $L$ on their border, we have $|\mathcal{F}| \leq \max\{6|L| - 12, 2\}$. To see this, we define the planar graph $G'$ such that $V(G') = L$ and for each face $F \in \mathcal{F}$ there is an edge in $G'$ connecting two vertices in $V(F) \cap L$. It is easy to observe that in a 3-connected simple graph each pair of vertices can lie on the border of at most two faces. As $G$ is 3-connected, this implies that every edge in $G'$ has multiplicity at most 2. Now, if $|V(G')| \geq 3$ then the planarity of $G'$ yields $|E(G')| \leq 2(3|L| - 6)$. By contrast, if $|V(G')| \leq 2$ then $|E(G')| \leq 2$ is trivial. For each face in $\mathcal{F}$ we defined an edge in $G'$, so $|\mathcal{F}| \leq |E(G')| \leq \max\{6|L| - 12, 2\}$.

For a node $w$ representing a call with input $(H, G, D, \varphi)$, we define $M(w)$ to be the set containing those vertices $\varphi(t)$ on the border of $\varphi(D)$ in $G$ such that $d_H(t) < d_G(\varphi(t))$. As $M(v_i)$ can only decrease after the deletion of some vertices, we get $M(v_{i-1}) \subseteq M(v_i)$ for every $i \in [q]$. Observe that in Step 5 of the branch represented by the edge $v_{i-1}v_i$, a face is added to $\varphi(D)$ that has at least two vertices in $M(v_i) \subseteq M(v_q)$. This follows because the conditions of (1) in Lemma 3.1.4 cannot hold in this step, and so the set $R \subseteq M(v_i)$ in Step 5

```
3-Connected Planar Cleaning $(H,G)$

1. Perform the 3-connectivity test.

2. Let $H_\theta$ denote an embedded version of $H$, and let $G_{\theta_1}$ and $G_{\theta_2}$ be the two
   possible embedded versions of $G$. For $i = 1, 2$ do:

     3. Let $xy \in E(H)$ be arbitrary. For all $(a,b)$ where $ab \in E(G)$ do:
          4. Let $\varphi_{a,b}$ denote the function mapping $x$ to $a$ and $y$ to $b$.
             Output('Yes') if GrowSolution($H_\theta, G_{\theta_i}, xy, \varphi_{a,b}$) returns 'Yes'.

5. Output('No').
```

Figure 3.5: The algorithm solving Cleaning(*3-Connected-Planar, Planar*).

has cardinality at least 2. By Step 4 of the algorithm, $|M(v_q)| \leq 2k$. As shown above, there can be at most $\max\{12k-12, 2\} \leq 12k-10$ faces in $G$ that are adjacent to at least 2 vertices in $M(v_q)$, so the number of extensions branches in $Q$, i.e. the length of $Q$ is at most $12k-10$. This enables us to bound the length of $P$, which is at most $k+1+(k+1)(12k-10) < 13k^2$. As every node in $T(I)$ has at most $2k+2$ children, there are at most $(2k+2)^{13k^2-1}$ leaves in $T(I)$, so the number of nodes in $T(I)$ is at most $f(k) = 2(2k+2)^{13k^2-1}$. □

By careful implementation, it can be ensured that the amount of work done when extending $\varphi$ on a face $F$ is linear in $|V(F)|$, as we only spend constant time at a given vertex. This implies that the consecutive iteration of Steps 3, 4, and 5 can be performed in a total of linear time in $|V(G)|$. As other steps also can be performed in time linear in $|V(G)|$, by Lemma 3.1.5 we can conclude that the running time of *GrowSolution* on input $(H, G, D, \varphi)$ is $f(k)|V(G)| = 2(2k+2)^{13k^2-1}|V(G)|$, where $k = |V(G)| - |V(H)|$.

As a result, there is an algorithm that solves Cleaning(*3-Connected-Planar, Planar*) in FPT time. The steps of the decision version of this algorithm are described in Figure 3.5. Its correctness easily follows from the discussion above. As it calls *GrowSolution* at most $4|E(G)| = O(n)$ times, we can conclude:

**Theorem 3.1.6.** *The* Cleaning(*3-Connected-Planar, Planar*) *problem on input* $(H,G)$ *can be solved in time* $k^{O(k^2)}n^2$*, where* $n = |V(H)|$ *and* $|V(G)| = n + k$.

## 3.2 Trees

The aim of this section is to present an FPT algorithm for Cleaning(*Tree*,–). We parameterize this problem by assigning the parameter $k = |V(G)| - |V(T)|$ for an input $(T, G)$.

Note that if $(T, G)$ is solvable, meaning that $T$ is an induced subgraph of $G$, then the treewidth of $G$ is at most $k+1$. However, this does not yield an obvious way to deal with the problem, as problems related to isomorphism typically remain hard for graphs of bounded treewidth. In particular, Induced Subgraph Isomorphism remains NP-hard when restricted to inputs $(H, G)$ where $H$ is a tree and $G$ has treewidth 2 [110]. Although the restriction of Graph Isomorphism to graphs of treewidth at most $k$ can be solved in $O(n^{k+4.5})$ [15], the parameterized complexity of this problem with $k$ being the parameter is still unknown. Note also that since Cleaning(*Tree*,–) contains the Induced Path prob-

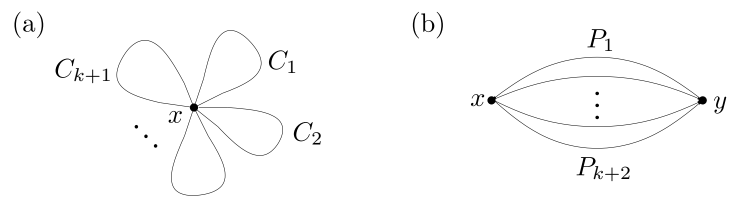


Figure 3.6: Figures (a) and (b) illustrating Reductions $\mathcal{A}$ and $\mathcal{B}$, respectively.

lem, the standard parameterization where the parameter is $|V(T)|$ yields a W[1]-complete problem [29].

W.l.o.g. we can assume that $G$ is simple, $n = |V(T)| > k$ (otherwise we can solve the problem by a brute force algorithm) and $e = |E(G)| = O(kn)$ (as we can automatically refuse instances where $e > n-1+k(n+k-1)$). Let $S$ be a fixed solution, i.e. let $G-S = T_S$ be a tree isomorphic to $T$. Throughout the run of the algorithm, we can assume that $G$ is connected, since by $n > k$ it is trivial to find the unique connected component of $G$ containing $T_S$.

## 3.2.1 Preprocessing

First, we introduce two kinds of reductions, each deleting some vertices from $G$ which must be included in $S$. See Figure 3.6 for an illustration.

**Reduction $\mathcal{A}$: cycles with one common vertex.** If for some vertex $x \in V(G)$ there exist cycles $C_1, \dots, C_{k+1}$ in $G$ such that $V(C_i) \cap V(C_j) = \{x\}$ for each pair $i \neq j$, then delete $x$.

The soundness of Reduction $\mathcal{A}$ is easy to see. Indeed, if such an $x$ is not contained in $S$, then $S$ must contain at least one vertex from each cycle $C_i$, implying $|S| \geq k+1$, a contradiction.

To implement Reduction $\mathcal{A}$, we can check whether the condition given in the reduction above holds for some $x \in V(G)$, by applying a technique based on generalized matchings, proposed by Bodlaender [19].

Using this method, we can reduce our problem to a b-matching problem in a graph having $O(|V(G)|)$ vertices and $O(|E(G)|)$ edges, with the degree constraints being at most $2k+2$ for each vertex and $O(|V(G)|)$ in total. Applying the algorithm of Gabow [59] for the obtained problem, we can solve it in $O(\sqrt{|V(G)|}|E(G)|) = O(\sqrt{n}e) = O(kn\sqrt{n})$ time. This means that Reduction $\mathcal{A}$ can be performed in $O(kn^{5/2})$ total time for all vertices of $G$.

**Reduction $\mathcal{B}$: disjoint paths between two vertices.** Suppose for some $x, y \in V(G)$ that there exist paths $P_1, \dots, P_{k+2}$ from $x$ to $y$ which are disjoint apart from their endpoints. Then, branch in two directions, deleting $x$ in the first, and deleting $y$ in the second branch.

The correctness of this rules can be seen by showing that $x$ or $y$ must be included in any solution $S$ of size at most $k$. Indeed, assuming $x, y \notin S$ implies the existence of a cycle through $x$ and $y$ in $G-S$, which is a contradiction. Using standard flow techniques we can check in time $O(ke)$ whether $(x, y)$ is such a pair of vertices, so finding such a pair takes time $O(ken^2) = O(k^2n^3)$. Since $|S| = k$, we can apply Reduction $\mathcal{B}$ at most $k$ times, which means a total of at most $2^k$ branches.

Now denote by $K$ the minimal connected subgraph of $G$ containing every cycle of $G$. Note that $K$ is unique, and is an induced subgraph of $G$. Let $K_3$ denote the vertices of $K$ whose degree in $K$ is at least 3.

**Lemma 3.2.1.** *If Reductions $\mathcal{A}$ and $\mathcal{B}$ cannot be applied, then $d_K(x) \leq k^2+k$ for every $x \in V(K-S)$ and $|K_3| < g(k) = 2k^3(k+1)+3k = O(k^4)$.*

*Proof.* Let us assume that $x \in V(K-S)$ has neighbors $v_1, v_2, \ldots, v_{k^2+k+1}$ in $K$. Observe that $K-S$ is a tree, and each of its leaves is adjacent to some vertex of $S$. Using this, we obtain that the edges $xv_i$ (for $i \in [k^2+k+1]$) can be extended to internally disjoint paths in $K$ starting from $x$, ending in a vertex of $S$, and having internal vertices in $K-S$. We note that such a path can be of length 1. As $|S| \leq k$, there must exist a vertex $s \in S$ such that at least $\lceil (k^2+k+1)/k \rceil = k+2$ of these paths end in $s$. Now, these paths form at least $k+2$ internally disjoint paths between $x$ and $s$, yielding a possibility for Reduction $\mathcal{B}$, a contradiction. This shows that $d_K(x) \leq k^2+k$ for every $x \in V(K-S)$.

Next, we are going to show that a vertex $s$ in $S$ cannot be adjacent to too many vertices in $V(K-S)$, as this would imply the existence of $k+1$ cycles whose only common vertex is $s$. For this reasoning, we need to prove the following simple fact: given a tree $T'$ with maximum degree $d$ and a set $Z \subseteq V(T')$ with cardinality at least $pd+2$, there always exists a set $\mathcal{P}$ of $p+1$ disjoint paths connecting vertices of $Z$.

To prove this claim, let us regard $T'$ as a rooted tree. We add paths to $\mathcal{P}$ in the following manner: we always choose a new path to put in $\mathcal{P}$ such that its distance from the root is the largest possible. When adding the path $P$ to $\mathcal{P}$, we delete those vertices from $Z$ that can no longer be connected to the root without crossing $P$. This ensures that connecting any two vertices of $Z$ in $T'$ always results in a path that can be added to $\mathcal{P}$. Note that by our choice on $P$ and by the maximum degree $d$ of $T'$ we obtain that $|Z|$ can decrease at most by $d$ in each step, except for the case when $P$ contains the root and thus $|Z|$ might decrease by $d+1$. Therefore, we can indeed put $p+1$ paths into $\mathcal{P}$.

Now, for a vertex $s \in S$, let $T_s$ denote the unique minimal subtree of $K-S$ containing $Z_s = N_{V(K-S)}(s)$. Suppose $|Z_s| \geq k(k^2+k)+2$ for some $s$. As every vertex in $T_s$ has maximum degree $k^2+k$ by the first claim of the lemma, using the claim proved above we get that there are $k+1$ disjoint paths in $T_s$ connecting vertices of $Z_s$. These paths together with $s$ form $k+1$ cycles whose only common vertex is $s$, contradicting our assumption that Reduction $\mathcal{A}$ is not applicable.

Thus, we get $|Z_s| \leq k(k^2+k)+1 = k^2(k+1)+1$ for each $s \in S$. Let $L$ denote the leaves of $K-S$. Every vertex in $L$ has a neighbor in $S$, so $L \subseteq N_{V(K-S)}(S) = \bigcup_{s \in S} Z_s$, implying $|L| \leq |N_{V(K-S)}(S)| \leq k^3(k+1)+k$. Observe that every vertex in $K_3 \setminus (S \cup N_{V(K-S)}(S))$ has degree at least 3 also in $K-S$. Since the number of vertices in the tree $K-S$ having degree at least 3 is less than the number $|L|$ of leaves, we get $|K_3| < |S| + |N_{V(K-S)}(S)| + |L| \leq |S| + 2|N_{V(K-S)}(S)|$, implying $|K_3| < 2k^3(k+1)+3k$. □

### 3.2.2 Growing a mapping

From now on, we assume that Reductions $\mathcal{A}$ and $\mathcal{B}$ cannot be applied. At this point, the algorithm checks whether the conditions of Lemma 3.2.1 are fulfilled, and correctly outputs 'No' if the conditions do not hold. Let $\phi$ denote the isomorphism from $T$ to $T_S$ that we are looking for. As in Section 3.1, we try to grow a partial mapping from $T$ to $T_S$, which is always a restriction of $\phi$. To begin, the algorithm chooses an arbitrary starting vertex $r_0$ in $T$, and branches on the choice of $\phi(r_0)$ in $G$, which means $|V(G)|$ possibilities.

Throughout its running, the algorithm may modify $G$ by deleting vertices of $S$ from it. We denote by $G^i$ the graph obtained from $G$ after the $i$-th step, with $G = G^0$. Assume that in the $i$-th step of the algorithm there is a subtree $D^i$ of $T$ on which $\phi$ is already known. The algorithm proceeds step by step, choosing a leaf $r^i$ of $D^i$ in the $i$-th step that has not been examined yet. For the chosen vertex $r^i$, it determines $\phi$ on $N_T(r^i)$ by applying a method described below. This means also that it adds $N_T(r^i)$ to $D^i$ to get $D^{i+1}$, deletes $N_G(\phi(r^i)) \cap S$ from $G^i$ to get $G^{i+1}$, and checks whether $\phi$ is still an isomorphism. When determining $\phi$ on $N_T(r^i)$, the algorithm may branch into a bounded number of branches, or may proceed with a single branch. Accordingly, we distinguish between *branching* and *simple cases.*

Let us describe the details of a single step executed by the algorithm. First, it checks whether $|V(G^i)| \geq |V(T)|$ holds, outputting 'No' if the condition fails. Next, it verifies some simple conditions considering the neighbors of $r^i$ and $\phi(r^i) = r'^i$. To do this, it determines the minimal connected subgraph $K^i$ of $G^i$ containing every cycle of $G^i$. Note that $K^i$ can be constructed from $G^i$ easily in linear time, as the 2-connected components of a graph can be determined in linear time, e.g. by applying depth first search [37].

To proceed, let us introduce some new notation. We divide the vertices of $N_T(r^i)$ into two groups as follows:

- those neighbors of $r^i$ that are in $D^i$,
- those neighbors of $r^i$ that are not in $D^i$. Let $t^i_1, \dots, t^i_{\alpha^i}$ denote these vertices, and let $T^i_j$ be the tree component of $T - r^i$ containing $t^i_j$.

Similarly, we classify the vertices of $N_G(r'^i)$ into three groups:

- those neighbors of $r'^i$ that are in $\phi(D^i)$,
- those neighbors of $r'^i$ outside $\phi(D^i)$ that are connected to $r'^i$ by edges not in $K^i$. Let $t'^i_1, \dots, t'^i_{\beta^i}$ denote these vertices, and $T'^i_j$ denote the component of $G^i - r'^i$ that includes $t'^i_j$. Observe that either $T'^i_j$ is a tree, or $r'^i \notin V(K^i)$ and $T'^i_j$ contains $K^i$.
- those neighbors of $r'^i$ outside $\phi(D^i)$ that are connected to $r'^i$ by edges in $K^i$. Let $\gamma^i$ be the number of such vertices.

Clearly, $\alpha^i \leq \beta^i + \gamma^i$, and the equality holds if and only if $N_{G^i}(r'^i) \cap S = \emptyset$. Thus, if the algorithm finds that $\alpha^i > \beta^i + \gamma^i$, then it outputs 'No'.

First, let us observe that if the tree $T^i_h$ is isomorphic to $T'^i_j$ for some $h$ and $j$, then w.l.o.g. we can assume that $\phi(T^i_h) = T'^i_j$. As the trees of a forest can be classified into equivalence classes with respect to isomorphism in time linear in the size of the forest [6, 78], this case can be noticed easily. Given two isomorphic trees, an isomorphism between them can also be found in linear time, so the algorithm can extend $\phi$ on $T^i_h$, adding also $T^i_h$ to the subgraph $D^i$. Hence, we only have to deal with the following case: no tree $T^i_h$ ($h \in [\alpha^i]$) is isomorphic to one of the graphs $T'^i_j$ ($j \in [\beta^i]$). This argument makes our situation significantly easier, since every graph $T'^i_j$ must contain some vertex from $S$. Therefore $\beta^i \leq |S| = k$. Clearly, if $r'^i \notin V(K^i)$ then $\gamma^i = 0$. If $r'^i \in V(K^i)$ then $r'^i$ can have degree at most $k^2 + k$ in $K^0$, and thus in $K^i$, by Lemma 3.2.1. Thus, we get $\gamma^i \leq k^2 + k$, implying also $\alpha^i \leq \beta^i + \gamma^i \leq k^2 + 2k$. The algorithm determines $\alpha^i, \beta^i$ and $\gamma^i$ in each step, and outputs 'No' if these bounds do not hold for them.

The algorithm faces one of the following two cases at each step.

**Simple case:** $\beta^i + \gamma^i \leq 1$. In this case, $\alpha^i \leq 1$. If $\beta^i + \gamma^i = 0$ then $\alpha^i = 0$, hence the algorithm proceeds with the next step by choosing another leaf of $D^i$ not yet visited. Otherwise, let $v$ be the unique vertex in $N_{G^i}(r'^i) \setminus V(\phi(D^i))$. If $\alpha^i = 0$ then $v$ must be in $S$, otherwise $\phi(t^i_1) = v$. According to this, the algorithm deletes $v$ or extends $\phi$ on $t^i_1$, adding also $t^i_1$ to $D^i$.

**Branching case:** $\beta^i + \gamma^i \geq 2$. In this case, the algorithm branches on every possible choice of determining $\phi$ on $N_T(r^i)$. Guessing $\phi(v)$ for a vertex $v \in N_{V(T-D^i)}(r^i)$ can result in at most $\beta^i + \gamma^i$ possibilities, so the number of possible branches in a branching step is at most $(\beta^i + \gamma^i)^{\alpha^i} \leq (k^2 + 2k)^{k^2+2k}$. After guessing $\phi(v)$ for each vertex $v \in N_{V(T-D^i)}(r^i)$, the algorithm puts the remaining vertices $N_G(r'^i) \setminus \{\phi(v) \mid v \in N_T(r^i)\}$ into $S$, deleting them from $G^i$.

**Lemma 3.2.2.** *In a single branch of a run of the algorithm described above on a solvable input for the* Cleaning(*Tree,*$-$) *problem with parameter* $k$*, there can be at most* $g(k) + 2k - 2 = 2k^3(k+1) + 5k - 2 = O(k^4)$ *branching steps.*

*Proof.* We use the notation applied in the description of the algorithm. The $i$-th step can only be a branching case if either $\gamma^i \geq 2$, $\beta^i \geq 2$, or $\beta^i = \gamma^i = 1$ holds. For each of these cases, we give an upper bound on the number of steps in a single branch of a run of the algorithm where these cases can happen.

To determine a bound for the case $\gamma^i \geq 2$, let $r^*$ be the first vertex in $T$ examined in a step such that $\phi(r^*)$ is in $K^0$. Recall that $K^0 - S$ is a tree, so supposing $\phi(r^i) \in V(K^0)$ we get that if $r^i \neq r^*$ then for the edge $e$ incident to $r^i$ in $D^i$ it must hold that $\phi(e)$ is in $K^0$. Now, observe that if $\gamma^i \geq 2$ holds, then this implies that either $r^i = r^*$ or $\phi(r^i)$ has at least three edges incident to it in $K^0$. The latter means that $r^i \in K_3$, where $K_3$ denotes the vertices of $K^0$ having degree at least three in $K^0$. Thus, the condition $\gamma^i \geq 2$ can hold in at most $|K_3| + 1 \leq g(k)$ steps, by Lemma 3.2.1.

Now, if the algorithm finds $\beta^i \geq 2$, then recall that both $T_1'^i$ and $T_2'^i$ include at least one vertex from $S$, and thus $G^i - \phi(D^i)$ has more connected components containing vertices of $S$ than $G^i - \phi(D^i - r^i)$ has. It is easy to see that this can be true for only at most $|S| - 1$ such vertices $r^i$, so this case can happen at most $k - 1$ times in a single branch of a run of the algorithm.

Finally, let $S^*$ denote those vertices of $S$ that are not contained in $K^0$. Clearly, if $s \in S^*$, then $s$ is not contained in any cycle of $G$, so $|N_G(s) \cap V(T_S)| \leq 1$. Now, if $\beta^i = \gamma^i = 1$, then $r'^i \in V(K)$ and the edge $r'^i t_1'^i$ must be one of the edges that connect to $K^0$ a tree in $G - K^0$ containing a vertex in $S^*$. Observe that there can be at most $|S^*| \leq k - 1$ such edges. Therefore, the claim follows. □

As Lemma 3.2.2 only bounds the number of branching steps for solvable inputs, the algorithm ensures the same bound on every input by maintaining a counter for these steps. Thus, it outputs 'No' if it encounters a branching case for the $(g(k) + 2k - 1)$-th time.

As the number of branches in a branching case is $k^{O(k^2)}$, and the number of branching cases in a single branch of a run of the algorithm is $O(k^4)$, the number of leaves in the search tree explored by the algorithm (i.e. the number of steps where the algorithm stops, regarding all the branches in total) is $k^{O(k^6)}$. At each vertex, the algorithm uses time at most linear in $|V(G)|$. The number of steps performed in a single branch of a run of the algorithm is at most $|V(T)|$, hence the algorithm needs quadratic time after choosing $\phi(r_0)$ for the starting vertex $r_0$. Trying all possibilities for $\phi(r_0)$ increases this to cubic time. Reductions $\mathcal{A}$ and $\mathcal{B}$ can also be executed in cubic time, as argued before, so we can conclude:

**Theorem 3.2.3.** *The* Cleaning(*Tree,*$-$) *problem on input* $(T, G)$ *can be solved in* $k^{O(k^6)} n^3$ *time, where* $n = |V(T)|$ *and* $|V(G)| = n + k$*.*

CHAPTER 4

# Induced Subgraph Isomorphism on interval graphs

In this chapter, we discuss the parameterized complexity of the following problem: given two interval graphs $G$ and $H$, decide whether we can delete some vertices of $G$ to obtain a graph isomorphic to $H$. On the one hand, we prove that this special case of Induced Subgraph Isomorphism is NP-hard, and we show that it is W[1]-hard when parameterized by the $|V(H)|$, denoting the number of vertices in the smaller graph.

On the other hand, we present a newly developed FPT algorithm for this problem, when parameterized by the number $|V(G)| - |V(H)|$, denoting the number of vertices which we have to delete from $G$ to obtain a graph isomorphic to $H$. Using the notation of the previous chapter, we will denote the resulting parameterized problem by Cleaning(*Interval*, *Interval*), with *Interval* standing for the class of interval graphs.

Interval graphs form an important and widely studied class of graphs. Thanks to their strict structure, many NP-hard problems become polynomial-time solvable when restricted to interval graphs [64, 51]. They have numerous applications in scheduling problems but also in various areas of computational biology. Apart from the theoretical interest, the investigation of the Cleaning problem for interval graphs is also motivated by its similarity with an important problem in biology, namely the Arc-Preserving Subsequence problem [109].

In Section 4.1 we give a brief introduction to a data structure called labeled PQ-trees which yield a canonical form for interval graphs. Section 4.2 contains the obtained hardness results, and Sections 4.3 and 4.4 cover our FPT algorithm for Cleaning(*Interval*, *Interval*). The results of this chapter appear in [108].

## 4.1 Interval graphs and labeled PQ-trees

Let $G$ be an interval graph, meaning that $G$ can be regarded as the intersection graph of a set of intervals. Formally, an interval representation of $G$ is a set $\{I_i \mid i \in [n]\}$ of intervals, where $I_i$ and $I_j$ intersect each other if and only if $v_i$ and $v_j$ are adjacent. We say that two intervals *properly intersect*, if they intersect, but none of them contains the other.

Let $\mathcal{C}(G)$ be the set of all maximal cliques in $G$, and let $\mathcal{C}(v) = \{C \mid v \in C, C \in \mathcal{C}(G)\}$ for some $v \in V(G)$. It is known that a graph is an interval graph if and only if its maximal cliques can be *ordered consecutively*, i.e. there is an ordering of $\mathcal{C}(G)$ such that the cliques in $\mathcal{C}(v)$

form a consecutive subsequence [65]. Note that any interval representation gives rise to a natural ordering of $\mathcal{C}(G)$, which is always a consecutive ordering. The set of all consecutive orderings of $\mathcal{C}(G)$ are usually represented by PQ-trees, a data structure introduced by Booth and Lueker [22].

A *PQ-tree* of $G$ is a rooted tree $T$ with ordered edges with the following properties: every non-leaf node is either a Q-node or a P-node, each P-node has at least 2 children, each Q-node has at least 3 children, and the leaves of $T$ are bijectively associated with the elements of $\mathcal{C}(G)$. For an illustration, see Figure 4.1. The *frontier* $F(T)$ of the PQ-tree $T$ is the permutation of $\mathcal{C}(G)$ that is obtained by ordering the cliques associated with the leaves of $T$ simply from left to right. Two PQ-trees $T_1$ and $T_2$ are *equivalent*, if one can be obtained from the other by applying a sequence of the following transformations: permuting the children of a P-node arbitrarily, or reversing the children of a Q-node. The consecutive orderings of the maximal cliques of a graph can be represented by a PQ-tree in the following sense: for each interval graph $G$ there exists a PQ-tree $T$, such that $\{F(T') \mid T'$ is a PQ-tree equivalent to $T\}$ yields the set of all consecutive orderings of $\mathcal{C}(G)$. Such a PQ-tree *represents* $G$. For any interval graph $G$ a PQ-tree representing it can be constructed in linear time [22].

This property of PQ-trees can be used in the recognition of interval graphs. However, to examine isomorphism of interval graphs, the information stored in a PQ-tree is not sufficient. For this purpose, a new data structure, the labeled PQ-tree has been defined [96, 35]. For a PQ-tree $T$ and some node $s \in V(T)$, let $T_s$ denote the subtree of $T$ rooted at $s$. For each vertex $v$ in $G$, let the *characteristic node* $R(v)$ of $v$ in a PQ-tree $T$ representing $G$ be the deepest node $s$ in $T$ such that the frontier of $T_s$ contains $\mathcal{C}(v)$. For a node $s \in V(T)$, we will also write $R^{-1}(s) = \{x \in V(G) \mid R(x) = s\}$, and if $T'$ is a subtree of $T$, then $R^{-1}(T') = \{x \in V(G) \mid R(x) \in V(T')\}$. Observe that if $R(v)$ is a P-node, then every clique in the frontier of $T_{R(v)}$ contains $v$. It is also true that if $R(v)$ is a Q-node with children $x_1, x_2, \ldots, x_m$, then those children of $R(v)$ whose frontier contains $v$ form a consecutive subseries of $x_1, \ldots x_m$. Formally, there must exist two indices $i < j$ such that $\mathcal{C}(v) = \{C \mid C \in F(T_{x_h})$ for some $i \leq h \leq j\}$.

A *labeled PQ-tree* of $G$ is a labeled version of a PQ-tree $T$ of $G$ where the labels store the following information. If $x$ is a P-node or a leaf, then its label is simply $|R^{-1}(x)|$. If $q$ is a Q-node with children $x_1, x_2, \ldots, x_m$ (from left to right), then for each $v \in R^{-1}(T_q)$ we define $Q_q(v)$ to be the pair $[a, b]$ such that $x_a$ and $x_b$ are the leftmost and rightmost children of $q$ whose frontier in $T$ contains $\mathcal{C}(v)$. (See again Figure 4.1.) Also, if $Q_q(v) = [a, b]$ for some vertex $v$, then we let $Q_q^{\text{left}}(v) = a$ and $Q_q^{\text{right}}(v) = b$. For some $1 \leq a \leq b \leq m$, the pair $[a, b]$ is a *block* of $q$. Considering blocks of a Q-node, we will use a terminology that treats them like intervals, so two blocks can be disjoint, intersecting, they contain indices, etc. The label $L(q)$ of $q$ encodes the values $|L_q(a, b)|$ for each $a < b$ in $[m]$, where $L_q(a, b)$ is the set $\{v \in R^{-1}(q) \mid Q_q(v) = [a, b]\}$.

Note that a PQ-tree can be labeled in linear time. Two labeled PQ-trees are *identical*, if they are isomorphic as rooted trees and the corresponding vertices have the same labels. Two labeled PQ-trees are *equivalent*, if they can be made identical by applying a sequence of transformations as above, with the modification that when reversing the children of a Q-node, its label must also be adjusted correctly. The key theorem that yields a way to handle isomorphism questions on interval graphs is the following:

**Theorem 4.1.1** ([96])**.** *Let $G_1$ and $G_2$ be two interval graphs, and let $T^L(G_1)$ and $T^L(G_2)$ be the labeled version of a PQ-tree representing $G_1$ and $G_2$, respectively. Then $G_1$ is isomorphic to $G_2$ if and only if $T^L(G_1)$ is equivalent to $T^L(G_2)$.*

Given a Q-node $q$ in a PQ-tree $T$, let $x_1, \ldots, x_m$ denote its children from left to right. For a given child $x_i$ of $q$, we define $M_q(i)$ to be the set of vertices $v \in R^{-1}(q)$ for which $Q_q(v)$

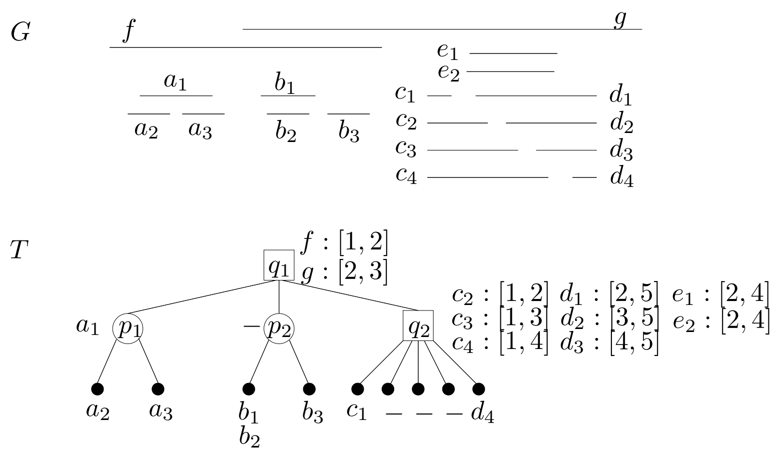


Figure 4.1: An interval representation of an interval graph $G$ and a labeled PQ-tree $T$ representing $G$. Squares, white and black circles represent Q-nodes, P-nodes and leaves, resp. For each node $x$ in $T$, we listed the vertices in $R^{-1}(x)$, together with the values $Q_x(v)$ for each $v \in R^{-1}(x)$ where $x$ is a Q-node. As an example, the frontier of $p_2$ is $(\{b_1, b_2, f, g\}, \{b_3, f, g\})$.

contains $i$, i.e. $M_q(i)$ is the union of those sets $L_q(a, b)$ for which $[a, b]$ contains $i$. Clearly, $M_q(i) \neq M_q(j)$ if $i \neq j$, since this would imply the interchangeability of the nodes $x_i$ and $x_j$. We say that some $w \in R^{-1}(q)$ *starts* or *ends* at $i$ if $Q_q^{\text{left}}(v) = i$ or $Q_q^{\text{right}}(v) = i$, respectively. We also denote by $M_q^+(i)$ and $M_q^-(i)$ the set of vertices that start or end at $i$, respectively. The maximality of the cliques in $F(T_{x_i})$ implies the following observation.

**Proposition 4.1.2.** *If $q$ is a Q-node in a PQ-tree $T$ and $x_i$ is the $i$-th child of $q$, then neither $R^{-1}(T_{x_i}) \cup M_q^+(i)$ nor $R^{-1}(T_{x_i}) \cup M_q^-(i)$ can be empty.*

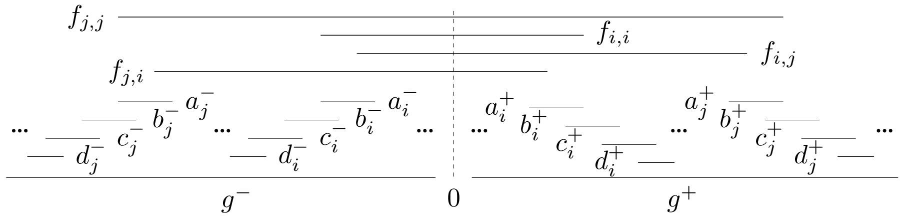
Given some interval representation $\rho$ for an interval graph $G$, we denote by $v_\rho^{\text{left}}$ and $v_\rho^{\text{right}}$ the left and right endpoints of the interval representing $v \in V(G)$. If no confusion arises, then we may drop the subscript $\rho$.

## 4.2 Hardness results

In this section, we prove the NP-hardness of Induced Subgraph Isomorphism for the case of interval graphs (the Cleaning(*Interval, Interval*) problem), and we also show the parameterized hardness of this problem, where the parameter is the size of the smaller graph.

**Theorem 4.2.1.** *(1) The* Induced Subgraph Isomorphism *problem is W[1]-hard if both input graphs are interval graphs, and the parameter is the number of vertices in the smaller input graph.*
*(2) The* Induced Subgraph Isomorphism *problem is NP-complete if both input graphs are interval graphs.*

*Proof.* To prove (1), we give an FPT reduction from the parameterized Clique problem. Let $F = (V, E)$ and $k$ be the input graph and the parameter given for Clique. We assume w.l.o.g. that $F$ is simple and $V = \{v_i \mid i \in [n]\}$. We construct two interval graphs $G$ and $H$ with $|V(H)| = O(k^2)$ such that $H$ is an induced subgraph of $G$ if and only if $F$ has a $k$-clique.



$f_{j,j}$
$f_{i,i}$
$f_{i,j}$
$f_{j,i}$
$a_j^-$ $b_j^-$ $c_j^-$ $d_j^-$
$a_i^-$ $b_i^-$ $c_i^-$ $d_i^-$
$a_i^+$ $b_i^+$ $c_i^+$ $d_i^+$
$a_j^+$ $b_j^+$ $c_j^+$ $d_j^+$
$g^-$
0
$g^+$

Figure 4.2: Illustration of the construction of the graph $G$. (The picture assumes $v_iv_j \in E$.)

The vertex set of $G$ consist of the vertices $a_i^s, b_i^s, c_i^s, d_i^s, f_{i,i}$ for each $i \in [n]$ and $s \in \{-,+\}$, vertices $f_{i,j}, f_{j,i}$ for each $v_iv_j \in E$, and two vertices $g^-$ and $g^+$. Note that $|V(G)| = 9n+2|E|+2$, which is polynomial in $n$. We define the edge set of $G$ by giving an interval representation for $G$. The intervals $I(x)$ representing a vertex $x \in V(F)$ are defined below. See also the illustration of Figure 4.2.

| | | |
|---|---|---|
| $I(a_i^+) = [10i-8, 10i-5]$ | $I(a_i^-) = [-10i+5, -10i+8]$ | if $i \in [n]$ |
| $I(b_i^+) = [10i-6, 10i-3]$ | $I(b_i^-) = [-10i+3, -10i+6]$ | if $i \in [n]$ |
| $I(c_i^+) = [10i-4, 10i-1]$ | $I(c_i^-) = [-10i+1, -10i+4]$ | if $i \in [n]$ |
| $I(d_i^+) = [10i-2, 10i]$ | $I(d_i^-) = [-10i, -10i+2]$ | if $i \in [n]$ |
| $I(f_{i,i}) = [-10i+5, 10i-5]$ | | if $i \in [n]$ |
| $I(f_{i,j}) = [-10i+7, 10j-7]$ | $I(f_{j,i}) = [-10j+7, 10i-7]$ | if $v_iv_j \in E$ |
| $I(g^-) = [-10n, -1]$ | $I(g^-) = [1, 10n]$ | |

Note that this construction is symmetric in the sense that for any interval $[x_1, x_2]$ in this interval representation, the interval $[-x_2, -x_1]$ is also present.

Also, we define the graph $H$, having $k^2 + 8k + 2$ vertices, as follows. Let the vertex set of $H$ consist of the vertices $\widetilde{a}_i^s, \widetilde{b}_i^s, \widetilde{c}_i^s, \widetilde{d}_i^s$ for each $i \in [k]$ and $s \in \{-,+\}$, the vertices $\widetilde{f}_{i,j}$ for each $(i,j) \in [k]^2$, and two vertices $\widetilde{g}^-$ and $\widetilde{g}^+$. Again, we define the edge set of $H$ by giving an interval representation for $H$ as follows.

| | | |
|---|---|---|
| $I(\widetilde{a}_i^+) = [10i-8, 10i-5]$ | $I(\widetilde{a}_i^-) = [-10i+5, -10i+8]$ | if $i \in [k]$ |
| $I(\widetilde{b}_i^+) = [10i-6, 10i-3]$ | $I(\widetilde{b}_i^-) = [-10i+3, -10i+6]$ | if $i \in [k]$ |
| $I(\widetilde{c}_i^+) = [10i-4, 10i-1]$ | $I(\widetilde{c}_i^-) = [-10i+1, -10i+4]$ | if $i \in [k]$ |
| $I(\widetilde{d}_i^+) = [10i-2, 10i]$ | $I(\widetilde{d}_i^-) = [-10i, -10i+2]$ | if $i \in [k]$ |
| $I(\widetilde{f}_{i,i}) = [-10i+5, 10-5]$ | | if $i \in [k]$ |
| $I(\widetilde{f}_{i,j}) = [-10i+7, 10j-7]$ | $I(\widetilde{f}_{j,i}) = [-10j+7, 10i-7]$ | if $i, j \in [k]$, $i \neq j$ |
| $I(\widetilde{g}^-) = [-10k, -1]$ | $I(\widetilde{g}^-) = [1, 10k]$ | |

First, if $C$ is a set of $k$ vertices in $F$ that form a clique, then $H$ is isomorphic to the subgraph of $G$ induced by the vertices $a_i^s, b_i^s, c_i^s, d_i^s, f_{i,i}$ for each $v_i \in C$ and $s \in \{-,+\}$, the vertices $f_{i,j}, f_{j,i}$ for each $\{v_i, v_j\} \subseteq C$, and the two vertices $g^-$ and $g^+$. This can be proven by presenting an isomorphism $\varphi$ from $H$ to the subgraph of $G$ induced by these vertices. It is easy to verify that the function $\varphi$ defined below indeed yields an isomorphism. Here, $c(i)$ denotes the index of the $i$-th vertex in the clique $C$, i.e. $C = \{v_{c(i)} \mid i \in [k]\}$.

| | |
|---|---|
| $\varphi(\widetilde{x}_i^s) = x_{c(i)}^s$ | for each $x \in \{a, b, c, d\}, s \in \{-,+\}, i \in [k]$ |
| $\varphi(\widetilde{f}_{i,j}) = f_{c(i),c(j)}^s$ | for each $i, j \in [k]^2$ |
| $\varphi(\widetilde{g}^s) = g^s$ | for each $s \in \{-,+\}$ |

For the other direction, suppose that $\varphi$ is an isomorphism from $H$ to an induced subgraph of $G$. We set $F = \{f_{i,j} \mid i = j \text{ or } v_iv_j \in E\}$, and we define $Z$ to contain those vertices of $G$ whose interval contains 0.

**Claim.** If there is a disjoint union of two $k$-stars in $K$ with centers $u_1$ and $u_2$, induced by vertices $\{u_1, u_2\} \cup J$, then $\{\varphi(u_1), \varphi(u_2)\} = \{g^-, g^+\}$ and $\varphi(J) \cap F = \emptyset$. To prove this claim, note that the vertices of $J$ are independent, so there can be at most one vertex in $\varphi(J)$ whose interval contains 0. Thus, either $\varphi(u_1)$ or $\varphi(u_2)$ must not be in $Z$, and must be adjacent to at least $k$ vertices not in $Z$. This implies that $\varphi(u_1)$ or $\varphi(u_2)$ must indeed be $g^-$ or $g^+$. Assuming, say, $\varphi(u_1) = g^-$ (the remaining cases are analogous), we obtain that the only common neighbor of the $k$ vertex of $\varphi(J)$ not adjacent to $\varphi(u_1)$ can be $g^+$. This immediately implies $\{\varphi(u_1), \varphi(u_2)\} = \{g^-, g^+\}$. From this, $\varphi(J) \cap F = \emptyset$ is clear, since no vertex of $J$ is adjacent to both $u_1$ and $u_2$. Hence, the claim is true.

Now, note that for some $x \in \{a, b, c, d\}$, the vertex set $\{\widetilde{x}_i^s \mid i \in [k], s \in \{-, +\}\} \cup \{\widetilde{g}^-, \widetilde{g}^+\}$ induces the disjoint union of two $k$-stars having centers $\widetilde{g}^-$ and $\widetilde{g}^+$ in $H$. Therefore, applying the above claim to each these vertex sets with $x \in \{a, b, c, d\}$, we obtain that $\{\varphi(\widetilde{g}^-), \varphi(\widetilde{g}^+)\} = \{g^-, g^+\}$, and also that $\varphi(\widetilde{X}) \cap F = \emptyset$ for the set $\widetilde{X}$ containing the vertices of the form $\widetilde{x}_i^s$ where $x \in \{a, b, c, d\}, s \in \{-, +\}$ and $i \in [k]$. By the symmetry of $H$ and $G$, we can assume w.l.o.g. that $\varphi(\widetilde{g}^-) = g^-$ and $\varphi(\widetilde{g}^+) = g^+$.

From this, we have that exactly $4k$ vertices of $\varphi(\widetilde{X})$ are represented by an interval whose left endpoint is positive, and the remaining $4k$ vertices of $\varphi(\widetilde{X})$ are represented by an interval whose right endpoint is negative. Now, observe that the vertices of $\widetilde{X}$ induce exactly $2k$ paths of length 4 in $H$, which leads us to the fact that their images by $\varphi$ must also induce 4-paths. Using this, it follows that for each $i \in [k]$ we can define $c(i, +), c(i, -) \in [n]$ such that

$$\varphi(\{\widetilde{a}_i^s, \widetilde{b}_i^s, \widetilde{c}_i^s, \widetilde{d}_i^s\}) = \{a_{c(i,s)}^s, b_{c(i,s)}^s, c_{c(i,s)}^s, d_{c(i,s)}^s\}$$

for each $i \in [k]$ and $s \in \{-, +\}$.

Note also that for both $s \in \{-, +\}$, the vertex $\widetilde{f}_{i,i}$ is adjacent to exactly two vertices from $\{\widetilde{a}_i^s, \widetilde{b}_i^s, \widetilde{c}_i^s, \widetilde{d}_i^s\}$. However, the unique vertex that is adjacent to exactly two vertices in the set $\{a_{c(i,s)}^s, b_{c(i,s)}^s, c_{c(i,s)}^s, d_{c(i,s)}^s\}$ is the vertex $f_{c(i,s),c(i,s)}$. From this, we get that $\varphi(\widetilde{f}_{i,i}) = f_{c(i,-),c(i,-)} = f_{c(i,+),c(i,+)}$, implying also $c(i, -) = c(i, +)$.

Finally, observe that if $i \neq j$, then the vertex $\widetilde{f}_{i,j}$ is adjacent to exactly one vertex both from $\{\widetilde{a}_i^-, \widetilde{b}_i^-, \widetilde{c}_i^-, \widetilde{d}_i^-\}$ and from $\{\widetilde{a}_j^+, \widetilde{b}_j^+, \widetilde{c}_j^+, \widetilde{d}_j^+\}$. This implies that $\varphi(\widetilde{f_{i,j}} = f_{c(i,-),c(j,+)}$ must hold, but $f_{c(i,-),c(j,+)}$ only exists if $v_{c(i,-)}$ and $v_{c(j,+)}$ are adjacent in $F$. Clearly, this implies that the vertices $\{v_{c(i,-)} = v_{c(i,+)} \mid i \in [k]\}$ form a clique in $F$, hence the second direction of the reduction is correct as well.

Observe that by the size of $G$ and $H$, this yields an FPT-reduction from the parameterized Clique problem to the Cleaning(*Interval, Interval*) problem (i.e. the Induced Subgraph Isomorphism problem for interval graphs) parameterized by the number of vertices in the smaller input graph, proving (1). Also, note that the construction of $G$ and $H$ takes time polynomial in $|V(F)|$ and $k$, so by the NP-hardness of the (unparameterized) Maximum Clique problem, this proves that the (unparameterized) Cleaning(*Interval, Interval*) problem is NP-hard as well. Its containment in NP is trivial, finishing the proof of (2). $\square$

## 4.3 Cleaning an interval graph

In this section, we present an algorithm that solves the Cleaning(*Interval, Interval*) problem. Given an input $(G', G)$ of this problem, we call a set $S \subseteq V(G)$ a *solution* for $(G', G)$, if $G'$ is isomorphic to $G - S$. In this case, let $\phi_S$ denote an isomorphism from $G'$ to $G - S$. Remember that $k = |V(G)| - |V(G')|$ is the parameter of the instance $(G', G)$. We denote by $T$ and $T'$ the labeled PQ-tree representing $G$ and $G'$, respectively. Let us fix an interval representation of $G$. For a subset $X$ of $V(G)$, let $X^{\text{left}} = \min\{x^{\text{left}} \mid x \in X\}$ and $X^{\text{right}} = \max\{x^{\text{right}} \mid x \in X\}$.

Our algorithm for Cleaning(*Interval, Interval*) is based on an algorithm denoted by $\mathcal{A}$ whose output on an input $(G', G)$ can be one of the following three concepts:

- a **necessary set**. We call a set $N \subseteq V(G)$ a *necessary set* for $(G', G)$, if $(G', G)$ has a solution if and only if there is a vertex $x \in N$ such that $(G', G - x)$ has a solution. Given a necessary set for $(G', G)$, we can branch on including one of its vertices in the solution.

- a **reduced input**. For subgraphs $H$ and $H'$ of $G$ and $G'$, respectively, we say that $(H', H)$ is a *reduced input* for $(G', G)$, if $(G', G)$ is solvable if and only if $(H', H)$ is solvable, every solution for $(H', H)$ is a solution for $(G', G)$, and $|V(H')| + |V(H)| < |V(G')| + |V(G)|$. Given a reduced input for $(G', G)$, we can clearly solve it instead of solving $(G', G)$.

- an **independent subproblem**. For subgraphs $H$ and $H'$ of $G$ and $G'$, respectively, we say that $(H', H)$ is an *independent subproblem* of $(G', G)$ having parameter $k$, if its parameter is at least 1 but at most $k-1$, for any solution $S$ of $(G', G)$ the set $S \cap V(H)$ is a solution for $(H', H)$, and if $(G', G)$ admits a solution then any solution $S$ of $(H', H)$ can be extended to be a solution for $(G', G)$. Note that given an independent subproblem of $(G', G)$, we can find a vertex of the solution by solving the independent subproblem having parameter smaller than $k$.

Observe that if $N$ is a necessary set for either an independent subproblem or a reduced input for $(G', G)$, then $N$ must be a necessary set for $(G', G)$ as well.

In Section 4.3.1 we make some useful observations about the structure of an interval graph. In Sections 4.3.2 and 4.4, we describe algorithm $\mathcal{A}$, that, given an input instance of CLEANING(*Interval, Interval*) with parameter $k$, does one of the followings in linear time:

- either determines a reduced input for $(G', G)$,

- or branches into at most $f_1(k) = k^{O(k^3)}$ possibilities, in each of the branches producing a necessary set of size at most $2k + 1$ or an independent subproblem of $(G', G)$.

Note that in the first case no branching is involved. If the second case applies and $\mathcal{A}$ branches, then the collection of outputs returned in the obtained branches must contain a correct output. In other words, at least one of the branches must produce an output that is indeed a necessary set of the required size or an independent subproblem of $(G', G)$.

Let us show how such an algorithm can be used as a sub-procedure in order to solve the CLEANING(*Interval, Interval*) problem. (See Figure 4.3 for an outline of the algorithm.) First, we construct an algorithm called *NecessarySet* that given an instance $(G', G)$ of CLEANING(*Interval, Interval*) finds a necessary set for $(G', G)$ in quadratic time. *NecessarySet* works by running $\mathcal{A}$ repeatedly, starting with the given input. In the case when $\mathcal{A}$ returns a reduced input, *NecessarySet* runs $\mathcal{A}$ with this reduced input again. In the case when $\mathcal{A}$ branches, returning a necessary set or an independent subproblem in each branch, *NecessarySet* runs $\mathcal{A}$ on each independent subproblem produced in any of the branches. Applying this method iteratively (and thus possibly branching again), we will get a necessary set at the end of each branch. Note that the parameter of the input decreases whenever a branching happens, and thus the corresponding search tree has at most $f_1(k)^k = k^{O(k^4)}$ leaves. Since at least one of the branches is correct, by taking the union of the necessary sets produced in the leaves of the search tree, we get a necessary set of size $f_2(k) = (2k+1)f_1(k)^k = k^{O(k^4)}$ for $(G', G)$. As each run of $\mathcal{A}$ takes linear time, and the number of calls of $\mathcal{A}$ is also linear in a single chain of branches, the whole algorithm takes quadratic time.

Now, we can solve CLEANING(*Interval, Interval*) by using *NecessarySet*. First, given an input $(G', G)$, we run *NecessarySet* on it. We branch on choosing a vertex $s$ from the produced output to put into the solution, and repeat the whole procedure with input $(G', G - s)$. This means a total of $f_2(k) = (2k+1)f_1(k)^k = k^{O(k^4)}$ new inputs to proceed with. We have to

```
***IntervalCleaning***$(G', G)$
  1. If $|V(G)| > |V(G')|$ then do:
         for each $s \in$ *NecessarySet*$(G', G)$ do:
             if *IntervalCleaning*$(G', G - s)$ return 'Yes', then return 'Yes'.
  2. Otherwise, if $G'$ is isomorphic to $G$ then return 'Yes'.
  3. Return 'No'.

***NecessarySet***$(G', G)$
  1. Set $N = \emptyset$.
  2. Call $\mathcal{A}(G', G)$.
     If it returns a reduced input $(H', H)$, then $N :=$ *NecessarySet*$(H', H)$.
     If it branches, then
         for each necessary set $X$ returned in a branch:
             set $N := N \cup X$,
         for each independent subproblem $(H', H)$ returned in a branch:
             set $N := N \cup$ *NecessarySet*$(H', H)$.
  3. Return $N$.
```

Figure 4.3: Outline of algorithms *IntervalCleaning* and *NecessarySet*.

repeat this at most $k$ times, so the whole algorithm has running time $f_2(k)^k|I|^2 = k^{O(k^5)}|I|^2$, where $|I|$ is the size of the original input of the problem. We can state this in the following theorem:

**Theorem 4.3.1.** Cleaning*(Interval, Interval) on input* $(G', G)$ *can be solved in* $k^{O(k^5)}n^2$ *time, where* $|V(G')| = n$ *and* $|V(G)| = n + k$.

## 4.3.1 Some structural observations

A nonempty set $M \subseteq V(G)$ is a *module* of $G$, if for every $x \in V(G) \setminus M$, $N_G(x)$ either includes $M$ or is disjoint from $M$. A module $M$ in $G$ is *complete*, if $G[M]$ is connected and there is no vertex in $x \in N_G(M)$ such that $N_G(x) \subseteq N_G[M]$. Lemma 4.3.2 gives a characterization of the complete modules of an interval graph. For an illustration, see Figure 4.1. Note that $\{a_1\}$ and $\{a_2, a_3\}$ are modules of $G$ that are not complete. The sets $\{a_1, a_2, a_3\}$, $\{b_1, b_2\}$ and $\{c_1, c_2, c_3, c_4, d_1, d_2, d_3, d_4, e_1, e_2\}$ are examples of complete module characterized by (a) of Lemma 4.3.2, and the set $\{e_1, e_2\}$ illustrates the complete modules characterized by (b) of Lemma 4.3.2.

**Lemma 4.3.2.** *Given an interval graph* $G$ *and a labeled PQ-tree* $T$ *representing* $G$*, some set* $M \subseteq V(G)$ *is a complete module of* $G$*, if and only if one of the following statements holds:*
*(a)* $M = R^{-1}(T_z)$ *for some* $z \in V(T)$*, and if* $z$ *is a P-node then* $R^{-1}(z) \neq \emptyset$
*(b)* $M = L_q(a, b)$ *for some Q-node* $q \in V(T)$ *having children* $x_1, \ldots, x_m$ *and some pair* $(a, b)$

*with $a < b$, such that $R^{-1}(T_{x_i}) = \emptyset$ for each $i$ contained in $[a,b]$, and $L_q(a',b') = \emptyset$ for each $[a',b']$ properly contained in $[a,b]$.*

*Proof.* First, let $M$ be a complete module in $G$. Let us choose a vertex $v \in M$ such that $R(v)$ is the closest possible to the root of $T$. Since $G[M]$ is connected, $v$ is unique, and we also get $R^{-1}(T_{R(v)}) \supseteq M$. First, suppose that $R(v)$ is a P-node or a leaf. Then $v$ is contained in each clique of $F(T_{R(v)})$. Thus, if $R(x)$ is in $T_{R(v)}$ for some vertex $x$, then $N_G(x) \subseteq N_G(v) \subseteq N_G[M]$. By the completeness of $M$, we get $x \in M$. Hence, $R^{-1}(T_{R(v)}) \subseteq M$ implying $R^{-1}(T_{R(v)}) = M$. Therefore, (a) holds in this case.

Now, suppose that $R(v)$ is a Q-node $q$ with children $x_1, \dots, x_m$, and let $M_q = M \cap R^{-1}(q)$. Let $a = \min\{Q_q^{\text{left}}(w) \mid w \in M_q\}$ and $b = \max\{Q_q^{\text{right}}(w) \mid w \in M_q\}$ Using the completeness of $M$, we can argue again that $R^{-1}(T_{x_h}) \subseteq M$ for each $h$ contained in $[a,b]$ and that $w \in M$ holds for each $w \in R^{-1}(q)$ such that $Q_q(w)$ is contained in $[a,b]$. Thus, if $[a,b] = [1,m]$ then $M = R^{-1}(T_q)$, implying that (a) holds. Otherwise, as $q$ is a Q-node, there must exist a vertex $u \in R^{-1}(q) \setminus M$ such that $Q_q(u)$ properly intersects $[a,b]$. As $u$ must be adjacent to each vertex of $M$ (as $M$ is a module), we get that $R^{-1}(T_{x_h}) = \emptyset$ for every $h$ in $[a,b]$ that is not contained in $Q_q(u)$. In particular, we get that either $R^{-1}(T_{x_a}) = \emptyset$ or $R^{-1}(T_{x_b}) = \emptyset$. We can assume w.l.o.g. that $R^{-1}(T_{x_a}) = \emptyset$ holds. Thus, $M_q^-(a) \neq \emptyset$, and since $M_q^-(a) \cap M \neq \emptyset$, using again that $M$ is a module, we obtain that each $w \in M_q$ must start in $a$ and also that $R^{-1}(T_{x_h}) = \emptyset$ for every $h$ in $[a,b]$. Note that this implies $M_q = M$. Now, from $R^{-1}(T_{x_b}) = \emptyset$ we get in a similar way that each $w \in M$ must end in $b$, proving $Q_q(w) = [a,b]$ for every $v, w \in M$. Now, using the completeness of $M$ and putting together these facts, we get that the conditions of (b) must hold.

For the other direction, it is easy to see that if (a) holds for some $M$, then $M$ indeed must be a complete module of $G$. Second, if $M = L_q(a,b)$ for some $q$ and $[a,b]$, then $M$ is clearly a module, and the remaining conditions of (b) ensure that $M$ is complete. □

We will say that a complete module $M$ is *simple*, if the conditions in (b) hold for $M$. Clearly, $N_G(M)$ is a clique if and only if $M$ is not simple, and if $M$ is simple then $G[M]$ is a clique. In Figure 4.1, $\{e_1, e_2\}$ is a simple complete module.

For a graph $H$, some set $M \subseteq V(G)$ is *an occurrence of $H$ in $G$ as a complete module*, if $M$ is a complete module for which $G[M]$ is isomorphic to $H$. Let $\mathcal{M}(H,G)$ be the set of the occurrences of $H$ in $G$ as a complete module. Using that each element of $\mathcal{M}(H,G)$ is a subset of $V(G)$ having size $|V(H)|$, we obtain the following consequence of Lemma 4.3.2.

**Proposition 4.3.3.** *For a graph $H$, the elements $\mathcal{M}(H,G)$ are pairwise disjoint.*

Moreover, if the graph $H$ is not a clique, then none of the occurrences of $H$ in $G$ as a complete module can be simple, so each set in $\mathcal{M}(H,G)$ must be of the form $R^{-1}(T_z)$ for some non-leaf node $z$ of $T$. This yields that the sets in $\mathcal{M}(H,G)$ are independent (where two vertex sets in a graph are independent if there is no edge between them). Lemma 4.3.4 below states some observations about what happens to a set of disjoint and independent complete modules in a graph after adding or deleting a vertex.

**Lemma 4.3.4.** *Suppose that $s \in V(G)$.*
*(1) If $M_1, \dots, M_\ell$ are disjoint independent complete modules in $G - s$, then $M_i$ is a complete module in $G$ for at least $\ell - 4$ indices $i \in [\ell]$.*
*(2) If $M_1, \dots, M_\ell$ are disjoint independent complete modules in $G$, then $M_i$ is a complete module in $G - s$ for at least $\ell - 4$ indices $i \in [\ell]$.*

*Proof.* As $M_i$ and $M_j$ are independent if $i \neq j$, we can assume that $M_1^{\text{left}} \leq M_1^{\text{right}} < \cdots < M_\ell^{\text{left}} \leq M_\ell^{\text{right}}$. Recall that each $M_i$ is connected by the definition of a complete module. We

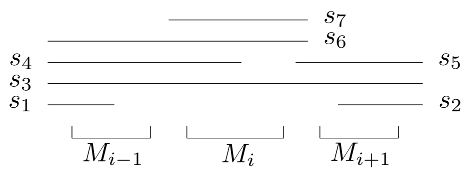


Figure 4.4: $M_{i-1}$, $M_i$, and $M_{i+1}$ illustrate complete modules of $G - S$. The set $M_i$ is untouched by $s_1$, $s_2$, and $s_3$, but this is not true for any vertex $s_j$, $j \geq 4$.

say that $M_i$ is *untouched* (by $s$), if either $s^{\text{left}} \leq M_{i-1}^{\text{right}}$ and $s^{\text{right}} \geq M_{i+1}^{\text{left}}$, or $s^{\text{right}} < M_{i-1}^{\text{right}}$, or $s^{\text{left}} > M_{i+1}^{\text{left}}$. (See also Figure 4.4.) If $M_a$ and $M_b$ are the first and the last one, respectively, among the sets $M_1, \ldots, M_\ell$ that have a vertex adjacent to $s$, then each $M_i$ except for $M_{a-1}$, $M_a$, $M_b$, and $M_{b+1}$ must be untouched by $s$.

To see (1), we show that if a complete module of $G - s$ is untouched, then it is a complete module of $G$. So assume that $M_i$ is untouched. Clearly, $s \notin M_i$. Since either $N_G(s) \supseteq M_i$ or $N_G(s) \cap M_i = \emptyset$, $M_i$ remains to be a module in $G$. Also, if $s \in N_G(M_i)$, then $s$ must have a neighbor in $M_{i-1}$ and $M_{i+1}$. Thus, $N_G(s) \nsubseteq N_G[M_i]$, so the completeness of $M_i$ in $G - s$ implies its completeness in $G$ as well.

To prove (2), suppose that $M_i$ is an untouched complete module in $G$. Clearly, $M_i$ is a module in $G - s$ as well, and since $s \notin M_i$, $M_i$ remains connected in $G - s$. Let $x$ be a vertex in $N_G(M_i)$. By the completeness of $M_i$, $x$ is adjacent to some vertex $y \notin N_G[M_i]$. Suppose that $x$ does not have a neighbor outside $N_{G-s}[M_i]$ in $G - s$. This can only happen if $y = s$. Now, since $y \notin N_G[M_i]$ and $M_i$ is untouched by $s$, $x$ must also be adjacent to a vertex of $M_{i-1}$ or $M_{i+1}$. Thus, $x$ has a neighbor in $V(G - s) \setminus N_{G-s}[M_i]$, proving the completeness of $M_i$. As $M_i$ is untouched for at least $\ell - 4$ indices $i \in [\ell]$, the statement follows. ☐

In the case when $H$ is a clique and $K$ is an occurrence of $H$ in $G$ as a complete module, we get that either $K = R^{-1}(\ell)$ for some leaf $\ell \in V(T)$, or $K$ is simple, i.e. $K = L_q(a, b)$ for some Q-node $q \in V(T)$ and some block $[a, b]$. In the latter case, Lemma 4.3.5 states a useful observation about the block $[a, b]$. This lemma uses the following definition: we say that a complete module $K$ of $G$ is *h-short*, if either $K = R^{-1}(\ell)$ for some leaf $\ell \in V(T)$, or $K = L_q(a, b)$ for some Q-node $q \in V(T)$ and some block $[a, b]$ with $b - a \leq h$. The sets $\{e_1, e_2\}$ and $\{b_1, b_2\}$ are 2-short complete modules of $G$ in Figure 4.1.

**Lemma 4.3.5.** *If $K$ is a complete module in $G$ such that $G[K]$ is a clique but $K$ is not h-short, then $|N_G(K)| \geq 2(h + 1)$.*

*Proof.* By the conditions of the lemma, we know that $K = L_q(a, b)$ for some Q-node $q \in V(T)$ with children $x_1, \ldots, x_m$ and some block $[a, b]$ such that $b - a \geq h + 1$. By the completeness of $K$, we get that $R^{-1}(T_{x_h}) = \emptyset$ for any $h$ contained in $[a, b]$, so $M^+(h)$ and $M^-(h)$ cannot be empty. Taking these sets for all $h$ in $[a, b]$, with the exception of the sets $M^+(a)$ and $M^-(b)$, we get $2(b - a) \geq 2(h + 1)$ nonempty sets that are pairwise disjoint, each containing some vertex of $N_G(K)$. This implies the bound $N_G(K) \geq 2(h + 1)$. ☐

Observe that if two different $h$-short complete modules $K_1$ and $K_2$ in $G$ are not independent, then $K_1 = L_q(a, b)$ and $K_2 = L_q(c, d)$ must hold for some Q-node $q$ in $T$ and some blocks $[a, b]$ and $[c, d]$ that properly intersect each other. Now, if $b - a \leq h$, then there can be at most $2h$ such blocks $[c, d]$ for which these conditions hold. This implies that given a $h$-short complete module $K$, there can be at most $2h$ different $h$-short complete modules of $G$ neighboring $K$ (but not equal to $K$). It is also easy to see that the maximum number of pairwise neighboring $h$-short complete modules in a graph is at most $h+1$. Making use of these facts, Lemma 4.3.6 states some results about $h$-short complete modules of a graph in a similar fashion as Lemma 4.3.4. As opposed to Lemma 4.3.4, here we do not require the complete modules to be independent.

**Lemma 4.3.6.** *Suppose that $s \in V(G)$.*
*(1) If $M_1, \ldots, M_\ell$ are disjoint $h$-short complete modules in $G-s$, then $M_i$ is a complete module in $G$ for at least $\ell-(3h+5)$ indices $i \in [\ell]$.*
*(2) If $M_1, \ldots, M_\ell$ are disjoint $h$-short complete modules in $G$, then $M_i$ is a complete module in $G-s$ for at least $\ell-(4h+3)$ indices $i \in [\ell]$.*

*Proof.* The proof relies on the observation that there can only be a few indices $i$ such that either $s^{\text{left}}$ or $s^{\text{right}}$ lies within $[M_i^{\text{left}}, M_i^{\text{right}}]$.

To see (1), suppose that $M_i$ is not a $h$-short complete module in $G$ for some $i$. Clearly, $G[M_i]$ is connected. First, assume that $M_i$ is not a module because there are some $x, y \in M_i$ such that $s$ is adjacent to $x$ but not to $y$. In this case, either $x^{\text{left}} < s^{\text{right}} < y^{\text{left}}$ or $y^{\text{right}} < s^{\text{left}} < x^{\text{right}}$. It is not hard to see that this implies that there can be at most two such modules $M_i$. Now, assume that $M_i$ is a module, but is not complete. This implies that $M_i \subseteq N_G(s) \subseteq N_G[M_i]$ is true. Note that if $j \neq i$ then $M_j \subseteq N_G(s) \subseteq N_G[M_j]$ is only possible if $M_i$ and $M_j$ are neighboring. Thus, there can be at most $h+1$ such indices $i$.

Finally, if $M_i$ is complete module in $G$ but it is not $h$-short, then the number of maximal cliques containing the vertices of $M_i$ must be more in $G$ than in $G-s$, implying that either $M_i^{\text{left}} < s^{\text{left}} \leq M_i^{\text{right}}$ or $M_i^{\text{left}} \leq s^{\text{right}} \leq M_i^{\text{right}}$. As $M_i^{\text{left}} < s^{\text{left}} \leq M_i^{\text{right}}$ and $M_j^{\text{left}} < s^{\text{left}} \leq M_j^{\text{right}}$ can only hold simultaneously if $M_i$ and $M_j$ are neighboring, and such a statement is also true for the latter condition, we get that there can be at most $2(h+1)$ indices $i$ for which $M_i$ is $h$-short in $G-s$ but not in $G$. Summing up these facts, we obtain that there can be at most $2+(h+1)+2(h+1) = 3h+5$ indices $i$ for which $M_i$ is not a $h$-short complete module in $G$.

To prove (2), notice that each $M_i$ remains a module in $G-s$ as well. Observe also that if $s \notin M_i$, then $M_i$ remains connected in $G-s$. By the disjointness of the sets $M_1, \ldots, M_\ell$, each of them is connected in $G-s$ except for at most one. Suppose that $M_{i_1}$, $M_{i_2}$, and $M_{i_3}$ are independent, and for each $j \in \{1,2,3\}$, $M_{i_j}$ is a connected module in $G-s$ but it is not complete. This means that there are vertices $x_1$, $x_2$, and $x_3$ such that $x_j \in N_G(M_{i_j})$, but $N_G(x_j) \subseteq N_G[M_{i_j}]$ for each $j$. By the completeness of these modules in $G$, this implies that each of $x_1$, $x_2$, and $x_3$ are adjacent to $s$, and $s \notin N_G[M_{i_j}]$ for any $j$. But this can only hold if some $x_j$ is adjacent to each vertex of $M_{i_{j'}}$ for some $j \neq j'$, and since $M_{i_j}$ and $M_{i_{j'}}$ are independent, this contradicts the assumption that $N_G(x_j) \subseteq N_G[M_{i_j}]$. Thus, there cannot exist such indices $i_1$, $i_2$ and $i_3$, implying that we can fix two indices $j$ and $j'$ such that for any $M_i$ that is a connected module in $G-s$ but not complete, $M_i$ is neighboring either $M_j$ or $M_{j'}$, implying that there can be at most $2(2h)+2$ such indices $i$. To finish, observe that if $M_i$ is a complete module in $G-s$, then it must be $h$-short, as the deletion of $s$ cannot increase the number of maximal cliques that contain $M_i$. □

### 4.3.2 Reduction rules

In this section, we introduce some reduction rules, each of which can be applied in linear time, and provides a necessary set, an independent subproblem, or a reduced input, as described earlier. Our aim is to handle all cases except for the case when both $G$ and $G'$ have a PQ-tree with a Q-node root. We always apply the first possible reduction. From now on, we assume that $S$ is a solution for $(G', G)$ and $\phi_S$ is an isomorphism from $G'$ to $G-S$.

**Rule 1. Isomorphic components.** Lemma 4.3.7 yields a simple reduction: if $G$ and $G'$ have isomorphic components, then algorithm $\mathcal{A}$ can output a **reduced input** of $(G', G)$.

Note that partitioning a set of interval graphs into isomorphism equivalence classes can be done in linear time [96] (see also [42, 135, 136]). Hence, this reduction can also be performed in linear time.

**Lemma 4.3.7.** *If $K$ and $K'$ are connected components of $G$ and $G'$, respectively, and $K$ is isomorphic to $K'$, then $(G'-K', G-K)$ is a reduced input of $(G', G)$.*

*Proof.* Trivially, $G'-K'$ has fewer vertices than $G'$, and any solution for $(G'-K', G-K)$ is a solution for $(G', G)$ as well, by the isomorphism of $K'$ and $K$. Therefore, we only have to prove that if $(G', G)$ is solvable then $(G'-K', G-K)$ is also solvable. Clearly, if $S \cap V(K) = \emptyset$, then we can assume w.l.o.g. that $\phi_S(K') = K$. In this case, $S$ is a solution for $(G'-K', G-K)$.

Now, if $S \cap V(K) \neq \emptyset$ then $K$ and $\phi_S(K')$ are disjoint. Moreover, $K$ and $\phi_S(K')$ are disjoint isomorphic connected components of $G - S_0$ where $S_0 = S \setminus V(K)$. Let $\kappa$ be an isomorphism from $K$ to $\phi_S(K')$. Notice that the role of $K$ and $\phi_S(K')$ can be interchanged, and we can replace $S \cap V(K)$ with $\kappa(S \cap V(K))$ in the solution. Thus, $S_0 \cup \kappa(S \cap V(K))$ is a solution for $(G', G)$ that is disjoint from $K$. Since this yields a solution for $(G'-K', G-K)$ as well, this finishes the proof. ☐

**Rule 2. Many components in $G'$.** This reduction is possible in the case when $G'$ has at least $4k+1$ components. Since Rule 1 cannot be applied, none of the components of $G$ is isomorphic to a component of $G'$. Our aim is to locate $\phi_S(K')$ in $G$ for one of the components $K'$ of $G'$. If we find $\phi_S(K')$ then we know that $N_G(\phi_S(K'))$ must be contained in $S$, so we can produce a **necessary set** of size 1 by outputting any of the vertices of $N_G(\phi_S(K'))$.

Given a graph $H$, recall that $\mathcal{M}(H, G)$ denotes the occurrences of $H$ in $G$ as a complete module. By Proposition 4.3.3, the elements of $\mathcal{M}(H, G)$ are disjoint subsets of $V(G)$. We can find $\mathcal{M}(H, G)$ in linear time, using the labeled PQ-tree of $G$ and the characterization of Lemma 4.3.2.

Relying on Lemmas 4.3.4 and 4.3.6, the algorithm performs the following reduction. Suppose that $K'_1, K'_2, \dots, K'_{k'}$ are the $k' = 4k+1$ largest connected components of $G'$, ordered decreasingly by their size, and let $S$ be a solution for $(G', G)$. As the vertex sets of the connected components of $G'$ are complete modules of $G'$, the sets $K_i = \phi_S(V(K'_i))$ for $i \in [k']$ are complete modules of $G - S$. By definition, these sets are also disjoint and independent. As a consequence of (1) in Lemma 4.3.4, we get that for at least $k' - 4k = 1$ indices $i \in [k']$ the set $K_i$ will be a complete module of $G$. We branch on the choice of $i$ to find such a set $K_i$, resulting in at most $k'$ possibilities. Observe that w.l.o.g. we can assume that the subgraph $G[K_i]$ is the first one (according to the given representation of $G$) among the components of $G - S$ isomorphic to $K'_i$.

It remains to describe how we can find $K_i$ in $G$. To begin, we suppose now that $K'_i$ is not a clique. Let us discuss a simplified case first, where we assume that $K'_i$ is not contained as an induced subgraph in any of the components $K'_j$ if $j \neq i$. Let $\mathcal{M}(K'_i, G) = \{A_1, A_2, \dots\}$, where the sets in $\mathcal{M}(K'_i, G)$ are ordered according to their order in the interval representation of $G$. Let $i^*$ denote the index for which $A_{i^*}$ is the first element in $\mathcal{M}(K'_i, G)$ that is a complete module in $G - S$ as well. Since $K'_i$ is not contained in a component of $G'$ having more vertices than $|V(K'_i)|$, $G[A_{i^*}]$ must be a connected component of $G - S$. Also, $G[A_{i^*}]$ is isomorphic to $K'_i$, and by the definition of $A_{i^*}$, it must be the first such component of $G - S$. Thus, we can conclude that $A_{i^*}$ equals $K_i$. By (2) of Lemma 4.3.4, there can be at most $4k$ sets in $\mathcal{M}(K'_i, G)$ that are not complete modules in $G - S$, so we get that $i^* \leq 4k+1 = k'$. Hence, we can find $K_i$ by guessing $i^*$ and branching into $k'$ directions.

Let us consider now the general case, where some of the components $K'_j$ can contain $K'_i$ in $G'$. (We still suppose that $K_i$ is not a clique.) For each $j < i$, we define an indicator variable $\delta(j)$ which has value 1 if and only if $K_j$ precedes $K_i$ in $G - S$. We guess $\delta(j)$ for each $j \in [i-1]$, which means at most $2^{k'-1}$ possibilities.

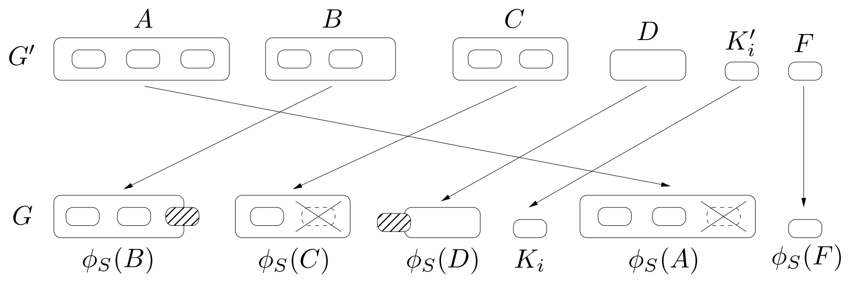


Figure 4.5: An illustration of Rule 2. In this example, the small rectangles denote elements of $\mathcal{M}(K'_i, G')$ and $\mathcal{M}(K'_i, G)$. Rectangles with a skew pattern are elements of $\mathcal{M}(K'_i, G)$ that are not complete modules of $G - S$. Crossed rectangles with a dashed border indicate if some set $\phi_S(X)$ is not a complete module of $G$ for some $X \in \mathcal{M}(K'_i, G')$. In this example, $i = 5$, $\delta(1) = 0, \delta(2) = \delta(3) = \delta(4) = 1$, $|\mathcal{M}'| = 4$ and $i^* = 6$.

Again, let $\mathcal{M}(K'_i, G) = \{A_1, A_2, \dots\}$, where the sets $A_h$ are ordered according to their order in the interval representation of $G$. Let $i^*$ denote the index for which $K_i = A_{i^*}$, and let $\mathcal{M}'$ stand for $\bigcup_{j<i,\delta(j)=1} \mathcal{M}(K'_i, K'_j)$, which is a collection of subsets of $V(G')$, each inducing a subgraph of $G'$ isomorphic to $K'_i$. As $K_i$ is not a clique, the elements of $\mathcal{M}'$ in $G'$ are disjoint and independent, so by (1) of Lemma 4.3.4 we get that for at least $|\mathcal{M}'| - 4k$ sets $A \in \mathcal{M}'$, the set $\phi_S(A)$ will be a complete module of $G$ as well. As all these sets precede $K_i$ in $G$, we get that $\phi_S(A) \in \{A_1, \dots, A_{i^*-1}\}$ holds for at least $|\mathcal{M}'| - 4k$ sets $A \in \mathcal{M}'$. From this, $i^* - 1 \geq |\mathcal{M}'| - 4k$ follows.

Clearly, for all those sets $A \in \{A_1, \dots, A_{i^*-1}\}$ which are complete modules in $G - S$ as well, $\phi_S^{-1}(A)$ must be contained in a component of $G'$ which is larger than $K'_i$. Here we used again the assumption that $G[K_i]$ is the first one among the components of $G - S$ isomorphic to $K_i$. Since such an $A$ precedes $K_i$, we obtain $\phi_S^{-1}(A) \in \mathcal{M}'$. By (2) of Lemma 4.3.4, there can be at most $4k$ sets among $A_1, \dots, A_{i^*-1}$ that are not complete modules in $G - S$, so we get that $\phi_S^{-1}(A) \in \mathcal{M}'$ for at least $i^* - 1 - 4k$ sets $A$ in $\{A_1, \dots, A_{i^*-1}\}$. This implies $i^* - 1 \leq |\mathcal{M}'| + 4k$. Altogether, we get the bounds $|\mathcal{M}'| - 4k + 1 \leq i^* \leq |\mathcal{M}'| + 4k + 1$. Since $|\mathcal{M}'|$ can be determined in linear time, by branching on the at most $8k + 1$ possibilities to choose $i^*$, we can find the vertex set $K_i$.

Now, we suppose that $K'_i$ is a clique. As $K_i$ is a component of $G - S$, $|N_G(K_i)| \leq k$, which by Lemma 4.3.5 implies that $K_i$ must be $k/2$-short. Using Lemma 4.3.6, we can find $K_i$ in a similar manner to the previous case. We denote by $\mathcal{N}(H, G)$ the occurrences of a graph $H$ in $G$ as a $k/2$-short complete module. Analogously to the previous case, let $\mathcal{N}(K'_i, G) = \{B_1, B_2, \dots\}$, where the sets in $\mathcal{N}(K'_i, G)$ are ordered according to their order in the fixed representation of $G$. We also let $K_i = B_{i^*}$ and $\mathcal{N}' = \bigcup_{j<i,\delta(j)=1} \mathcal{N}(K'_i, K'_j)$. Now, using Lemma 4.3.6 just as in the reasoning above, we get the bounds $|\mathcal{N}'| - k(3k/2 + 5) \leq i^* - 1 \leq |\mathcal{N}'| + k(2k + 3)$. Again, $|\mathcal{N}'|$ can be determined in linear time, so by branching on the at most $k(7k/2 + 8) + 1$ possibilities to choose $i^*$, we can find the vertex set $K_i$.

Since Rule 1 cannot be applied, none of the components of $G$ can be isomorphic to a component of $G'$, hence $S_i = N_G(K_i)$ is not empty. Clearly $S_i \subseteq S$, so we get that $\{s\}$ is a necessary set for any $s \in S_i$. The total number of possible branches in this reduction is at most $(4k + 1)2^{4k}(k(7k/2 + 8) + 1) = 2^{O(k)}$.

**Rule 3. Disconnected $G$.** Here we give a reduction for the case when $G$ is not connected, but the previous reductions cannot be performed. First, observe that each component of $G$ contains at least one vertex from $S$, as none of them is isomorphic to a component of $G'$.

Thus, if $G$ has more than $k$ components then there cannot exist a solution of size $k$, so we can **reject**. Otherwise, let us fix an arbitrary component $K$ of $G$. We branch on the choice of those components of $G'$ whose vertices are in $\phi_S^{-1}(K - S)$, for some fixed solution $S$. Let the union of these components be $G'_K$. Note that guessing $G'_K$ yields at most $2^{4k}$ possibilities, since $G'$ has at most $4k$ components. By our assumptions, $1 \leq k' < k$ holds for the parameter $k'$ of the instance $(G'_K, K)$, so $(G'_K, K)$ is clearly an **independent subproblem** of $(G', G)$.

**Rule 4. Universal vertex in $G$.** A vertex $x$ is *universal* in $G$, if $N_G(x) = V(G - x)$. Such vertices imply a simple reduction by Lemma 4.3.8 which allows $\mathcal{A}$ to output either a **necessary set** of size 1 or a **reduced input** of $(G', G)$.

**Lemma 4.3.8.** *Let $x$ be universal in $G$. If there is no universal vertex in $G'$, then $\{x\}$ is a necessary set for $(G', G)$. If $x'$ is universal in $G'$, then $(G' - x', G - x)$ is a reduced input of $(G', G)$.*

*Proof.* Clearly, if $x$ is universal in $G$ and $x \notin S$ for a solution $S$, then it remains universal in $G - S$. Thus, if no vertex is universal in $G'$, then $x \in S$ must hold.

Suppose $x'$ is universal in $G'$, and $S$ is an arbitrary solution. Clearly, if $x \notin S$, then $x$ and $y = \phi_S(x')$ are both universal in $G - S$, so if $x \neq y$ then we can swap the role of $x$ and $y$ such that $\phi_S$ maps $x'$ to $x$. Now, if $x \in S$ then $S' = S \cup \{y\} \setminus \{x\}$ is a solution in which the isomorphism from $G'$ to $G - S'$ can map $x'$ to $x$. This implies that $(G' - x', G - x)$ is a reduced input of $(G', G)$. □

**Rule 5. Disconnected $G'$.** Suppose that none of the previous reductions can be applied, and $G'$ is disconnected. Clearly, $G$ must be connected, and $G'$ has at most $4k$ components. Let $S$ be a solution. For each component $K'$ in $G'$, let $I(K')$ be the union of the intervals representing $\phi_S(K')$ in the fixed representation of $G$, i.e. $I(K') = [\phi_S(V(K'))^{\text{left}}, \phi_S(V(K'))^{\text{right}}]$. Since the components of $G'$ are connected and independent, the intervals $I(K'_1)$ and $I(K'_2)$ are disjoint for two different components $K'_1$ and $K'_2$ of $G'$.

Let $Q$ be the component of $G'$ such that $I(Q)$ is the first among the intervals $\{I(K') \mid K'$ is a component of $G'\}$. Clearly, if $x^{\text{right}} \leq \phi_S(V(Q))^{\text{right}}$ for some vertex $x \in V(G)$, then either $x \in S$ or $x \in \phi_S(Q)$, thus the number $s_Q$ of such vertices is at least $|V(Q)|$ but at most $|V(Q)| + k$. Therefore, we first guess $Q$, and then we guess the value of $s_Q$, which yields at most $4k(k+1)$ possibilities. Now, ordering the vertices of $G$ such that $x$ precedes $y$ if $x^{\text{right}} < y^{\text{right}}$ and putting the first $s_Q$ vertices in this ordering into a set $B$, we get $\phi_S(Q) \subseteq B \subseteq \phi_S(Q) \cup S$. Since $G$ is connected, there must exist an edge $e = xy$ running between $B$ and $V(G) \setminus B$. Clearly, at least one endpoint of $e$ must be in $S$, thus we can output the **necessary set** $\{x, y\}$.

**Rule 6. Universal vertex in $G'$.** Suppose that some vertex $x'$ is universal in $G'$. Let $a$ and $b$ be vertices of $G$ defined such that $a^{\text{right}} = \min\{x^{\text{right}} \mid x \in V(G)\}$ and $b^{\text{left}} = \max\{x^{\text{left}} \mid x \in V(G)\}$. As there is no universal vertex in $G$, we know that $x^{\text{left}} > a^{\text{right}}$ or $x^{\text{right}} < b^{\text{left}}$ for each $x \in V(G)$, i.e. no vertex in $G$ is adjacent to both $a$ and $b$. As $\phi_S(x')$ is universal in $G - S$ for any fixed solution $S$, we get that $\{a, b\}$ is a **necessary set**.

## 4.4 The Q-Q case

From now on, we assume that none of the reductions given in Section 4.3.2 can be applied. Thus, $G$ and $G'$ are connected, and none of them contains universal vertices, so in particular, none of them can be a clique. This implies that if $r$ and $r'$ is the root of $T$ and $T'$, respectively, then both $r$ and $r'$ are Q-nodes. Let $m$ and $m'$ denote the number of the children of $r$ and $r'$, respectively. When indexing elements of $[m]$ and $[m']$, we will try to use $i$ and $j$, respectively, whenever it makes sense. Let $x_i$ and $x'_j$ denote the $i$-th and $j$-th child of $r$ and $r'$, respectively, and let $X_i = R^{-1}(T_{x_i})$ and $X'_j = R^{-1}(T'_{x'_j})$, for all $i \in [m]$ and $j \in [m']$.

Let us call a solution $S$ *local*, if there is an $i \in [m]$ such that $S \supseteq V(G) \setminus N_G[X_i]$, i.e. $S$ contains every vertex of $G$ except for the closed neighborhood of some $X_i$. Suppose that $S$ is a solution that is not local, and $\phi_S$ is an isomorphism from $G'$ to $G - S$. The following definitions try to give a bound on those indices $i$ in $[m]$ which somehow contribute to $\phi_S(X'_j)$ for some $j \in [m']$. For an index $j \in [m']$, let:

$$\begin{aligned}\alpha_S(j) &= \min\{Q_r^{\text{left}}(\phi_S(v)) \mid v \in X'_j \cup M^+_{r'}(j)\}\\ \beta_S(j) &= \max\{Q_r^{\text{right}}(\phi_S(v)) \mid v \in X'_j \cup M^-_{r'}(j)\}.\end{aligned}$$

Observe that by

$$\min\{Q_r^{\text{left}}(\phi_S(v)) \mid v \in X'_j \cup M^+_{r'}(j)\} \le \min\{Q_r^{\text{left}}(\phi_S(v)) \mid v \in X'_j\}$$

$$\le \max\{Q_r^{\text{right}}(\phi_S(v)) \mid v \in X'_j\} \le \max\{Q_r^{\text{right}}(\phi_S(v)) \mid v \in X'_j \cup M^-_{r'}(j)\}$$

we obtain that $\alpha_S(j) \le \beta_S(j)$ holds for any $j \in [m']$. We let $I_S(j)$ be the block $[\alpha_S(j), \beta_S(j)]$.

The following lemma summarizes some useful observations.

**Lemma 4.4.1.** *Suppose that $S$ is a solution for $(G', G)$ that is not local. Then either all of the following statements hold, or all of them hold after reversing the children of $r'$:*
*(1) $Q_{r'}^{\text{dir}_1}(v) < Q_{r'}^{\text{dir}_2}(w)$ for some $v, w \in V(G')$ and $\text{dir}_1, \text{dir}_2 \in \{\text{left}, \text{right}\}$ implies that $Q_r^{\text{dir}_1}(\phi_S(v)) < Q_r^{\text{dir}_2}(\phi_S(w))$ holds as well.*
*(2) For any $j_1 < j_2$ in $[m']$, the block $I_S(j_1)$ precedes $I_S(j_2)$.*
*(3) If $m = m'$, then $I_S(j) = [j, j]$ for each $j \in [m']$.*
*(4) If $i \in [m]$ then $X_i \setminus S$ is contained in $\phi_S(X'_j)$ for some $j \in [m']$.*
*(5) $\phi_S(R^{-1}(r')) \subseteq R^{-1}(r)$.*

*Proof.* Let the sets $A_1, \dots, A_s$ be disjoint subsets of some set $\{a_i | i \in [r]\}$. We say that the series $a_1, \dots, a_r$, *respects* the series $A_1, \dots, A_s$ if for any $p$, the elements of $A_p$ precede the elements of $A_{p+1}$ in $a_1, \dots, a_r$.

For some $j \in [m']$, let $C'_j$ be the set of maximal cliques of $G'$ contained in $F(T'_{x'_j})$. As $T'$ represents $G'$ and its root is a Q-node, any consecutive ordering of the maximal cliques of $G'$ respects either $C'_1, C'_2, \dots, C'_{m'}$ or $C'_{m'}, C'_{m'-1}, \dots, C'_1$.

Let $C_j$ be the set of maximal cliques of $G$ that contain $\phi_S(K)$ for some $K \in C'_j$. Clearly, the sets $C_j$ ($j \in [m']$) are disjoint by the maximality of the cliques in $F(T')$. The interval representation of $G$ yields an interval representation of $G - S$ and hence of $G'$, which implies that any consecutive ordering of the cliques in $F(T)$ must respect either $C_1, \dots, C_{m'}$ or $C_{m'}, \dots, C_1$. This implies that if $j_1 \ne j_2$, then the deepest node in $T$ whose frontier contains every clique in $C_{j_1} \cup C_{j_2}$ must be some unique Q-node $q$, the same for each pair $(j_1, j_2)$. Note also that if $q$ is contained in $T_{x_i}$ for some $i \in [m]$, then every vertex of $\phi_s(V(G'))$ must be contained in some element of $F(T_{x_i})$. This yields that $S \supseteq V(G) \setminus N_G[X_i]$, contradicting to the assumption that $S$ is not local. Hence $q = r$. The definition of $q$ shows that by possibly reversing the children of $r$ in $T$, we can find disjoint blocks $B_1, B_2, \dots, B_{m'}$ following each other in this order in [1,m] such that for each $j \in [m']$ every clique of $C_j$ is contained in $\bigcup_{i \in B_j} F(T_{x_i})$.

This observation can be easily seen to imply claim (1), by simply recalling what $Q_{r'}^{\text{dir}_1}(v) < Q_{r'}^{\text{dir}_2}(w)$ means by definition for some $v, w \in V(G')$ and $\text{dir}_1, \text{dir}_2 \in \{\text{left}, \text{right}\}$, considering the maximal cliques of $G'$.

Now, it is easy to prove (2), (3), (4), and (5), using the assumption that (1) holds (which can indeed be achieved by possibly reversing the children of $r'$). To prove (2), observe that for any $j \in [m']$, we have that $Q^{\text{left}}_{r'}(v) = j$ for each $v \in X'_j \cup M^+_{r'}(j)$ and $Q^{\text{right}}_{r'}(v) = j$ for each $v \in X'_j \cup M^-_{r'}(j)$. Thus, for any $j_1 < j_2$ in $[m']$, $v \in X'_{j_1} \cup M^-_{r'}(j_1)$, and $w \in X'_{j_2} \cup M^+_{r'}(j_2)$ we obtain $Q^{\text{right}}_{r'}(v) < Q^{\text{left}}_{r'}(w)$, which immediately implies $Q^{\text{right}}_{r}(\phi_S(v)) < Q^{\text{left}}_{r}(\phi_S(w))$ by using the assumption that (1) holds. By definition, this means $\beta_S(j_1) < \alpha_S(j_2)$, from which (2) follows.

Note that (3) is directly implied by (2).

To see (4), consider an $x \in X_i \setminus S$ and let $x'$ be the vertex in $G'$ for which $x = \phi_S(x')$. Since $Q^{\text{left}}_{r}(x) = Q^{\text{right}}_{r}(x) = i$, by (1) we must have $Q^{\text{left}}_{r'}(x') = Q^{\text{right}}_{r'}(x')$ as well. Thus, $x' \in X'_j$ for some $j \in [m']$. Now, assuming that $\phi_S(y')$ is also in $X_i$ but $y' \notin X'_j$, we obtain that either $Q^{\text{right}}_{r'}(x') < Q^{\text{left}}_{r'}(y')$ or $Q^{\text{right}}_{r'}(y') < Q^{\text{left}}_{r'}(x')$. But using (1), these both contradict $\phi_S(y') \in X_i$.

Finally, (5) follows immediately from (4). □

Using Lemma 4.4.1, we can handle an easy case when no branching is needed, and a reduced input can be constructed. Let $L(r)$ and $L'(r')$ denote the labels of $r$ and $r'$ in $T$ and $T'$, respectively.

**Lemma 4.4.2.** *If $m = m'$, $L(r) = L'(r')$ and there is an $i \in [m]$ such that $G[X_j]$ is isomorphic to $G'[X'_j]$ for all $j \neq i$, then $(G'[X'_i], G[X_i])$ is a reduced input of $(G', G)$.*

*Proof.* First, we show that a solution $S$ for $(G'[X'_i], G[X_i])$ is a solution for $(G', G)$. Clearly, the conditions of the lemma imply that there is an isomorphism from $G' - X'_i$ to $G - X_i$ mapping $N_{G'}(X'_i)$ to $N_G(X_i)$. This isomorphism can be extended to map $X'_i$ to $X_i \setminus S$, since $S$ is a solution for $(G'[X'_i], G[X_i])$. In the other direction, suppose that $S$ is a solution for $(G', G)$. Note that $S$ need not be a solution for $(G'[X'_i], G[X_i])$, as $S$ may contain vertices not in $X_i$. However, to prove the lemma it suffices to show that a solution exists for $(G'[X'_i], G[X_i])$, meaning that $G'[X'_i]$ is isomorphic to an induced subgraph of $G[X_i]$.

Let us assume first that $S$ is not local. Suppose $S \cap X_j \neq \emptyset$ for some $j \neq i$. Since $G[X_j]$ is isomorphic to $G'[X'_j]$, we have $|\phi_S(X'_j)| = |X_j| > |X_j \setminus S|$. By Lemma 4.4.1, this implies that $I_S(j) = [j, j]$ becomes true only after reversing the children of $r'$, meaning that $I_S(h) = [m-h+1, m-h+1]$ for each $h \in [m]$ (according to the PQ-tree $T'$). In particular, this means that $\phi_S(X'_i) \subseteq X_{m-i+1}$. Hence, $G'[X'_i]$ is isomorphic to an induced subgraph of $G[X_{m-i+1}]$. Since $G[X_{m-i+1}]$ is isomorphic to $G'[X'_{m-i+1}]$, $G'[X'_i]$ is also isomorphic to an induced subgraph of $G'[X'_{m-i+1}]$. From this, by $\phi_S(X'_{m-i+1}) \subseteq X_i$ we get that $G'[X'_i]$ is isomorphic to an induced subgraph of $G[X_i]$.

Now, suppose that $S$ is a local solution, and $S \supseteq V(G) \setminus N_G[X_h]$ for some $h \in [m]$. Since each vertex of $R^{-1}(r) \setminus S$ must be adjacent to every vertex in $N_G[X_h]$, such vertices would be universal in $G - S$. Thus, as $G'$ contains no universal vertices, $S \supseteq R^{-1}(r)$ follows. Hence, we get $\phi_S(V(G')) \subseteq X_h$ and thus $|X_h| \geq |V(G')| > |X'_h|$, implying $h = i$. But $\phi_S(V(G')) \subseteq X_i$ clearly implies that $G'$ and therefore also $G'[X'_i]$ must be isomorphic to an induced subgraph of $G[X_i]$. This finishes the proof. □

Observe that it can be tested in linear time whether the conditions of Lemma 4.4.2 hold after possibly reversing the children of $r'$. If this is the case, algorithm $\mathcal{A}$ proceeds with the **reduced input** guaranteed by Lemma 4.4.2. Otherwise it branches into a few directions as follows. In each of these branches, $\mathcal{A}$ will output either a necessary set of size at most $2k+1$, or an independent subproblem of $(G', G)$.

In the first branch, it assumes that the solution $S$ is local. In this case, given any two vertices $a \in X_1$ and $b \in X_m$, a solution must include at least one of $a$ and $b$. (Note that $X_1$ and $X_m$ cannot be empty.) Thus, $\mathcal{A}$ outputs the **necessary set** $\{a, b\}$. In all other branches, we assume that $S$ is not local. Algorithm $\mathcal{A}$ branches into two more directions, according to whether the children of $r'$ have to be reversed to achieve the properties of Lemma 4.4.1. Thus, in the followings we may assume that these properties hold.

First, observe that Lemma 4.4.1 implies that $m \geq m'$ must hold, so otherwise the algorithm can **reject**. First, we examine the case $m = m'$, and then we deal with the case $m > m'$ in Section 4.4.1. Observe that Lemma 4.4.1 also implies that $\phi_S$ must map every $L_{r'}(a,b)$ to a vertex in $L_r(a,b)$, for any block $[a,b]$ in $[1,m]$. So, if there is a block $[a,b]$ such that $|L_r(a,b)| < |L_{r'}(a,b)|$, then the algorithm has to **reject**. If the converse is true, i.e. $|L_r(a,b)| > |L_{r'}(a,b)|$ for some block $[a,b]$, then some vertex $v \in L_r(a,b)$ must be included in $S$. Since each vertex in $L_r(a,b)$ has the same neighborhood, the algorithm can choose $v$ arbitrarily from $L_r(a,b)$ and output the **necessary set** $\{v\}$.

If none of these cases happen, then $L(r) = L'(r')$, so as the conditions of Lemma 4.4.2 do not hold, there must exist two indices $i_1 \neq i_2 \in [m]$, such that $G[X_{i_1}]$ and $G[X_{i_2}]$ is not isomorphic to $G'[X'_{i_1}]$ and $G'[X'_{i_2}]$, respectively. As $L(r) = L'(r')$, by Lemma 4.4.1 we get $\phi_S(R^{-1}(r')) = R^{-1}(r)$ and $\phi_S(X'_h) = X_h \setminus S$ for each $h \in [m]$. Thus, it is easy to see that for each $h \in [m]$, the set $S \cap X_h$ yields a solution for the instance $(G'[X'_h], G[X_h])$, and conversely, if $(G', G)$ is solvable then any solution for $(G'[X'_h], G[X_h])$ can be extended to a solution for $(G', G)$. Now, using $X_{i_1} \cap S \neq \emptyset$ and $X_{i_2} \cap S \neq \emptyset$, we know that the parameter of the instance $(G'[X'_{i_1}], G[X_{i_1}])$ must be at least 1 but at most $k-1$. If this indeed holds, then $\mathcal{A}$ outputs $(G'[X'_{i_1}], G[X_{i_1}])$ as an **independent subproblem**, otherwise it **rejects** the instance.

## 4.4.1 Identifying fragments for the case $m > m'$.

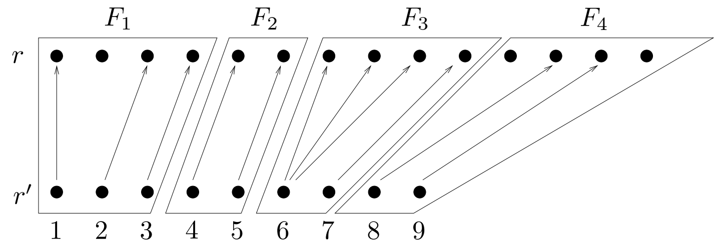
The rest of the paper deals with the case where $S$ is not local, and $m > m'$. In this case, we will try to determine $I_S(j)$ for each $j \in [m']$. To do this, $\mathcal{A}$ will branch several times on determining $I_S(j)$ for some $j \in [m']$. Proposition 4.4.3 helps us to bound the number of resulting branches. To state this proposition, we use the following notation: given some block $[i_1, i_2]$ in $[1,m]$, let $W_r(i_1,i_2)$ contain those vertices $v$ for which $Q_r(v)$ is contained in $[i_1,i_2]$. Let also $w_r(i_1,i_2) = |W_r(i_1,i_2)|$. We define $W_{r'}(j_1,j_2)$ and $w_{r'}(j_1,j_2)$ for some block $[j_1,j_2]$ in $[1,m']$ similarly. Using Lemma 4.4.1 and the definition of $\alpha_S(j)$ and $\beta_S(j)$, it is easy to prove the following:

**Proposition 4.4.3.** *Suppose that $S$ is a solution that is not local, and the properties of Lemma 4.4.1 hold. This implies the followings.*
*(1) $\phi_S(W_{r'}(j_1,j_2)) = W_r(\alpha_S(j_1), \beta_S(j_2)) \setminus S$ for every block $[j_1,j_2]$ in $[1,m']$.*
*(2) $w_{r'}(1,j) \leq w_r(1,\beta_S(j)) \leq w_{r'}(1,j) + k$ and $w_{r'}(j,m') \leq w_r(\alpha_S(j),m) \leq w_{r'}(j,m') + k$ hold for every $j \in [m']$.*

Note that $w_r(1,i) < w_r(1,i+1)$ for every $i \in [m-1]$, even if $X_{i+1} = \emptyset$ (as in this case $M_r^-(i+1) \neq \emptyset$), and similarly we get $w_r(i,m) > w_r(i+1,m)$ for every $i \in [m-1]$. Therefore, the bounds of Proposition 4.4.3 yield at most $(k+1)^2$ possibilities for choosing $[\alpha_S(j), \beta_S(j)]$, for some $j \in [m']$.

Since determining $I_S(j)$ for each $j \in [m']$ using Proposition 4.4.3 would result in too many branches, we need some other tools. Hence, we introduce a structure called fragmentation that can be used to "approximate" the sets $I_S(j)$ for each $j \in [m']$. By iteratively refining the fragmentation, we can get closer and closer to actually determine these sets. Given a set of disjoint blocks $\{[a'_h, b'_h] \mid h \in [f]\}$ in $[1,m']$ and a corresponding set of disjoint blocks $\{[a_h, b_h] \mid h \in [f]\}$ in $[1,m]$ with $F_h$ denoting the pair $([a'_h, b'_h], [a_h, b_h])$, the set $\{F_h \mid h \in [f]\}$ is a *fragmentation* for $(T, T', S)$, if

- $a_h \leq \alpha_S(a'_h)$ and $\beta_S(b'_h) \leq b_h$ for each $h \in [f]$, and



| $F$ | $\delta(F)$ | $\sigma(F)$ |
|---|---|---|
| $F_1$ | 0 | 1 |
| $F_2$ | 1 | 0 |
| $F_3$ | 1 | 2 |
| $F_4$ | 3 | 2 |

Figure 4.6: Illustration of a fragmentation containing four fragments. An arrow leading from $j$ to $i$ indicates $i \in I_S(j)$. There are three non-trivial fragments: $F_1, F_3, F_4$. The indices $1, 4, 5$ are left-aligned, $2, 3, 4, 5, 7$ are right-aligned, $6$ is wide and $8, 9$ are skew.

- $a'_{h+1} = b'_h + 1$ and $a_{h+1} = b_h + 1$ for each $h \in [f-1]$.

We will call the element $F_h$ for some $h \in [f]$ a *fragment*. We define $\sigma(F_h) = (b_h - a_h) - (b'_h - a'_h)$ and $\delta(F_h) = a_h - a'_h$, which are both clearly non-negative integers. Note that $\delta(F_{h+1}) = \delta(F_h) + \sigma(F_h)$ holds for each $h \in [f-1]$. We say that some $j \in [m']$ is *contained* in the fragment $F_h$, if $a'_h \leq j \leq b'_h$. In this case, we write $\delta(j) = \delta(F_h)$ and $\sigma(j) = \sigma(F_h)$. We will say that a fragment $F$ is *trivial* if $\sigma(F_h) = 0$, and *non-trivial* otherwise. We also call an index in $[m']$ trivial (or non-trivial) in a fragmentation, if the fragment containing it is trivial (or non-trivial, respectively). An *annotated fragmentation* for $(T, T', S)$ is a pair $(\mathcal{F}, U)$ formed by a fragmentation $\mathcal{F}$ for $(T, T', S)$ and a set $U \subseteq [m']$ such that each $j \in U$ is trivial in $\mathcal{F}$. We say that the trivial indices contained in $U$ are *important*. See Figure 4.6 for an illustration.

Let us suppose that we are given a fragmentation $\mathcal{F}$ for $(T, T', S)$, and some $j \in [m']$ contained in a fragment $F \in \mathcal{F}$. We will use the notation $j_{\text{left}} = j + \delta(F)$ and $j_{\text{right}} = j + \delta(F) + \sigma(F)$. Also, we will write $B^+_r(i) = M^+_r(i) \cup X_i$ and $B^+_{r'}(j) = M^+_{r'}(j) \cup X'_j$. For some block $[i_1, i_2]$ in $[1, m]$ let $B^+_r(i_1, i_2) = \bigcup_{h \in [i_1, i_2]} B^+_r(h)$, and we define $B^+_{r'}(j_1, j_2)$ for some block $[j_1, j_2]$ in $[1, m']$ analogously. Proposition 4.4.4 is easy to prove, using Lemma 4.4.1.

**Proposition 4.4.4.** *For each $j \in [m']$, the followings hold:*
*(i) $j_{\text{left}} \leq \alpha_S(j) \leq \beta_S(j) \leq j_{\text{right}}$.*
*(ii) $\phi_S(M^+_{r'}(j)) \subseteq \bigcup_{h \in I_S(j)} M^+_r(h)$ and $\phi_S(M^-_{r'}(j)) \subseteq \bigcup_{h \in I_S(j)} M^-_r(h)$.*
*(iii) if $I_S(j) = [i, i]$, then $M^+_r(i) \setminus S = \phi_S(M^+_{r'}(j))$ and $M^-_r(i) \setminus S = \phi_S(M^-_{r'}(j))$.*
*(iv) if $j < j'$, $\alpha_S(j) = i$ and $\beta_S(j') = i'$, then $\phi_S(B^+_{r'}(j, j')) = B^+_r(i, i') \setminus S$.*

We will classify the index $j$ as follows:

- If $|I_S(j)| > 1$, then $j$ is *wide.*
- If $I_S(j) = [j_{\text{left}}, j_{\text{left}}]$, then $j$ is *left-aligned.*
- If $I_S(j) = [j_{\text{right}}, j_{\text{right}}]$, then $j$ is *right-aligned.*
- If $I_S(j) = [i, i]$ such that $j_{\text{left}} < i < j_{\text{right}}$, then $j$ is *skew.*

If $F$ is trivial, then by Proposition 4.4.4, only $\alpha_S(j) = \beta_S(j) = j_{\text{left}} = j_{\text{right}}$ is possible. Thus, each trivial index must be both left- and right-aligned.

Lemma 4.4.5 shows that a solvable instance can only contain at most $2k$ non-trivial fragments. Thus, if a given fragmentation contains more than $2k$ non-trivial fragments, then $\mathcal{A}$ can correctly reject, as such a fragmentation does not correspond to any solution.

**Lemma 4.4.5.** *Any fragmentation $\mathcal{F}$ for $(T, T', S)$ can have at most $2k$ non-trivial fragments.*

*Proof.* We will show that every non-trivial fragment of $\mathcal{F}$ contains an index $j$ for which $X_j \cup M_r^+(j) \cup M_r^-(j)$ contains a vertex of $S$. Since any $s \in S$ can be contained in at most two sets of this form, this proves that there can be at most $2k$ non-trivial fragments in $\mathcal{F}$.

Observe that if $F = ([a', b'], [a, b])$ is a non-trivial fragment in $\mathcal{F}$, then either $|I_S(j)| > 1$ must hold for some $j$ in $[a', b']$, or some $i$ in $[a, b]$ is not contained in any of the blocks $\{I_S(h) \mid a' \leq h \leq b'\}$. In the latter case we have $X_i \cup M_r^+(i) \cup M_r^-(i) \subseteq S$. By Proposition 4.1.2, $X_i \cup M_r^+(i) \cup M_r^-(i) \neq \emptyset$, so it indeed contains a vertex of $S$. Now, suppose that the former case holds, and $j$ is wide. Reversing the children of $r$ between $\alpha_S(j)$ and $\beta_S(j)$ cannot result in a PQ-tree representing $G$, hence there must be a vertex $z \in R^{-1}(r)$ such that $Q_r(z)$ properly intersects $I_S(j)$.

Let $Q_r(z) = [z_1, z_2]$. We assume $z_1 < \alpha_S(j) \leq z_2 < \beta_S(j)$, as the case $\alpha_S(j) < z_1 \leq \beta_S(j) < z_2$ can be handled analogously. First, if $z \in S$, then $M_r^-(z_2)$ contains a vertex of $S$, so we are done. Otherwise, $z \notin S$ implies that $z$ must be contained in $\phi_S(M_{r'}^-(j))$, from which $X_{\beta_S(j)} \cup M_r^+(\beta_S(j)) \subseteq S$ follows. Again, $X_{\beta_S(j)} \cup M_r^+(\beta_S(j)) \neq \emptyset$ by Proposition 4.1.2, so in this case we obtain $\emptyset \neq X_{\beta_S(j)} \cup M_r^+(\beta_S(j)) \subseteq S$. This proves the lemma. □

Before describing the remaining steps of the algorithm, we need some additional notation.

Let $T^{\text{rev}}$ and $T'^{\text{rev}}$ denote the labeled PQ-trees obtained by reversing the children of $r$ and $r'$, respectively. We write $j^{\text{rev}}$ for the index $m' - j + 1$ corresponding to $j$ in $T^{\text{rev}}$, and we also let $X^{\text{rev}} = \{j^{\text{rev}} \mid j \in X\}$ for any set $X \subseteq [m']$. For a fragment $F = ([a', b'], [a, b])$ we let $F^{\text{rev}} = ([b'^{\text{rev}}, a'^{\text{rev}}], [b^{\text{rev}}, a^{\text{rev}}])$, so a fragmentation $\mathcal{F}$ for $(T, T', S)$ clearly yields a fragmentation $\mathcal{F}^{\text{rev}} = \{F^{\text{rev}} | F \in \mathcal{F}\}$ for $(T^{\text{rev}}, T'^{\text{rev}}, S)$. Note that if $j$ is left-aligned (right-aligned) in $\mathcal{F}$, then the index $j^{\text{rev}}$ is right-aligned (left-aligned, resp.) in $\mathcal{F}^{\text{rev}}$.

Given some $i \in [m]$, let us order the vertices $v$ in $M_r^+(i)$ increasingly according to $Q_r^{\text{right}}(v)$. Similarly, we order vertices $v$ in $M_r^-(i)$ increasingly according to $Q_r^{\text{left}}(v)$. In both cases, we break ties arbitrarily. Also, we order vertices of $M_{r'}^+(j)$ and $M_{r'}^-(j)$ in $T'$ the same way for some $j \in [m']$. Now, we construct the sets $P_{\text{left}}^+(j)$ and $P_{\text{left}}^-(j)$, both containing pairs of vertices, in the following way. We put a pair $(v, w)$ into $P_{\text{left}}^+(j)$, if $v \in M_{r'}^+(j)$, $w \in M_r^+(j_{\text{left}})$, and $v$ has the same rank (according to the above ordering) in $M_{r'}^+(j)$ as the rank of $w$ in $M_r^+(j_{\text{left}})$. Similarly, we put a pair $(v, w)$ into $P_{\text{left}}^-(j)$, if $v \in M_{r'}^-(j)$, $w \in M_r^-(j_{\text{left}})$, and $v$ has the same rank in $M_{r'}^-(j)$ as $w$ in $M_r^-(j_{\text{left}})$. In addition, we define the sets $P_{\text{right}}^+(j)$ and $P_{\text{right}}^-(j)$ analogously, by substituting $j_{\text{right}}$ for $j_{\text{left}}$ in the definitions. The key properties of these sets are summarized in the following proposition.

**Proposition 4.4.6.** *W.l.o.g. we can assume $\phi_S(v) = w$ in the following cases:*
*(1) If $(v, w) \in P_{\text{left}}^+(j)$ and $|M_{r'}^+(j)| = |M_r^+(j_{\text{left}})|$ for some left-aligned $j$.*
*(2) If $(v, w) \in P_{\text{left}}^-(j)$ and $|M_{r'}^-(j)| = |M_r^-(j_{\text{left}})|$ for some left-aligned $j$.*
*(3) If $(v, w) \in P_{\text{right}}^+(j)$ and $|M_{r'}^+(j)| = |M_r^+(j_{\text{right}})|$ for some right-aligned $j$.*
*(4) If $(v, w) \in P_{\text{right}}^-(j)$ and $|M_{r'}^-(j)| = |M_r^-(j_{\text{right}})|$ for some right-aligned $j$.*

*Proof.* We only show (1), as all the other statements are analogous. To see (1), observe that as $j$ is left-aligned, $|M_{r'}^+(j)| = |M_r^+(j_{\text{left}})|$ implies that $\phi_S$ must map $M_{r'}^+(j)$ to $M_r^+(j_{\text{left}})$ bijectively.

By Lemma 4.4.1, if $Q^{\text{right}}(v) < Q^{\text{right}}(v')$, then $Q^{\text{right}}(\phi_S(v)) < Q^{\text{right}}(\phi_S(v'))$ holds as well. Note also that the vertices of $L_r(j, j')$ for some $j' \in [m']$ are equivalent in the sense that they have the same neighborhood. Thus, we can assume w.l.o.g. that if $v$ precedes $v'$ in the above defined ordering of $M_{r'}^+(j)$, then $\phi_S(v)$ precedes $\phi_S(v')$ in the similar ordering of $M_r^+(j_{\text{left}})$. Thus, $\phi_S$ must indeed map $v$ to $w$, by the definition of $P_{\text{left}}^+(j)$. □

Given two non-trivial fragments $F$ and $H$ of a fragmentation with $F$ preceding $H$, we define three disjoint subsets of vertices in $R^{-1}(r')$ starting in $F$ and ending in $H$. These sets will be denoted by $\mathcal{L}(F,H)$, $\mathcal{R}(F,H)$, and $\mathcal{X}(F,H)$, and we construct them as follows. Suppose that $v \in L_{r'}(y,j)$ for some $y$ and $j$ contained in $F$ and $H$, respectively. We put $v$ in exactly one of these three sets, if $(v,w) \in P^{-}_{\text{left}}(j)$ for some vertex $w$, and $y_{\text{left}} \leq Q^{\text{left}}_{r}(w) \leq y_{\text{right}}$. Now, if $Q^{\text{left}}_{r}(w) = y_{\text{left}}$ then we put $v$ into $\mathcal{L}(F,H)$, if $Q^{\text{left}}_{r}(w) = y_{\text{right}}$ then we put $v$ into $\mathcal{R}(F,H)$, and if $y_{\text{left}} < Q^{\text{left}}_{r}(w) < y_{\text{right}}$ then we put $v$ into $\mathcal{X}(F,H)$. Loosely speaking, if each vertex in $H$ is left-aligned, and some vertex of $\mathcal{R}(F,H)$ starts at $y$, then $y$ should be right-aligned. Similarly, if each vertex in $H$ is left-aligned, and some vertex of $\mathcal{X}(F,H)$ starts at $y$, then $y$ should be either wide or skew. Since we would like to ensure each index to be left-aligned, we will try to get rid of vertices of $\mathcal{R}(F,H)$ and $\mathcal{X}(F,H)$.

We say that two indices $y_1, y_2 \in [m']$ are *conflicting* for $(F,H)$, if $y_1 \leq y_2$, $M^{+}_{r'}(y_1) \cap \mathcal{R}(F,H) \neq \emptyset$ and $M^{+}_{r'}(y_2) \cap \mathcal{L}(F,H) \neq \emptyset$. In such a case, we say that any $j \geq \max\{j_1, j_2\}$ contained in $H$ is *conflict-inducing* for $(F,H)$ (and for the conflicting pair $(y_1,y_2)$), where $j_1$ denotes the minimal index for which $L_{r'}(y_1,j_1) \cap \mathcal{R}(F,H) \neq \emptyset$, and $j_2$ denotes the minimal index for which $L_{r'}(y_2,j_2) \cap \mathcal{L}(F,H) \neq \emptyset$. Informally, if a conflict-inducing index in $H$ is left-aligned, then this shows that a right-aligned index should precede a left-aligned index in $F$, which cannot happen. In addition, if $\mathcal{L}(F,H) \neq \emptyset$, then let $L^{\max}(F,H)$ denote the largest index $y$ in $F$ for which $M^{+}_{r'}(y) \cap \mathcal{L}(F,H) \neq \emptyset$. Let the *L-critical index for* $(F,H)$ be the smallest index $j$ contained in $H$ for which $L_{r'}(L^{\max}(F,H),j) \cap \mathcal{L}(F,H) \neq \emptyset$. Similarly, if $\mathcal{R}(F,H) \neq \emptyset$, then let $R^{\min}(F,H)$ denote the smallest index $y$ in $F$ for which $M^{+}_{r'}(y) \cap \mathcal{R}(F,H) \neq \emptyset$. Also, let the *R-critical index for* $(F,H)$ be the smallest index $j$ in $H$ for which $L_{r'}(R^{\min}(F,H),j) \cap \mathcal{R}(F,H) \neq \emptyset$.

Now, an index $j$ in $H$ is *LR-critical* for $(F,H)$, if either $j$ is the R-critical index for $(F,H)$ and $\mathcal{L}(F,H) = \emptyset$, or $j = \max\{j_L, j_R\}$ where $j_L$ is the L-critical and $j_R$ is the R-critical index for $(F,H)$. Note that both cases require $\mathcal{R}(F,H) \neq \emptyset$. Moreover, $H$ contains an LR-critical index for $(F,H)$, if and only if $\mathcal{R}(F,H) \neq \emptyset$. Intuitively, if an LR-critical index in $H$ is left-aligned, then this implies that some index $y$ in $F$ is right-aligned.

Note that the definitions of the sets $\mathcal{L}(F,H), \mathcal{R}(F,H)$, and $\mathcal{X}(F,H)$ together with the definitions connected to them as described above depend on the given fragmentation, so whenever the fragmentation changes, these must be adjusted appropriately as well. (In particular, vertices in $\mathcal{L}(F,H), \mathcal{R}(F,H)$, and $\mathcal{X}(F,H)$ must start and end in two different non-trivial fragments.)

Let $(\mathcal{F},U)$ be an annotated fragmentation for $(T,T',S)$. Our aim is to ensure that the properties given below hold for each index $j \in [m']$. Intuitively, these properties mirror the expectation that every index should be left-aligned. Note that although we cannot decide whether $(\mathcal{F},U)$ is a correct fragmentation without knowing the solution $S$, we are able to check whether these properties hold for some index $j$ in $(\mathcal{F},U)$.

**Property 1:** $G'[X'_j]$ is isomorphic to $G[X_{j_{\text{left}}}]$.

**Property 2:** $|M^{+}_{r'}(j)| \leq |M^{+}_{r}(j_{\text{left}})| \leq |M^{+}_{r'}(j)| + k$ and $|M^{-}_{r'}(j)| \leq |M^{-}_{r}(j_{\text{left}})| \leq |M^{-}_{r'}(j)| + k$.

**Property 3:** If $j$ is non-trivial, then $|M^{+}_{r'}(j)| = |M^{+}_{r}(j_{\text{left}})|$ and $|M^{-}_{r'}(j)| = |M^{-}_{r}(j_{\text{left}})|$.

**Property 4:** If $j$ is non-trivial, then $|L_{r'}(y,j)| = |L_{r}(y_{\text{left}}, j_{\text{left}})|$ for any $y < j$ contained in the same fragment as $j$.

**Property 5:** If $j$ is non-trivial, then for every $(v,w) \in P^{+}_{\text{left}}(j)$ where $Q^{\text{right}}_{r'}(v) = y$ is non-trivial, $y_{\text{left}} \leq Q^{\text{right}}_{r}(w) \leq y_{\text{right}}$ holds. Also, for every $(v,w) \in P^{-}_{\text{left}}(j)$ such that $Q^{\text{left}}_{r'}(v) = y$ is non-trivial, $y_{\text{left}} \leq Q^{\text{left}}_{r}(w) \leq y_{\text{right}}$ holds.

**Property 6:** If $j$ is non-trivial, then no vertex in $\mathcal{X}(F,H)$ (for some $F$ and $H$) ends in $j$.

**Property 7:** $j$ is not conflict-inducing for any $(F,H)$.

**Property 8:** $j$ is not LR-critical for any $(F,H)$.

**Property 9:** If $j$ is non-trivial, then for every $(v,w) \in P^{+}_{\text{left}}(j)$ where $Q^{\text{right}}_{r'}(v) = y$ is non-trivial, $Q^{\text{right}}_{r}(w) = y_{\text{left}}$ holds. Also, for every $(v,w) \in P^{-}_{\text{left}}(j)$ such that $Q^{\text{left}}_{r'}(v) = y$ is non-trivial, $Q^{\text{left}}_{r}(w) = y_{\text{left}}$ holds.

**Property 10:** If $j$ is non-trivial, then for each important trivial index $u \in U$, $|L_{r'}(j,u)| = |L_r(j_{\text{left}}, u_{\text{left}})|$ holds if $u > j$, and $|L_{r'}(u,j)| = |L_r(u_{\text{left}}, j_{\text{left}})|$ holds if $u < j$.

Observe that each of these properties depend on the fragmentation $\mathcal{F}$, and Property 10 depends on the set $U$ as well. Also, if some property holds for an index $j$ in $(\mathcal{F},U)$, then this does not imply that the property holds for $j^{\text{rev}}$ in $(\mathcal{F}^{\text{rev}}, U^{\text{rev}})$, as most of these properties are not symmetric. For example, $j_{\text{left}}$ and $j_{\text{right}}$ both have a different meaning in the fragmentation $\mathcal{F}$ and in $\mathcal{F}^{\text{rev}}$. We say that an index $j \in [m']$ *violates* Property $\ell$ $(1 \le \ell \le 10)$ in an annotated fragmentation $(\mathcal{F},U)$, if Property $\ell$ does not hold for $j$ in $(\mathcal{F},U)$. If the first nine properties hold for each index of $[m']$ both in $(\mathcal{F},U)$ and its reversed version $(\mathcal{F}^{\text{rev}}, U^{\text{rev}})$, then we say that $(\mathcal{F},U)$ is *9-proper*. We say that $(\mathcal{F},U)$ is *proper*, if it is 9-proper, and Property 10 holds hold for each index of $[m']$ in $(\mathcal{F},U)$. Observe that $(\mathcal{F},U)$ can be proper even if Property 10 does not hold in $(\mathcal{F}^{\text{rev}}, U^{\text{rev}})$.

Let us describe our strategy. We start with an annotated fragmentation where $U = \emptyset$ and the fragmentation contains only the unique fragment $([1,m'],[1,m])$, implying that there are no trivial indices. Given an annotated fragmentation $(\mathcal{F},U)$, we do the following: if one of Properties $1,2,\dots,10$ does not hold for some index $j$ in $(\mathcal{F},U)$ for $(T,T',S)$, or one of the first nine properties does not hold for some $j$ in the reversed annotated fragmentation $(\mathcal{F}^{\text{rev}}, U^{\text{rev}})$ for $(T^{\text{rev}}, T'^{\text{rev}}, S)$, then we either output a necessary set or an independent subproblem, or we modify the given annotated fragmentation. To do this, we will branch on $I_S(j)$, and handle each possible case according to the type of $j$. Note that we do not care whether Property 10 hold for the indices in the reversed instance. If the given annotated fragmentation is proper, algorithm $\mathcal{A}$ will find a solution using Lemmas 4.4.10 and 4.4.11.

Observe that we can assume w.l.o.g. that there exists an $\ell$ $(1 \le \ell \le 10)$ such that Properties $1,\dots,\ell-1$ hold for each index both in the annotated fragmentation $(\mathcal{F},U)$ and in its reversed version $(\mathcal{F}^{\text{rev}}, U^{\text{rev}})$, but Property $\ell$ is violated by an index in $[m']$ in $(\mathcal{F},U)$. Otherwise, we simply reverse the instance. Let us now remark that we only reverse the instance if this condition is not true.

To begin, the algorithm takes the first index $j$ that violates Property $\ell$, and branches into $(k+1)^2$ directions to choose $I_S(j)$, using Proposition 4.4.3. Then, $\mathcal{A}$ handles each of the cases in a different manner, according to whether $j$ turns out to be left-aligned, right-aligned, skew, or wide. We consider these cases in a general way that is essentially independent from $\ell$, and mainly relies on the type of $j$. We suppose that $j$ is contained in a fragment $F = ([a',b'],[a,b])$, and we say that $j$ is *extremal*, if $j = a'$.

**Left-aligned index.** We deal with the case when $j$ is left-aligned in Section 4.4.3, whose results are summarized by the following lemma.

**Lemma 4.4.7.** *Suppose that Property $\ell$ ($1 \le \ell \le 10$) does not hold for some $j \in [m']$ in the annotated fragmentation $(\mathcal{F},U)$, but all the previous properties hold for each index both in $(\mathcal{F},U)$ and in $(\mathcal{F}^{\text{rev}}, U^{\text{rev}})$. If $j$ is left-aligned, then algorithm $\mathcal{A}$ can do one of the followings in linear time:*

- *produce a necessary set of size at most $2k+1$,*

- *produce an independent subproblem,*
- *produce an index that is either wide or skew,*
- *reject correctly.*

By Lemma 4.4.7, the only case when algorithm $\mathcal{A}$ does not reject or produce an output is the case when it produces an index $j'$ that is wide or skew. If this happens, then $\mathcal{A}$ branches on those choices of $I_S(j')$ where $j'$ is indeed wide or skew, and handles them according to the cases described below. Note that as a consequence, the maximum number of branches in a step may increase from $(k+1)^2$ to $2(k+1)^2 - 1$. (This means that we do not treat the branchings on $I_S(j)$ and $I_S(j')$ separately, and rather consider it as a single branching with at most $2(k+1)^2 - 1$ directions.)

Observe that if $j$ is trivial, then it is both left- and right-aligned. We treat trivial indices as left-aligned.

**Wide index.** Suppose that $j$ is wide, i.e. $|I_S(j)| > 1$. In this case, we can construct a **necessary set** of size 2. Recall that, using the arguments of the proof of Lemma 4.4.5, we can either find a vertex $z \in R^{-1}(r)$ such that if $z \notin S$ then $\emptyset \neq X_{\beta_S(j)} \cup M_r^+(\beta_S(j)) \subseteq S$, or we can find a vertex $w \in R^{-1}(r)$ such that if $w \notin S$ then $\emptyset \neq X_{\alpha_S(j)} \cup M_r^-(\alpha_S(j)) \subseteq S$. Clearly, $\mathcal{A}$ can output a necessary set of size 2 in both cases.

**Extremal right-aligned or skew cases.** Assume that $j = a'$ and $j$ is skew or right-aligned. In these cases, $X_i \cup M_r^+(i) \cup M_r^-(i)$ must be contained in $S$ for each $i$ in $[a, \alpha_S(a')-1]$, so in particular, $X_a \cup M_r^+(a) \cup M_r^-(a) \subseteq S$. As $X_a \cup M_r^+(a) \cup M_r^-(a) \neq \emptyset$ by Proposition 4.1.2, we can construct a **necessary set** of size 1 by taking an arbitrary vertex from this set, and $\mathcal{A}$ can stop by outputting it.

**Non-extremal skew case.** Suppose that $j > a'$ and $j$ is skew, meaning that $I_S(j) = [i,i]$ for some $i$ with $j_{\text{left}} < i < j_{\text{right}}$. In this case, we can divide the fragment $F$, or more precisely, we can delete $F$ from the fragmentation $\mathcal{F}$ and add the new fragments $([a', j-1], [a, i-1])$ and $([j, b'], [i, b])$. Note that the newly introduced fragments are non-trivial by the bounds on $i$. We also modify $U$ by declaring every trivial index of the fragmentation to be important (no matter whether it was important or not before).

**Non-extremal right-aligned case.** Suppose that $j > a'$ and $j$ is right-aligned. In this case, we replace $F$ by new fragments $F_1 = ([a', j-1], [a, j_{\text{right}}-1])$ and $F_2 = ([j, b'], [j_{\text{right}}, b])$. This yields a fragmentation where $F_1$ is non-trivial and $F_2$ is trivial. We refer to this operation as performing a *right split* at $j$. If this happens because $j$ violated Property $\ell$ for some $\ell \leq 9$, then we set every trivial index (including those contained in $F_2$) to be important, by putting them into $U$. In the case when $\ell = 10$, we do not modify $U$, so the trivial indices of $F_2$ will not be important.

The above process either stops by producing an appropriate output, or it ends by providing an annotated fragmentation that is proper. Thanks to the observations of Lemma 4.4.12, stating that the properties ensured during some step in this process will not be violated later on (except for a few cases), we will be able to bound the running time of this process in Section 4.4.2, by proving that the height of the explored search tree is bounded by a function of $k$. In the remaining steps of the algorithm, the set $U$ will never be modified, and the only possible modification of the actual fragmentation will be to perform a right split.

The following two lemmas capture some useful properties of an arbitrary annotated fragmentation $(\mathcal{F}, U)$ obtained by the algorithm after this point. Lemma 4.4.8 states facts about an annotated fragmentation obtained from a 9-proper annotated fragmentation by applying right splits to it. Lemma 4.4.9 gives sufficient conditions for the properties of an annotated fragmentation to remain true after applying a right split to it.

**Lemma 4.4.8.** *Let $(\mathcal{F}, U)$ be a 9-proper annotated fragmentation whose trivial indices are all important. Suppose that $\mathcal{F}'$ is obtained by applying an arbitrary number of right splits to the fragmentation $\mathcal{F}$. Then the followings hold for each $j \in [m']$ that is either non-trivial or not important in $(\mathcal{F}', U)$:*
*(1) $|M^+_{r'}(j)| = |M^+_r(j_{\text{right}})|$ and $|M^-_{r'}(j)| = |M^-_r(j_{\text{right}})|$.*
*(2) The following holds for every non-trivial or not important $y \neq j$ and $v \in L_{r'}(j, y)$. If $(v, w) \in P^+_{\text{right}}(j)$ for some $w \in M^+_r(j_{\text{right}})$, then $Q^{\text{right}}_r(w) = y_{\text{right}}$. Similarly, if $(v, w) \in P^-_{\text{right}}(j)$ for some $w \in M^-_r(j_{\text{right}})$, then $Q^{\text{left}}_r(w) = y_{\text{right}}$.*

*Proof.* First, we show that the statements of the lemma hold for $(\mathcal{F}, U)$. To see this, recall that each trivial index in $(\mathcal{F}, U)$ is important, therefore statements (1) and (2) for $(\mathcal{F}, U)$ are equivalent to Properties 3 and 9 for $(\mathcal{F}^{\text{rev}}, U^{\text{rev}})$, respectively. Since $(\mathcal{F}, U)$ is 9-proper, these properties indeed hold for each index in $(\mathcal{F}^{\text{rev}}, U^{\text{rev}})$.

To see that these statements remain true after applying a sequence of right splits to $(\mathcal{F}, U)$, we need two simple observations. First, notice that the value of $j_{\text{right}}$ for an index $j \in [m']$ does not change in a right split. Second, the set of non-trivial or not important trivial indices does not change either, since the performed right splits do not modify the set $U$ of important trivial indices. Thus, statements (1) and (2) for some index $j$ have exactly the same meaning in $(\mathcal{F}', U)$ as in $(\mathcal{F}, U)$. This proves the lemma. □

Given a fragmentation $\mathcal{F}$ for $(T, T', S)$, a fragment $F$, and some $\ell$ $(1 \leq \ell \leq 9)$, let $\pi(\mathcal{F}, F, \ell)$ be 1 if Property $\ell$ holds for each index $j$ in $F \in \mathcal{F}$, and 0 otherwise.

**Lemma 4.4.9.** *Let $\mathcal{F}'$ be a fragmentation obtained from $\mathcal{F}$ by dividing a fragment $F \in \mathcal{F}$ into fragments $F_1$ and $F_2$ with a right split (with $F_1$ preceding $F_2$). Let $1 \leq \ell \leq 9$.*
*(1) Suppose $j$ is not contained in $F_2$ and $\ell \neq 8$. If Property $\ell$ holds for $j$ in $\mathcal{F}$ (or in $\mathcal{F}^{\text{rev}}$), then Property $\ell$ holds for $j$ in $\mathcal{F}'$ (or in $\mathcal{F}'^{\text{rev}}$) as well.*
*(2) Suppose $\pi(\mathcal{F}, H, \ell) = 1$ for a fragment $H$. If $H \neq F$ then $\pi(\mathcal{F}', H, \ell) = 1$, and if $H = F$ then $\pi(\mathcal{F}', F_1, \ell) = 1$.*
*(3) Suppose $\pi(\mathcal{F}^{\text{rev}}, H^{\text{rev}}, \ell) = 1$ for a fragment $H \in \mathcal{F}$. If $H \neq F$ then $\pi(\mathcal{F}'^{\text{rev}}, H^{\text{rev}}, \ell) = 1$, and if $H = F$ then $\pi(\mathcal{F}'^{\text{rev}}, F_1^{\text{rev}}, \ell) = 1$.*
*(4) If $\pi(\mathcal{F}^{\text{rev}}, F^{\text{rev}}, \ell) = 1$, then $\pi(\mathcal{F}'^{\text{rev}}, F_2^{\text{rev}}, \ell) = \pi(\mathcal{F}', F_2, \ell) = 1$.*
*(5) If $(\mathcal{F}, U)$ is a proper annotated fragmentation, then so is $(\mathcal{F}', U)$.*

*Proof.* To see (1), we need some basic observations. First, if $j$ is not contained in $F_2$, then $j_{\text{left}}$ is the same according to $\mathcal{F}'$ as it is in $\mathcal{F}$, and this is also true for $j_{\text{right}}$. Second, the set of non-trivial indices in $\mathcal{F}'$ is a subset of the non-trivial indices in $\mathcal{F}$. These conditions directly imply (1) for each case where $\ell \notin \{6, 7, 8\}$, using only the definitions of these properties.

Now, observe that if a vertex in $R^{-1}(r')$ is contained in $\mathcal{L}(H'_1, H'_2)$, for some $H'_1$ and $H'_2$ in the fragmentation $\mathcal{F}'$, then it is contained in $\mathcal{L}(H_1, H_2)$ for some $H_1$ and $H_2$ in $\mathcal{F}$ as well. Clearly, the analogous fact holds also for the sets $\mathcal{R}(H'_1, H'_2)$ and $\mathcal{X}(H'_1, H'_2)$ for some $H'_1$ and $H'_2$. Thus, if $j$ violates Property 6 or 7 in $\mathcal{F}'$, then it also violates it in $\mathcal{F}$, proving (1).

Clearly, (2) and (3) follow directly from (1) in the cases where $\ell \neq 8$. For the case $\ell = 8$, observe that $\pi(\mathcal{F}, H, 8) = 1$ implies $\mathcal{R}(H_0, H) = \emptyset$ for every $H_0$ preceding $H$. Hence, the requirements of statement (2) follow immediately. The analogous claim in the reversed instance shows that (3) also holds for $\ell = 8$.

To prove (4), let $j$ be contained in $F_2$. Note that Properties $3, 4, \ldots, 9$ vacuously hold for $j$ in $\mathcal{F}'$, because $F_2$ is trivial. Using that $j_{\text{left}} = j_{\text{right}}$ and the definitions of Properties 1 and 2, we get that if one of these two properties holds for $j^{\text{rev}}$ in $\mathcal{F}^{\text{rev}}$, then it holds for $j$ in $\mathcal{F}'$ as well. Finally, observe that if Property $\ell$ holds for some trivial index $j$ in $\mathcal{F}'$, then it trivially holds for $j^{\text{rev}}$ in $\mathcal{F}'^{\text{rev}}$, proving (4).

| $a \setminus b$ | | trivial | | non-trivial | |
|---|---|---|---|---|---|
| | | $\notin U$ | $\in U$ | $\notin Z$ | $\in W$ |
| trivial | $\in U$ | (v) | | (i) | |
| | $\notin U$ | | | | (iv) |
| non-trivial | | (ii) | | | (iii) |

Table 4.1: The cases of Lemma 4.4.10, where $U$ denotes important indices, $Z$ denotes the last indices of non-trivial fragments, and $W$ denotes right-constrained indices.

To prove (5), assume that $(\mathcal{F}, U)$ is proper. By (2), (3), and (4), we immediately obtain that $(\mathcal{F}', U)$ is 9-proper, so we only have to verify that Property 10 holds. But since the set $U$ of important trivial indices is the same in both fragmentations, and $j_{\text{left}}$ is the same in $(\mathcal{F}', U)$ as in $(\mathcal{F}, U)$ for each non-trivial or important trivial index $j$ of $\mathcal{F}'$, Property 10 also remains true for each index. □

Given a proper annotated fragmentation $(\mathcal{F}, U)$, algorithm $\mathcal{A}$ makes use of Lemma 4.4.10 below.

To state Lemma 4.4.10, we need one more definition: we call an index $j$ *right-constrained*, if $j$ is contained in a non-trivial fragment $F$, and there exists a vertex $v \in M^+_{r'}(j)$ such that $\phi_S(v) \in M^+_r(j_{\text{right}})$. Note that this definition depends on the solution $S$. Algorithm $\mathcal{A}$ maintains a set $W$ to store indices which turn out to be right-constrained. We will show that if $j$ is right-constrained, then $j+1$ must be right-aligned and thus a right split can be performed, except for the case when $j$ is the last index of the fragment. We will denote by $Z_{\mathcal{F}}$ the set of indices $j$ for which $j$ is the last index of some non-trivial fragment in $\mathcal{F}$. If no confusion arises, we will drop the subscript $\mathcal{F}$.

Lemma 4.4.10 gives sufficient conditions for $\mathcal{A}$ to do some of the followings.

- Find out that some non-trivial index $j$ is right-aligned. In this case, $\mathcal{A}$ performs a right split at $j$ in the actual fragmentation.
- Find out that some index $j$ is right-constrained, and put it into $W$.
- Reject, or stop by outputting a necessary set of size 1.

The algorithm applies Lemma 4.4.10 repeatedly, until it either stops or it finds that none of the conditions given in the lemma apply.

**Lemma 4.4.10.** *Let $(\mathcal{F}, U)$ be a proper annotated fragmentation for $(T, T', S)$ obtained by algorithm $\mathcal{A}$, and let $a, b \in [m']$ with $a < b$.*
*(i) If $a$ is trivial but not important, $b$ is non-trivial, $b \notin Z$ and $L_{r'}(a, b) \neq \emptyset$, then $b+1$ is right-aligned.*
*(ii) If $a$ is non-trivial, $b$ is trivial but not important, and $L_{r'}(a, b) \neq \emptyset$, then $a$ is right-constrained. Also, if $a \notin Z$ then $a+1$ is right-aligned.*
*(iii) If $a$ is non-trivial, $b$ is right-constrained, and $L_{r'}(a, b) \neq \emptyset$, then $a$ is right-constrained. Also, if $a \notin Z$ then $a+1$ is right-aligned.*
*(iv) If $a \in U$, $b$ is right-constrained, and $|L_{r'}(a, b)| \neq |L_r(a_{\text{left}}, b_{\text{right}})|$, then algorithm $\mathcal{A}$ can either **reject** or output a **necessary set** of size 1.*
*(v) If $a$ and $b$ are trivial and $|L_{r'}(a, b)| \neq |L_r(a_{\text{left}}, b_{\text{left}})|$, then algorithm $\mathcal{A}$ can either **reject** or output a **necessary set** of size 1.*

*Proof.* Let $A$ and $B$ be the fragments in $\mathcal{F}$ containing $a$ and $b$, respectively. Recall that the conditions of Lemma 4.4.8 are true for every proper annotated fragmentation for $(T, T', S)$ obtained by algorithm $\mathcal{A}$, in particular for $\mathcal{F}$.

First, suppose that the conditions of (i) hold. As $a$ is a not important trivial index in $(\mathcal{F}, U)$, claim (1) of Lemma 4.4.8 implies $|M^+_{r'}(a)| = |M^+_r(a_{\text{right}})|$. Let $v \in L_{r'}(a, b)$ and $(v, w) \in P^+_{\text{right}}(a)$. As $a$ is trivial, it is right-aligned as well, so we obtain $\phi_S(v) = w$ by Proposition 4.4.6. Using claim (2) of Lemma 4.4.8 for $a$, we obtain $Q^{\text{right}}_r(w) = b_{\text{right}}$. By $\phi_S(v) = w$, this implies $\beta_S(b) \geq b_{\text{right}}$. Thus, $\alpha_S(b+1) \geq b_{\text{right}} + 1 = (b+1)_{\text{right}}$, showing that $b + 1$ is indeed right-aligned.

The proof of (ii) is analogous with the proof of (i). By exchanging the roles of $a$ and $b$, we obtain $Q_r(\phi_S(v)) = [a_{\text{right}}, b_{\text{right}}]$ for some $v \in L_{r'}(a, b)$ in a straightforward way. Observe that this proves $a$ to be right-constrained. If $a \notin Z$, then $A$ contains $a + 1$ as well. Hence, from $\beta_S(a) \geq a_{\text{right}}$ we get $\alpha_S(a+1) \geq a_{\text{right}} + 1 = (a+1)_{\text{right}}$. Thus, $a + 1$ is right-aligned.

To see (iii) and (iv), suppose that $b$ is right-constrained and $u^+$ is a vertex in $M^+_{r'}(b)$ with $\phi_S(u^+) \in M^+_r(b_{\text{right}})$. Suppose $u^- \in M^-_{r'}(b)$ for some $u^-$. Clearly, $u^-u^+$ is an edge in $G'$, so $\phi_S(u^-)$ and $\phi_S(u^+)$ must be adjacent in $G$ as well. By $\phi_S(u^+) \in M^+_r(b_{\text{right}})$ we get $Q^{\text{right}}_r(\phi_S(u^-)) \geq b_{\text{right}}$. By Proposition 4.4.4, this implies $Q^{\text{right}}_r(\phi_S(u^-)) = b_{\text{right}}$. Using claim (1) of Lemma 4.4.8 for $b$, we get $M^-_r(b_{\text{right}}) = \phi_S(M^-_{r'}(b))$. Letting $v \in L_{r'}(a, b)$ and $(v, w) \in P^-_{\text{right}}(b)$ we obtain $\phi_S(v) = w$ as in Proposition 4.4.6.

To prove (iii), assume also that $a$ is non-trivial. By claim (2) of Lemma 4.4.8 for $b$, this implies $Q_r(w) = [a_{\text{right}}, b_{\text{right}}]$. This means that $a$ is right-constrained. From $a \notin Z$ we again obtain that $a + 1$ is right-aligned, using the arguments of the proof of (ii).

To see (iv), assume $a \in U$. Using Proposition 4.4.4, $I_S(a) = [a_{\text{left}}, a_{\text{left}}]$, and the above mentioned arguments, we get that $\phi_S(L_{r'}(a, b)) = L_r(a_{\text{left}}, b_{\text{right}}) \backslash S$. Therefore, if $|L_{r'}(a, b)| > |L_r(a_{\text{left}}, b_{\text{right}})|$ then $\mathcal{A}$ can reject, and if $|L_{r'}(a, b)| < |L_r(a_{\text{left}}, b_{\text{right}})|$ then it can output a necessary set of size 1 by outputting $\{s\}$ for an arbitrary $s \in L_r(a_{\text{left}}, b_{\text{right}})$.

Finally, assume that the conditions of (v) hold for $a$ and $b$. As both of them are left-aligned, Proposition 4.4.4 implies $\phi_S(L_{r'}(a, b)) = L_r(a_{\text{left}}, b_{\text{left}}) \setminus S$. Hence, $\mathcal{A}$ can proceed essentially the same way as in the previous case. □

After applying Lemma 4.4.10 repeatedly, algorithm $\mathcal{A}$ either stops by outputting 'No' or a necessary set of size 1, or it finds that none of the conditions (i)-(v) of Lemma 4.4.10 holds. Observe that each $w \in W$ must be the last index of the fragment containing $w$, since whenever $\mathcal{A}$ puts some index $j \notin Z$ into $W$, then it also sets $j + 1$ right-aligned, resulting in a right split.

Let $(\mathcal{F}, U)$ be the final annotated fragmentation obtained. Note that the algorithm does not modify the set $U$ of important trivial indices when applying Lemma 4.4.10, and it can only modify the actual fragmentation by performing a right split. Thus, statements (1) and (2) of Lemma 4.4.8 remain true for $(\mathcal{F}, U)$. By claim (5) of Lemma 4.4.9, we obtain that $(\mathcal{F}, U)$ remains proper as well. Making use of these lemmas, Lemma 4.4.11 yields that $\mathcal{A}$ can find a solution in linear time. This finishes the description of algorithm $\mathcal{A}$.

**Lemma 4.4.11.** *Let $(\mathcal{F}, U)$ be a proper annotated fragmentation for $(T, T', S)$ obtained by algorithm $\mathcal{A}$. If none of the conditions (i)-(v) of Lemma 4.4.10 holds, then $\mathcal{A}$ can produce a solution in linear time.*

*Proof.* We construct an isomorphism $\phi$ from $G'$ to an induced subgraph of $G$. Our basic approach is to treat almost all indices as if they were left-aligned, except for the vertices of $W$. Recall that $Z$ denotes the set of indices that are the last index of some non-trivial fragment, and $W \subseteq Z$ is the set of right-constrained vertices that $\mathcal{A}$ has found using Lemma 4.4.10. Let $N$ contain those non-trivial indices in $[m']$ that are not in $W$. Also, let $Y$ denote the set of trivial indices in $[m']$ that are not important. Clearly, $[m'] = N \cup W \cup U \cup Y$.

As Property 1 holds both for $\mathcal{F}$ in $(T, T', S)$ and for $\mathcal{F}^{\text{rev}}$ in $(T^{\text{rev}}, T'^{\text{rev}}, S)$, we know that there is an isomorphism $\phi_j^{\text{left}}$ from $G'[X'_j]$ to $G[X_{j_{\text{left}}}]$ and an isomorphism $\phi_j^{\text{right}}$ from $G'[X'_j]$ to $G[X_{j_{\text{right}}}]$ for each $j \in [m']$. By [96], $\phi_j^{\text{left}}$ and $\phi_j^{\text{right}}$ can be found in time linear in $|X'_j|$. We set $\phi(x) = \phi_j^{\text{left}}(x)$ for each $x \in X'_j$ where $j \in N \cup U \cup Y$, and we set $\phi(x) = \phi_j^{\text{right}}(x)$ for each $x \in X'_j$ where $j \in W$. Our aim is to extend $\phi$ on vertices of $R^{-1}(r')$ such that it remains an isomorphism. To this end, we set a variable $\Delta(j)$ for each $j \in [m']$, by letting $\Delta(j) = j_{\text{left}}$ if $j \in N \cup U \cup Y$, and $\Delta(j) = j_{\text{right}}$ if $j \in W$. Clearly, $\Delta(j) = j_{\text{left}} = j_{\text{right}}$ if $j \in U \cup Y$.

The purpose of the notation $\Delta$ is the following. Given some $a < b$, in almost every case we will let $\phi$ map vertices of $L_{r'}(a, b)$ bijectively to vertices of $L_r(\Delta(a), \Delta(b))$. This can be done if $a$ and $b$ *match*, meaning that $|L_{r'}(a, b)| = |L_r(\Delta(a), \Delta(b))|$. However, there remain cases where $a$ and $b$ do not match. Each such case will fulfill one of the following conditions:

(A) $a \in W$ and $|L_{r'}(a, b)| = |L_r(a_{\text{left}}, \Delta(b))|$.
In this case, we let $\phi$ map $L_{r'}(a, b)$ bijectively to $L_r(a_{\text{left}}, \Delta(b))$. The block $[a_{\text{left}}, \Delta(b)]$ contains $\Delta(a) = a_{\text{right}}$. Thus, every vertex of $\phi(L_{r'}(a, b))$ will be adjacent to the vertices of $M_r(\Delta(a)) \cup X_{\Delta(a)}$. Since either $a-1 \in N$ or $a-1$ is not in the same fragment as $a$, we obtain $\Delta(a-1) < a_{\text{left}}$. Hence, vertices of $\phi(L_{r'}(a, b))$ will not be adjacent to vertices of $M_r^-(\Delta(a-1)) \cup X_{\Delta(a-1)}$.

(B) $b \in Z$ and $|L_{r'}(a, b)| = |L_r(\Delta(a), b_{\text{right}})|$.
In this case, we let $\phi$ map $L_{r'}(a, b)$ bijectively to $L_r(\Delta(a), b_{\text{right}})$. Again, $[\Delta(a), b_{\text{right}}]$ contains $\Delta(b) = b_{\text{left}}$, therefore the vertices of $\phi(L_{r'}(a, b))$ will be adjacent to the vertices of $M_r(\Delta(b)) \cup X_{\Delta(b)}$. Also, by $b \in Z$ we obtain $\Delta(b+1) \geq (b+1)_{\text{left}} > b_{\text{right}}$, so the vertices of $\phi(L_{r'}(a, b))$ will not be adjacent to vertices of $M_r^+(\Delta(b+1)) \cup X_{\Delta(b+1)}$.

It is easy to see that the above construction ensures that the vertices of $\phi(L_{r'}(a_1, b_1))$ and $\phi(L_{r'}(a_2, b_2))$ are neighboring if and only if $L_{r'}(a_1, b_1)$ and $L_{r'}(a_2, b_2)$ are neighboring. (In particular, it is not possible that some vertex of $\phi(M_{r'}^-(j))$ ends in $j_{\text{left}}$ but some vertex of $\phi(M_{r'}^+(j))$ starts in $j_{\text{right}}$. )

It remains to show that if $a, b \in [m']$ and $a < b$, then they either match, or $L_{r'}(a, b) = \emptyset$, or one of the conditions (A) or (B) hold. First, let us show those cases where $a$ and $b$ match.

(a) If $a, b \in N$, then $|L_{r'}(a, b)| = |L_r(a_{\text{left}}, b_{\text{left}})| = |L_r(\Delta(a), \Delta(b))|$ because Properties 3 and 9 hold for $b$ in $(\mathcal{F}, U)$.

(b) If $a \in N$ and $b \in U$ or vice versa, then $|L_{r'}(a, b)| = |L_r(a_{\text{left}}, b_{\text{left}})| = |L_r(\Delta(a), \Delta(b))|$, since Property 10 holds for $a$ and $b$ in $(\mathcal{F}, U)$.

(c) If $a, b \in W \cup Y$ then Lemma 4.4.8 for $b$ guarantees $|L_{r'}(a, b)| = |L_r(a_{\text{right}}, b_{\text{right}})|$. Using that $a_{\text{right}} = \Delta(a)$ and $b_{\text{right}} = \Delta(b)$ hold if $a, b \in W \cup Y$, this immediately shows $|L_{r'}(a, b)| = |L_r(\Delta(a), \Delta(b))|$.

(d) If $a \in U$ and $b \in W$ then $|L_{r'}(a, b)| = |L_r(a_{\text{left}}, b_{\text{right}})|$, since the conditions of (iv) in Lemma 4.4.10 do not apply. By $a_{\text{left}} = \Delta(a)$ and $b_{\text{right}} = \Delta(b)$, this again means $|L_{r'}(a, b)| = |L_r(\Delta(a), \Delta(b))|$.

(e) If $a, b \in U \cup Y$ then $|L_{r'}(a, b)| = |L_r(a_{\text{left}}, b_{\text{left}})| = |L_r(\Delta(a), \Delta(b))|$, as the conditions of (v) in Lemma 4.4.10 do not apply.

Next, we show $L_{r'}(a, b) = \emptyset$ for some $a$ and $b$ with $a < b$. First, if $a \in Y$ and $b \in N \setminus Z$, then this holds because (i) of Lemma 4.4.10 is not applicable. Also, $L_{r'}(a, b) = \emptyset$ must be true if $a \in N$ and $b \in Y$, as otherwise (ii) of Lemma 4.4.10 would apply. Third, $L_{r'}(a, b) = \emptyset$ if $a \in N$ and $b \in W$, since (iii) of Lemma 4.4.10 does not apply.

We complete the proof by showing (A) or (B) for all remaining cases.

|  | $N$ |  |  |  |  |
|---|---|---|---|---|---|
| $a \backslash b$ | $N \setminus Z$ | $Z$ | $W$ | $Y$ | $U$ |
| $N$ | a |  | – | – | b |
| $W$ | f |  | c |  | g |
| $Y$ | – | h |  |  |  |
| $U$ | b |  | d |  | e |

Table 4.2: The cases of the proof for Lemma 4.4.11.

(f) If $a \in W$ and $b \in N$, then (A) holds, because by Properties 3 and 9 for $b$ in $(\mathcal{F}, U)$, we obtain $|L_{r'}(a, b)| = |L_r(a_{\text{left}}, b_{\text{left}})|$.

(g) If $a \in W$ and $b \in U$, then we have $|L_{r'}(a, b)| = |L_r(a_{\text{left}}, b_{\text{left}})|$, since Property 10 holds for $a$ in $(\mathcal{F}, U)$. Hence, this case also fulfills (A).

(h) If $a \in Y$ and $b \in Z$, then $|L_{r'}(a, b)| = |L_r(a_{\text{right}}, b_{\text{right}})|$ by (1) and (2) of Lemma 4.4.8 for $b$. By $a_{\text{left}} = a_{\text{right}}$, (B) holds.

Table 4.2 shows that we considered every case. Thus, $\phi$ is an isomorphism from $G'$ to an induced subgraph of $G$, so $\mathcal{A}$ can output $V(G) \setminus \phi(V(G'))$ as a solution. It is also clear that this takes linear time. □

## 4.4.2 Running time analysis for algorithm $\mathcal{A}$

Let $N(\mathcal{F})$ denote the set of non-trivial fragments in $\mathcal{F}$. We define the *measure* $\mu(\mathcal{F})$ of a given fragmentation $\mathcal{F}$ for $(T, T', S)$ as follows:

$$\mu(\mathcal{F}) = \sum_{\substack{F \in N(\mathcal{F}) \\ 1 \leq \ell \leq 9}} \pi(\mathcal{F}, F, \ell) + \sum_{\substack{F \in N(\mathcal{F}^{\text{rev}}) \\ 1 \leq \ell \leq 9}} \pi(\mathcal{F}^{\text{rev}}, F, \ell).$$

Note that $\mu(\mathcal{F}) = \mu(\mathcal{F}^{\text{rev}})$ is trivial, so reversing a fragmentation does not change its measure. Recall that $\mathcal{F}^{\text{rev}}$ is a fragmentation for $(T^{\text{rev}}, T'^{\text{rev}}, S)$.

**Lemma 4.4.12.** *Let $\mathcal{F}_1, \ldots, \mathcal{F}_t, \mathcal{F}_{t+1}$ be a series a fragmentations such that for each $i \in [t]$ algorithm $\mathcal{A}$ obtains $\mathcal{F}_{i+1}$ from $\mathcal{F}_i$ by applying a right split at an index $j_i$ violating Property $\ell_i$ in $\mathcal{F}_i$. Let $H_i$ denote the fragment of $\mathcal{F}_i$ containing $j_i$.*
*(1) $\mu(\mathcal{F}_{i+1}) \geq \mu(\mathcal{F}_i)$ for each $i \in [t]$. If $\ell_i \neq 8$, then $\mu(\mathcal{F}_{i+1}) > \mu(\mathcal{F}_i)$ also holds.*
*(2) If $\mu(\mathcal{F}_1) = \mu(\mathcal{F}_t)$, then $\ell_i = 8$ for every $i \in [t]$, and $H_i$ contains every index in $H_{i+1}$ for each $i \in [t]$.*
*(3) If $\mu(\mathcal{F}_1) = \mu(\mathcal{F}_t)$, then $t \leq k$.*

*Proof.* To prove (1), observe that $\mu(\mathcal{F}_{i+1}) \geq \mu(\mathcal{F}_i)$ follows directly from claims (2) and (3) of Lemma 4.4.9. Let $H'_i$ be the non-trivial fragment obtained from $H_i$ after the right split at $j_i$. Now, by the choice of $j_i$, Property $\ell_i$ is violated by $j_i$ in $\mathcal{F}_i$, but is not violated by any index $j'$ preceding $j_i$ in $\mathcal{F}_i$. In all cases where $\ell_i \neq 8$, claim (1) of Lemma 4.4.9 implies that the indices preceding $j_i$ cannot violate Property $\ell_i$ in $\mathcal{F}_{i+1}$, yielding $\pi(\mathcal{F}_i, H_i, \ell_i) = 0$ but $\pi(\mathcal{F}_i, H'_i, \ell_i) = 1$. Considering claims (2) and (3) of Lemma 4.4.9 again, (1) follows.

Observe that if $\mu(\mathcal{F}_1) = \mu(\mathcal{F}_t)$, then $\ell_i = 8$ for every $i \in [t]$ follows directly from the above discussion. Suppose that $H_i$ is a counterexample for (2), meaning that $H_i$ does not contain the indices of $H_{i+1}$. Since $\mathcal{F}_{i+1}$ is obtained from $\mathcal{F}_i$ by a right split, this can only happen if $H_{i+1}$ is a non-trivial fragment of $\mathcal{F}_i$ different from $H_i$. Recall that a fragment $B$ contains some index violating Property 8, if and only if $\mathcal{R}(A, B) \neq \emptyset$ holds for some fragment $A$ in the fragmentation. Hence, $\pi(\mathcal{F}_{i+1}, H_{i+1}, 8) = 0$ implies $\pi(\mathcal{F}_i, H_{i+1}, 8) = 0$.

Since the algorithm always chooses the first index violating some property to branch on, $j_i$ must be the smallest index that is LR-critical for some pair of fragments in $\mathcal{F}_i$. Therefore, $H_i$ must precede $H_{i+1}$. But now, the choice of $j_{i+1}$ indicates $\pi(\mathcal{F}_{i+1}, H'_i, 8) = 1$, where $H'_i$ is the non-trivial fragment of $\mathcal{F}_{i+1}$ obtained by splitting $H_i$ at $j_i$ in $\mathcal{F}_i$. Together with $\pi(\mathcal{F}_i, H_i, 8) = 0$ and statements (2) and (3) of Lemma 4.4.9, this shows $\mu(\mathcal{F}_{i+1}) > \mu(\mathcal{F}_i)$, a contradiction.

To prove (3), let $H \in \mathcal{F}_1$ contain every $j_i$. Let $P(H)$ denote the set of non-trivial fragments in $\mathcal{F}_1$ preceding $H$. For some $i \in [t]$ and some $F \in P(H)$, let $F_i$ denote the unique non-trivial fragment of $\mathcal{F}_i$ that contains some of the indices contained in $F$. We denote by $PR^i(H)$ those fragments $F$ in $P(H)$ for which $\mathcal{R}(F_i, H_i) \neq \emptyset$ holds in $\mathcal{F}_i$. Clearly, $PR^i(H) \neq \emptyset$ for any $i \in [t]$ by claim (2), and note also $PR^{i+1}(H) \subseteq PR^i(H)$. If $F \in PR^i(H)$, then we define $d^i(F)$ as follows. If $\mathcal{L}(F_i, H_i) = \emptyset$ in $\mathcal{F}_i$, then let $y_i^L$ be the first index contained in $F_i$ minus one, otherwise let $y_i^L$ have the value of $L^{\max}(F_i, H_i)$ in $\mathcal{F}_i$. Also, let $y_i^R$ be the value of $R^{\min}(F_i, H_i)$ in $\mathcal{F}_i$. We set $d^i(F) = y_i^R - y_i^L$.

Let $A \in PR^{i+1}(H)$ denote a non-trivial fragment such that $j_i$ is LR-critical for $(A_i, H_i)$. We show $d^{i+1}(A) > d^i(A)$. Clearly, $j_i$ is either L-critical or R-critical for $(A_i, H_i)$. First, let us assume that $j_i$ is L-critical for $(A_i, H_i)$. Observe that the definition of L-critical indices implies that for any vertex $v$ starting at $L^{\max}(A_i, H_i)$ and contained in $\mathcal{L}(A_i, H_i)$ in $\mathcal{F}_i$, we know $Q_{r'}^{\text{right}}(v) \geq j_i$. Since $\mathcal{F}_{i+1}$ is obtained by performing the right split at $j_i$ in $\mathcal{F}_i$, every index of $H_{i+1}$ precedes $j_i$, implying that such a $v$ can not be contained in $\mathcal{L}(A_{i+1}, H_{i+1})$ in $\mathcal{F}_{i+1}$. Thus, $L^{\max}(A_{i+1}, H_{i+1}) \neq L^{\max}(A_i, H_i)$, from which $y_{i+1}^L < y_i^L$ follows. Therefore, we have $d^{i+1}(A) > d^i(A)$.

Second, let us assume that $j_i$ is R-critical for $(A_i, H_i)$. Using the definition of R-criticality, implies that for any vertex $v$ starting at $R^{\min}(A_i, H_i)$ and contained in $\mathcal{R}(A_i, H_i)$ in $\mathcal{F}_i$, we know $Q_{r'}^{\text{right}}(v) \geq j_i$. Again, we know that every index of $H_{i+1}$ precedes $j_i$. From this, we have that $v$ can not be contained in $\mathcal{R}(A_{i+1}, H_{i+1})$ in $\mathcal{F}_{i+1}$, implying $y_{i+1}^R > y_i^R$. Therefore, we have $d^{i+1}(A) > d^i(A)$ in this case as well.

Now, we claim that $1 \leq d^i(A) \leq \sigma(A)$ for any $A \in PR^i(H)$. First, it is clear that for any $\ell < 8$, Property $\ell$ holds for each index both in $\mathcal{F}_i$ and in the reversed fragmentation $\mathcal{F}_i^{\text{rev}}$, as otherwise the algorithm would branch on an index violating Property $\ell$. Thus, $L^{\max}(A_i, H_i) \geq R^{\min}(A_i, H_i)$ can not happen, as this would mean that there is a conflict-inducing index in $H_i$ for $(A_i, H_i)$, violating Property 7. This directly implies $1 \leq d^i(A)$.

Second, assume $d^i(A) = y_i^R - y_i^L > \sigma(A)$. This implies that $h = y_i^R - \sigma(A)$ is contained in $A_i$, but no vertex of $\mathcal{L}(A_i, H_i) \cup \mathcal{R}(A_i, H_i)$ starts in $h$. However, by Properties 3 and 5 for $y_i^R$, we know that some vertex in $M_r^+((y_i^R)_{\text{left}}) = M_r^+(h_{\text{right}})$ ends $H_i$. Using these properties for $h^{\text{rev}}$ in the reversed instance, we obtain that some vertex $v$ in $M_{r'}^+(h)$ must also end in $H_i$. By Property 5 for $h$, $v$ must be contained in one of the sets $\mathcal{L}(A_i, H_i)$, $\mathcal{R}(A_i, H_i)$, $\mathcal{X}(A_i, H_i)$. But $y_i^L < h < y_i^R$, so we obtain $v \in \mathcal{X}(A_i, H_i)$. Therefore, some index in $H_i$ violates Property 6, a contradiction.

Now, observe that for any $i \in [t]$, $j_i$ is LR-critical for some $(A_i, H_i)$ with $A \in PR^i(H)$. If $A \in PR^{i+1}(H)$ as well, then $d^{i+1}(A) > d^i(A)$. By our bounds on $d^i(A)$, this yields that there can be at most $\sigma(A)$ indices $i$ where $j_i$ is LR-critical for $(A_i, H_i)$. (Here we also used that $d^i(A)$ can not decrease, by its definition.) This clearly implies $t \leq \sum_{F \in P(H)} \sigma(F) = \delta(H)$.

To finish the proof, we show $\delta(H) \leq k$. Let $b'$ be the last index preceding the indices in $H$, and let $b = b' + \delta(H)$. Recall that $\phi_S(B_{r'}^+(1, b')) = B_r^+(1, b) \setminus S$ by Proposition 4.4.4. Using that Properties 1 and 3 hold for every index in $[m']$ and that $B_r^+(i) \neq \emptyset$ by Proposition 4.1.2

for any $i \in [m]$, we obtain

$$|B^+_{r'}(1,b')| + k \geq |B^+_r(1,b)| = \left| \bigcup_{1\leq j\leq b'} B^+_r(j_{\text{left}}) \right| + \left| \bigcup_{\substack{1\leq i\leq b,\\ i\notin\{j_{\text{left}}:1\leq j\leq b'\}}} B^+_r(i) \right|$$

$$\geq |B^+_{r'}(1,b')| + \sum_{F\in P(H)} \sigma(F) = |B^+_{r'}(1,b')| + \delta(H).$$

This shows $k \geq \delta(H)$, proving the lemma. ☐

Now, we can state the key properties of algorithm $\mathcal{A}$, which prove Theorem 4.3.1.

**Lemma 4.4.13.** *Given an input $(G', G)$ where $|V(G')| = n$ and $|V(G)| = n+k$, algorithm $\mathcal{A}$ either produces a reduced input in $O(n)$ time, or branches into at most $k^{O(k^3)}$ such that in each branch it either correctly refuses the instance, or outputs an independent subproblem or a necessary set of size at most $2k+1$. Moreover, each branch takes $O(n)$ time.*

*Proof.* Let us overview the steps of algorithm $\mathcal{A}$. First, it tries to apply the reduction rules described in Section 4.3.2. In this phase, it either outputs a reduced input in linear time, or it may branch into at most $(4k+1)2^{4k}(k(7k/2+8)+1) = 2^{O(k)}$ branches. In each branch it either correctly outputs 'No', outputs a necessary set of size at most 2, or outputs an independent subproblem having parameter at most $k-1$ but at least 1. These steps can be done in linear time, as argued in Section 4.3.2.

If none of the reductions in Section 4.3.2 can be applied, then $\mathcal{A}$ first checks whether a reduced input can be output by using Lemma 4.4.2. If not, then it branches into 3 directions, according to whether $S$ is local, and if not, whether the children of $r'$ should be reversed to achieve the properties of Lemma 4.4.1. In the first branch, it outputs a necessary set of size at most 2. In the other two branches, it checks whether the annotated fragmentation $AF_0$ produced in the beginning is proper. While the annotated fragmentation is not proper, $\mathcal{A}$ chooses the smallest $\ell$ and the smallest index $j$ violating Property $\ell$ (maybe in the reversed instance), and branches into at most $2(k+1)^2-1$ directions. In these branches, $\mathcal{A}$ either modifies the actual annotated fragmentation or stops by outputting an independent subproblem, a necessary set of size at most $2k+1$, or rejecting.

Let us consider a sequence of $t$ such branchings performed by $\mathcal{A}$, and let $AF_0, AF_1, \dots, AF_t$ be the sequence of annotated fragmentations produced in this process. (We interpret these as annotated fragmentations for $(T, T', S)$ and not for $(T^{\text{rev}}, T'^{\text{rev}}, S)$.) Let us call a continuous subsequence $\mathcal{S}$ of $AF_0, AF_1, \dots, AF_t$ a *segment*, if each annotated fragmentation in $\mathcal{S}$ has the same number of non-trivial fragments, and $\mathcal{S}$ is maximal with respect to this property. By Lemma 4.4.5, the algorithm can reject if there are more than $2k$ non-trivial fragments in a fragmentation, so $AF_0, AF_1, \dots, AF_t$ can contain at most $2k$ segments. Let $\mathcal{S} = AF_{t_1}, AF_{t_1+1}, \dots, AF_{t_2}$ be such a segment. Clearly, each $AF_h$ $(t_1 < h \leq t_2)$ is obtained from $AF_{h-1}$ by performing a right split either in the original or in the reversed instance (the latter meaning that $AF_h^{\text{rev}}$ is obtained from $AF_{h-1}^{\text{rev}}$ by a right split).

Let $AF_p$ be the first 9-proper annotated fragmentation in the segment. By Lemma 4.4.12, each subsequence of $AF_{t_1}, \dots, AF_p$ where the measure does not increase can have length at most $k$. (The measure of an annotated fragmentation is the measure of its fragmentation.) By (2) of Lemma 4.4.12, $AF_{t_1}$ has a non-trivial fragment containing each of those indices for which the algorithm performed a branching (because of Property 8) in some $AF_h$, $t_1 \leq h \leq p$. Taking into account that the number of non-trivial fragments can not exceed $2k$, but branchings can also happen in the reversed instance, we obtain that there can be at most $4k$ maximal subsequences in $AF_{t_1}, \dots, AF_p$ of length at least 2 where the measure is constant. Using Lemma 4.4.5, we get that $\mu(AF_p) \leq 36k$, implying $p \leq t_1 + 4k^2 + 36k$.

Clearly, $\mathcal{A}$ obtains $AF_{p+1}, AF_{p+2}, \ldots, AF_{t_2}$ while trying to ensure Property 10, by performing right splits in the original instance. Observe that if $\mathcal{A}$ obtains $AF_h$ ($p+1 \leq h \leq t_2$) by applying a right split at $j$, then by the choice of $j$, Property 10 holds for each index $j' \leq j$ in any $AF_{h'}$ where $h' \geq h$. This, together with Lemma 4.4.5 implies that $\mathcal{A}$ can perform at most $2k$ such branchings, implying that $t_2 \leq p + 2k \leq t_1 + 4k^2 + 38k$. Altogether, this implies $t \leq 2k(4k^2 + 38k)$, proving that the maximum length of a sequence of branchings performed by $\mathcal{A}$ in order to obtain a proper annotated fragmentation can be at most $8k^3 + 76k^2 = O(k^3)$.

Essentially, this means that the search tree that $\mathcal{A}$ investigates has height at most $O(k^3)$. Since one branching results in at most $2(k+1)^2 - 1$ directions, we obtain that the total number of resulting branches in a run of algorithm $\mathcal{A}$ can be bounded by some functions $f(k) = k^{O(k^3)}$. In each of these branches, if $\mathcal{A}$ does not stop, then it has a proper annotated fragmentation $(\mathcal{F}, U)$. After this, algorithm $\mathcal{A}$ does not perform any more branchings. Instead, it applies Lemma 4.4.10 repeatedly. If the algorithm reaches a state where Lemma 4.4.10 does not apply, then it outputs a solution in linear time using Lemma 4.4.11.

It is easy to verify that each branch can be performed in linear time. The only non-trivial task is to show that the repeated application of Lemma 4.4.10 can be implemented in linear time, but this easily follows from the fact that none of the conditions of Lemma 4.4.10 can be applied twice for a block $[a, b]$. □

## 4.4.3 The proof of Lemma 4.4.7

In this section we prove Lemma 4.4.7. Suppose that Property $\ell$ ($1 \leq \ell \leq 10$) does not hold for some $j \in [m']$ in the annotated fragmentation $(\mathcal{F}, U)$, but all the previous properties hold for each index both in $(\mathcal{F}, U)$ and in $(\mathcal{F}^{\text{rev}}, U^{\text{rev}})$. Suppose also that $j$ is left-aligned, i.e. $I_S(j) = [j_{\text{left}}, j_{\text{left}}]$. Below we describe the detailed steps of algorithm $\mathcal{A}$ depending on the property that is violated by $j$.

**Property 1:** $G'[X'_j]$ is isomorphic to $G[X_{j_{\text{left}}}]$.

If $j$ violates Property 1, then $G'[X'_j]$ is not isomorphic to $G[X_{j_{\text{left}}}]$, which implies $S \cap X_{j_{\text{left}}} \neq \emptyset$. From $I_S(j) = [j_{\text{left}}, j_{\text{left}}]$ we obtain that $S \cap X_{j_{\text{left}}}$ must be a solution for $(G'[X'_j], G[X_{j_{\text{left}}}])$. Conversely, if $(G', G)$ is solvable, then any solution for $(G'[X'_j], G[X_{j_{\text{left}}}])$ can be extended to a solution for $(G', G)$. By $m > m'$, $G - X_{j_{\text{left}}}$ can not be isomorphic to $G' - X'_j$, so $S \subseteq X_{j_{\text{left}}}$ is not possible. Therefore, if the parameter of $(G'[X'_j], G[X_{j_{\text{left}}}])$ is more than $k-1$ (or less than 1), then the algorithm can refuse the instance. Thus, $\mathcal{A}$ can either **reject**, or it can output the **independent subproblem** $(G'[X'_j], G[X_{j_{\text{left}}}])$.

**Property 2:** $|M^+_{r'}(j)| \leq |M^+_r(j_{\text{left}})| \leq |M^+_{r'}(j)| + k$ and $|M^-_{r'}(j)| \leq |M^-_r(j_{\text{left}})| \leq |M^-_{r'}(j)| + k$.

By $I_S(j) = [j_{\text{left}}, j_{\text{left}}]$ and Proposition 4.4.4, we can observe that $M^+_r(j_{\text{left}}) \setminus S = \phi_S(M^+_{r'}(j))$ and $M^-_r(j_{\text{left}}) \setminus S = \phi_S(M^-_{r'}(j))$. If $j$ violates Property 2, then this contradicts to $|S| \leq k$, and thus algorithm $\mathcal{A}$ can **reject**.

**Lemma 4.4.14.** *If Properties 1 and 2 hold for each index both in $(\mathcal{F}, U)$ and in $(\mathcal{F}^{\text{rev}}, U^{\text{rev}})$, and there is an index $h \in [m']$ contained in a non-trivial fragment $F$ such that $|M^+_{r'}(h)| > k$ or $|M^-_{r'}(h)| > k$, then there is no solution for $(G', G)$.*

*Proof.* As Property 2 holds for each index in $F = ([a', b'], [a, b])$, $|M^+_{r'}(j)| \leq |M^+_r(j_{\text{left}})|$ holds for each $j \in [m']$. Similarly, as Property 2 holds for each index in the reversed instance, we obtain that $|M^+_{r'}(j)| \leq |M^+_r(j_{\text{right}})|$ must hold for each $j \in [m']$. Supposing $|M^+_{r'}(h)| > k$, we get

$$\sum_{a \leq i \leq b} |M^+_r(i)| = \sum_{a' \leq j < h} |M^+_r(j_{\text{left}})| + \sum_{0 \leq d < \sigma(F)} |M^+_r(h_{\text{left}} + d)|$$

$$+ \sum_{h \leq j \leq b'} |M^+_r(j_{\text{right}})| \geq |M^+_{r'}(h)| + \sum_{a' \leq j \leq b'} |M^+_{r'}(j)| > k + \sum_{a' \leq j \leq b'} |M^+_{r'}(j)|.$$

Observe that we used $\sigma(F) > 0$ in the first inequality.

Proposition 4.4.4 yields $\phi_S(B^+_{r'}(a', b')) = B^+_r(a, b) \setminus S$, implying $|B^+_r(a, b)| \leq |B^+_{r'}(a', b'))| + k$. Using that Property 1 holds for each index, we also have $|X'_j| = |X_{j_{\text{left}}}|$ for each $j \in [m']$, implying $\sum_{a \leq i \leq b} |X_i| \geq \sum_{a' \leq j \leq b'} |X'_j|$. Hence, we obtain

$$\sum_{a \leq i \leq b} |M^+_r(i)| \leq \sum_{a' \leq j \leq b'} |M^+_{r'}(j)| + k,$$

contradicting the above inequality. The case $|M^-_{r'}(h)| > k$ can be handled in the same way. □

> **Property 3:** If $j$ is non-trivial, then $|M^+_{r'}(j)| = |M^+_r(j_{\text{left}})|$ and $|M^-_{r'}(j)| = |M^-_r(j_{\text{left}})|$.

By $I_S(j) = [j_{\text{left}}, j_{\text{left}}]$ and Proposition 4.4.4, we obtain that $M^+_r(j_{\text{left}}) \setminus S = \phi_S(M^+_{r'}(j))$ and $M^-_r(j_{\text{left}}) \setminus S = \phi_S(M^-_{r'}(j))$. Clearly, if $|M^+_r(j_{\text{left}})| < |M^+_{r'}(j)|$ or $|M^-_r(j_{\text{left}})| < |M^-_{r'}(j)|$, then algorithm $\mathcal{A}$ can output 'No'. If this is not the case, then $S$ must contain at least one vertex from $M^+_r(j_{\text{left}})$ or $M^-_r(j_{\text{left}})$, because $j$ violates Property 3. If $|M^+_{r'}(j)| > k$ or $|M^-_{r'}(j)| > k$, then $\mathcal{A}$ can output 'No' as well, by Lemma 4.4.14. Thus, if $\mathcal{A}$ does not **reject**, then it can output a **necessary set** of size at most $k + 1$ in both cases, by taking $|M^+_{r'}(j)| + 1$ or $|M^-_{r'}(j)| + 1$ arbitrary vertices from $M^+_r(j_{\text{left}})$ or $M^-_r(j_{\text{left}})$, respectively.

> **Property 4:** If $j$ is non-trivial, then $|L_{r'}(y, j)| = |L_r(y_{\text{left}}, j_{\text{left}})|$ for any $y < j$ contained in the same fragment as $j$.

Suppose that $|L_{r'}(y, j)| \neq |L_r(y_{\text{left}}, j_{\text{left}})|$ for some $y < j$ contained in the same fragment that contains $j$. Since $j$ is left-aligned, we get that $y$ must also be left-aligned as well by $y < j$, i.e. $I_S(y) = [y_{\text{left}}, y_{\text{left}}]$. By Proposition 4.4.4, this implies $L_r(y_{\text{left}}, j_{\text{left}}) \setminus S = \phi_S(L_{r'}(y, j))$. Thus, if $|L_{r'}(y, j)| > |L_r(y_{\text{left}}, j_{\text{left}})|$ $L_r(y_{\text{left}}, j_{\text{left}})$ contains at least one vertex from $S$. Since each vertex in $L_r(y_{\text{left}}, j_{\text{left}})$ has the same neighborhood, $\mathcal{A}$ can output $\{s\}$ as a **necessary set** for some arbitrarily chosen $s$ in $L_r(y_{\text{left}}, j_{\text{left}})$.

> **Property 5:** If $j$ is non-trivial, then for every $(v, w) \in P^+_{\text{left}}(j)$ where $Q^{\text{right}}_{r'}(v) = y$ is non-trivial, $y_{\text{left}} \leq Q^{\text{right}}_r(w) \leq y_{\text{right}}$ holds. Also, for every $(v, w) \in P^-_{\text{left}}(j)$ such that $Q^{\text{left}}_{r'}(v) = y$ is non-trivial, $y_{\text{left}} \leq Q^{\text{left}}_r(w) \leq y_{\text{right}}$ holds.

Suppose that $j$ violates Property 5, because $(v, w) \in P^+_{\text{left}}(j)$ such that $Q^{\text{right}}_{r'}(v) = y$ is non-trivial, but $y_{\text{left}} \leq Q^{\text{right}}_r(w) \leq y_{\text{right}}$ does not hold. We show that $\mathcal{A}$ can output 'No' in this case. As Property 3 holds for $j$, $|M^+_{r'}(j)| = |M^+_r(j_{\text{left}})|$. As $j$ is left-aligned, $\phi_S(v) = w$ by Proposition 4.4.6. But from this, Proposition 4.4.4 implies $\alpha_S(y) \leq Q^{\text{right}}_r(w) \leq \beta_S(y)$. By Proposition 4.4.4 we know $y_{\text{left}} \leq \alpha_S(y) \leq \beta_S(y) \leq y_{\text{right}}$ as well. Therefore, $\mathcal{A}$ can indeed refuse the instance. Supposing that Property 5 does not hold because of the case where some $(v, w) \in P^-_{\text{left}}(j)$ is considered leads to the same result, so it is straightforward to verify that $\mathcal{A}$ can **reject** in both cases.

The observation below, used in the forthcoming three cases, is easy to see:

**Proposition 4.4.15.** *Suppose that the first five properties hold for a given (annotated) fragmentation. Let $y$ and $j$ be indices of $[m']$ contained in non-trivial fragments $F$ and $H$, respectively, and suppose that $j$ is left-aligned. Then $v \in L_{r'}(y,j)$ implies the followings.*
*(1) $v \in \mathcal{L}(F,H) \cup \mathcal{R}(F,H) \cup \mathcal{X}(F,H)$.*
*(2) If $v \in \mathcal{L}(F,H)$, then $\alpha_S(y) = y_{\text{left}}$.*
*(3) If $v \in \mathcal{R}(F,H)$, then $\beta_S(y) = y_{\text{right}}$.*
*(4) If $v \in \mathcal{X}(F,H)$, then $y$ is either wide or skew.*

**Property 6:** If $j$ is non-trivial, then no vertex in $\mathcal{X}(F,H)$ (for some $F$ and $H$) ends in $j$.

Suppose that Property 6 does not hold for $j$, so there is a vertex in $L_{r'}(y,j) \cap \mathcal{X}(F,H)$ for some $y < j$. As $j$ is left-aligned, Proposition 4.4.15 implies that $y$ is either **wide or skew**.

**Property 7:** $j$ is not conflict-inducing for any $(F,H)$.

Suppose that $j$ violates Property 7 because it is conflict-inducing for some $(F,H)$ and for some conflicting pair of indices $(y_1,y_2)$. Let $j_1$ be the minimal index for which $L_{r'}(y_1,j_1) \cap \mathcal{R}(F,H) \neq \emptyset$, and let $j_2$ be the minimal index for which $L_{r'}(y_2,j_2) \cap \mathcal{L}(F,H) \neq \emptyset$. Since $j \geq \max\{j_1,j_2\}$, and $j$ is left-aligned, we know that both $j_1$ and $j_2$ are left-aligned as well. By Proposition 4.4.15, this implies $\beta_S(y_1) = y_{1\text{right}}$ and $\alpha_S(y_2) = y_{2\text{left}}$. If $y_1 < y_2$, then this yields a contradiction by Proposition 4.4.4, so $\mathcal{A}$ can **reject**. In the case where $y_1 = y_2 = y$, we get $I_S(y) = [y_{\text{left}}, y_{\text{right}}]$, and since $y$ is non-trivial, algorithm $\mathcal{A}$ can output $y$ as a **wide index**.

For the case of Property 8, we need the following simple lemma:

**Lemma 4.4.16.** *Suppose that a fragmentation for $(T,T',S)$ contains a fragment $F = ([a',b'],[a,b])$ with $0 < b'-a' \leq \sigma(F)$, and the first four properties hold for each index contained in $F$ both in the given fragmentation and its reversed version. Then $\mathcal{A}$ can produce a necessary set of size at most $2k+1$.*

*Proof.* Since Properties 1 and 3 hold for each index contained in $F$, we obtain $|B^+_{r'}(a',b')| = |B^+_r(a'_{\text{left}}, b'_{\text{left}})|$. Using Proposition 4.4.4 we have $B^+_r(a,b) \setminus S = \phi_S(B^+_{r'}(a',b'))$. Proposition 4.1.2 yields $B^+_{r'}(j) \neq \emptyset$ for any $j$, so we get $|B^+_r(a,b)| > |B^+_{r'}(a',b')|$. Hence, fixing an arbitrary set $N \subseteq B^+_r(a,b)$ of size $|B^+_{r'}(a',b')|+1$, we get that $N$ is a nonempty necessary set. We claim $|B^+_{r'}(a',b')| \leq 2k$, which implies $|N| \leq 2k+1$. Thus, $\mathcal{A}$ can indeed output $N$, proving the lemma.

It remains to show $|B^+_{r'}(a',b')| \leq 2k$. Recall $|B^+_{r'}(a',b')| = |B^+_r(a'_{\text{left}}, b'_{\text{left}})|$. As Properties 1 and 3 hold for each index contained in $F^{\text{rev}}$ in the reversed fragmentation, we get $|B^+_{r'}(a',b')| = |B^+_r(a'_{\text{right}}, b'_{\text{right}})|$ as well. Using $a'_{\text{right}} - b'_{\text{left}} = a' - b' + \sigma(F) \geq 0$, we obtain that $B^+_r(a'_{\text{left}}, b'_{\text{left}}) \cap B^+_r(a'_{\text{right}}, b'_{\text{right}}) \subseteq B^+_r(b'_{\text{left}})$. Moreover, if $b'-a' < \sigma(F)$ also holds, then actually $B^+_r(a'_{\text{left}}, b'_{\text{left}}) \cap B^+_r(a'_{\text{right}}, b'_{\text{right}}) = \emptyset$.

By the above paragraph, $b'-a' < \sigma(F)$ implies $|B^+_r(a,b)| \geq 2|B^+_{r'}(a',b')|$, therefore we get $|B^+_{r'}(a',b')| \leq k$. However, $b'-a' = \sigma(F)$ yields $|B^+_{r'}(a',b')| + k \geq |B^+_r(a,b)| \geq 2|B^+_{r'}(a',b')| - |B^+_r(b'_{\text{left}})|$, implying $|B^+_{r'}(a',b')| \leq k + |B^+_r(b'_{\text{left}})|$. Taking into account that $|B^+_{r'}(a')| = |B^+_r(b'_{\text{left}})| = |B^+_{r'}(b')|$ holds by Properties 1 and 3 for $b'$ and for $a'^{\text{rev}}$, we also have $|B^+_{r'}(a',b')| \geq 2|B^+_r(b'_{\text{left}})|$. Summarizing all these, $|B^+_{r'}(a',b')| \leq 2k$ follows. □

**Property 8:** $j$ is not LR-critical for any $(F,H)$.

Suppose $j$ violates Property 8, meaning that $j$ is LR-critical for some $(F,H)$. In this case, $R^{\min}(F,H) = y^R$ is an index contained in $F$. Since $j$ is left-aligned, the R-critical index for $(F, H)$ is also left-aligned, hence Proposition 4.4.15 yields $\beta_S(y^R) = {y^R}_{\text{right}}$. Let $a'$ be the first index of $[m']$ contained in $F$.

First, if $y^R < a' + \sigma(F)$, then we apply Lemma 4.4.16 as follows. Clearly, by $\beta_S(y^R) = {y^R}_{\text{right}}$ we can perform a right split at $y^R$. The obtained fragmentation will contain the fragment $F' = ([a', y^R], [{a'}_{\text{left}}, {y^R}_{\text{right}}])$, so $y^R - a' < \sigma(F) = \sigma(F')$ shows that $\mathcal{A}$ can produce a **necessary set** of size at most $2k+1$ by using Lemma 4.4.16.

Now, suppose $y^R \geq a' + \sigma(F)$. In this case, there is an index $t$ in $F$ for which $t_{\text{right}} = {y^R}_{\text{left}}$. By Properties 3 and 5 for $y^R$, we know that there is a vertex in $M_r^+({y^R}_{\text{left}})$ that ends in the fragment $H$. Using Properties 3 and 5 again for $t^{\text{rev}}$ in the reversed instance, we know that there must be a vertex $v$ in $M_{r'}^+(t)$ that ends in the fragment $H$. By Proposition 4.4.15, $v \in \mathcal{L}(F, H) \cup \mathcal{R}(F, H) \cup \mathcal{X}(F, H)$. Observe that $v \notin \mathcal{X}(F, H)$, as Property 6 holds for every index in $[m']$. Also, $v \notin \mathcal{R}(F, H)$ by the definition of $y^R = R^{\min}(F, H)$. Thus, we know that $v \in \mathcal{L}(F, H)$, implying $y^L = L^{\max}(F, H) \geq t$ as well. As Property 7 holds for each index, we also have $y^L < y^R$.

To finish the case, observe that since $j$ is left-aligned and LR-critical for $(F, H)$, Proposition 4.4.15 yields $\alpha_S(y^L) = {y^L}_{\text{left}}$. Using $\beta_S(y^R) = {y^R}_{\text{right}}$ again, we can produce a fragmentation for $(T, T', S)$ that contains the fragment $F' = ([y^L, y^R], [{y^L}_{\text{left}}, {y^R}_{\text{right}}])$. (This can be thought of as performing a right split at $y^R$, and a right split at $(y^L)^{\text{rev}}$ in the reversed instance.) Hence, $y^R - y^L \leq y^R - t = \sigma(F) = \sigma(F')$ shows that $\mathcal{A}$ can produce a **necessary set** of size at most $2k+1$ by using Lemma 4.4.16.

> **Property 9:** If $j$ is non-trivial, then for every $(v, w) \in P^+_{\text{left}}(j)$ where $Q^{\text{right}}_{r'}(v) = y$ is non-trivial, $Q^{\text{right}}_{r}(w) = y_{\text{left}}$ holds. Also, for every $(v, w) \in P^-_{\text{left}}(j)$ such that $Q^{\text{left}}_{r'}(v) = y$ is non-trivial, $Q^{\text{left}}_{r}(w) = y_{\text{left}}$ holds.

Observe that if Property 9 does not hold for an index $j$, then by Proposition 4.4.15, either $M^+_{r'}(j)$ or $M^-_{r'}(j)$ contains a vertex in $\mathcal{R}(F, H) \cup \mathcal{X}(F, H)$ for some $(F, H)$. But this means that one of Properties 6 and 8 must be violated, which is a contradiction. Thus, $\mathcal{A}$ can correctly **reject**.

> **Property 10:** If $j$ is non-trivial, then for each important trivial index $u \in U$, $|L_{r'}(j, u)| = |L_r(j_{\text{left}}, u_{\text{left}})|$ holds if $u > j$, and $|L_{r'}(u, j)| = |L_r(u_{\text{left}}, j_{\text{left}})|$ holds if $u < j$.

Suppose that $j$ violates Property 10, because $|L_{r'}(j, u)| \neq |L_r(j_{\text{left}}, u_{\text{left}})|$ for some $u > j$. (The case when $u < j$ can be handled in the same way.) Since $u$ is contained in a trivial fragment, $I_S(u) = [u_{\text{left}}, u_{\text{left}}]$. Thus, by $I_S(j) = [j_{\text{left}}, j_{\text{left}}]$ and Proposition 4.4.4, we get $L_r(j_{\text{left}}, u_{\text{left}}) \setminus S = \phi_S(L_{r'}(j, u))$. If $|L_{r'}(j, u)| > |L_r(j_{\text{left}}, u_{\text{left}})|$, then $\mathcal{A}$ can **reject** the instance. Otherwise, we can argue as before that $\{s\}$ is a **necessary set** for any $s \in L_r(j_{\text{left}}, u_{\text{left}})$.

CHAPTER 5

# Stable matching with ties

In this chapter, we consider numerous variants of the Stable Marriage with Ties and Incomplete Lists (or shortly, SMTI) problem. We investigate the parameterized complexity of finding a maximum stable matching for an instance of SMTI with different parameterizations such as the number of ties, the maximum or the total length of ties present. We also study the possibilities for giving a permissive local search algorithm for this problem. In addition, we present strong FPT-inapproximability results for two optimization problems related to SMTI.

The input of the SMTI problem is a triple $(X, Y, L)$. Here $X$ and $Y$ are sets of *women* and *men*, respectively. A $p \in X \cup Y$ is a *person*, and for each person $a$ we define $\mathcal{O}(a)$ to be the set containing the members of the opposite sex.

We describe the preferences of a person $a$ with the *preference list* $L(a)$. The precedence list $L(a)$ is an ordered list containing the acceptable partners for $a$. Since ties may be involved, the ordering of these lists is not necessarily strict. We assume that acceptance is mutual, i.e. either $a$ and $b$ are both acceptable for each other, forming an *acceptable pair*, or both of them find each other unacceptable. The preference lists of an instance determine the *ranking function* $\rho^L : (X \times Y) \cup (Y \times X) \to \mathbb{N} \cup \{\infty\}$, describing the ranking of the members of the opposite sex for each person. For some $b \in \mathcal{O}(a)$, if $b$ is not contained in $L(a)$ then we let $\rho^L(a, b) = \infty$, and if $b$ is contained in $L(a)$ then we define $\rho^L(a, b)$ as the (possibly joint) ranking of $b$ in $L(a)$ (i.e. one plus the number of persons strictly preceding $b$ in $L(a)$). When no confusion arises, we may leave the superscript $L$, and only write $\rho$ for a given ranking function. We say that *$a$ prefers $b$ to $c$* if $\rho(a, b) < \rho(a, c)$.

Ties can be present, meaning that $\rho(a, b) = \rho(a, c)$ is possible even if $b \neq c$. Formally, a *tie* with respect to $a$ is a set $T \subseteq \mathcal{O}(a)$ of maximum cardinality such that $|T| \geq 2$ and $\rho(a, t_1) = \rho(a, t_2) \neq \infty$ for every $t_1, t_2 \in T$. A person $a$ is *indifferent*, if there exists a tie w.r.t. $a$, and the *length* of a tie $T$ is $|T|$.

For an instance $I$ of SMTI, we will use the following parameterization functions:

- $\kappa_1(I)$ denotes the number of ties in $I$.
- $\kappa_2(I)$ denotes the maximum length of a tie in $I$.
- $\kappa_3(I)$ denotes the *total length of the ties* in $I$, which is the sum of the length of each tie in the instance. Clearly, $\kappa_3(I) \leq \kappa_1(I)\kappa_2(I)$.

A *matching* for $(X, Y, L)$ is a subset $M$ of the acceptable pairs w.r.t. $L$, where $|\{q \mid pq \in M\}| \leq 1$ for each person $p$. If $xy \in M$, then we say that $x$ and $y$ are *covered* by $M$, $M$ *assigns* $y$ to $x$ and vice versa, which will be denoted by $M(x) = y$ and $M(y) = x$. We will use the notation $M(x) = \oslash$ for the case when $x$ is not covered by $M$, and we also extend $r$ such that $\rho(p, \oslash) = \infty$ for each person $p$. The *size* of a matching $M$, denoted by $|M|$, is the number of pairs contained in $M$. We say that a pair $xy$ is a *blocking pair for M* if $\rho(x, y) < \rho(x, M(x))$ and $\rho(y, x) < \rho(y, M(y))$, i.e. both $x$ and $y$ *strictly* prefer each other to their partner in $M$ (if existent). A matching is *stable* if no blocking pair exists for it. The task of the SMTI problem is to find a stable matching, if existing.

Although it is known that a stable matching can always be found for every instance of SMTI by applying the Gale-Shapley algorithm [60, 72], there are several problems connected to SMTI that are much harder. In the Maximum Stable Marriage with Ties and Incomplete Lists problem (MaxSMTI), the task is to find a stable matching of maximum size. In Section 5.1, we study the parameterized complexity of this problem, with respect to the parameterization functions $\kappa_1, \kappa_2$, and $\kappa_3$ defined above. In Section 5.2, we examine the possibilities for giving a local search algorithm for MaxSMTI. We present results stating that a local search algorithm for this problem cannot have FPT running time with certain parameterizations, assuming FPT $\neq$ W[1].

In Section 5.3, we investigate two problems where we aim to find stable matchings that may not maximize the size of the matching, but minimize some cost function instead. Both of these problems (namely, finding an egalitarian or a minimum regret stable matching) are polynomial-time solvable if no ties are allowed, but are inapproximable by polynomial-time algorithms in a strong sense otherwise. We examine the possibilities of giving an FPT approximation algorithm [102] for these problems. Such algorithms provide an approximation of the optimal solution within running time that is not polynomial but FPT, when considering some natural parameters.

The results of this chapter were published in [105]. Section 5.4 contains a summary of these results.

## 5.1 Parameterized complexity

If the preference lists are complete, meaning that each person finds every member of the opposite gender acceptable, or if no tie can be contained in the preference lists, then every stable matching must have the same size [72]. But if both ties and incomplete preference lists may occur, then stable matchings of different sizes may exist for a given instance [98]. The following problem, called Maximum Stable Marriage with Ties and Incomplete Lists (or shortly MaxSMTI), has been shown to be NP-hard [84]: given an instance $I$ of SMTI and an integer $s$, find a stable matching for $I$ of size at least $s$.

Moreover, it has been proven in [98] that MaxSMTI is NP-complete even in the special case when only women can be indifferent, each tie has length 2, and ties are only present at the end of the preference lists (i.e. if $t \in T$ for a tie $T$ w.r.t. $a$, then $\rho(a, x) > \rho(a, t)$ implies $\rho(a, x) = \infty$). Thus, there is no hope to give an FPT algorithm for MaxSMTI with parameterization $\kappa_2$, since the problem is NP-hard even if the parameter has value 2.

However, if no ties are involved in an instance of MaxSMTI, then a stable matching of maximum size can be found in linear time with an extension of the Gale-Shapley algorithm [60, 72]. This can be used when the total length of ties (i.e. $\kappa_3(I)$) is small for some instance $I$, since we can apply a brute force algorithm that breaks ties in all possible ways and finds a stable matching of maximum size for all the instances obtained.

**Theorem 5.1.1.** MaxSMTI *is FPT with parameterization* $\kappa_3$.

*Proof.* Let $I = (X, Y, L)$ be the instance given. We use a method for breaking ties as follows. Formally, let $I$ contain the ties $\{T_i \mid i \in [\kappa_1(I)]\}$, and let $t_i^j$ denote the $j$-th element of the tie $T_i$ (according to some fixed order). If $\pi_i$ is a bijection from $T_i$ to $[|T_i|]$ (i.e. a permutation of $T_i$) for each $i \in [\kappa_1(I)]$, then the instance $(X, Y, L')$ can be obtained from $I$ by *breaking ties* according to $(\pi_1, \dots, \pi_{\kappa_1(I)})$, if the ranking function $\rho^{L'}$ is defined such that $\rho^{L'}(a, b) < \rho^{L'}(a, c)$ if and only if either $\rho^L(a, b) < \rho^L(a, c)$ or $b$ and $c$ are both in the tie $T_i$ w.r.t. $a$ and $\pi_i(b) < \pi_i(c)$.

To produce a solution, we break ties in various ways. Let $P_i = \{\pi_i^j \mid j \in [|T_i|]\}$ where $\pi_i^j$ is an arbitrary bijection from $T_i$ to $[|T_i|]$ for which $\pi_i^j(t_i^j) = 1$ holds, i.e. $\pi_i^j$ puts the $j$-th element of $T_i$ in the first place. Using this, we break ties according to each element of $P_1 \times \dots \times P_{\kappa_1(I)}$, apply the Gale-Shapley algorithm for each instance obtained, and then output the stable matching having maximum size among the set $\mathcal{M}$ of stable matchings obtained.

We claim that all stable matchings for $I$ can be obtained as a stable matching of an instance obtained from $I$ by breaking ties this way. It is easy to verify that any matching in $M \in \mathcal{M}$ is a stable matching for $I$. Conversely, if $M$ is a stable matching for $I$, then it is also stable in the instance obtained by breaking ties according to $(\pi_1, \dots, \pi_{\kappa_1(I)})$ where $\pi_i = \pi_i^j$ if $T_i$ is a tie with respect to some $a$ such that $M(a) = t_i^j$, otherwise $\pi_i$ can be any permutation from $P_i$. Thus, $M$ is contained in $\mathcal{M}$.

Clearly, as we have to break ties in at most

$$\prod_{i \in [\kappa_1(I)]} |T_i| \le \left( \frac{\sum_{i \in [\kappa_1(I)]} |T_i|}{\kappa_1(I)} \right)^{\kappa_1(I)} = \left( \frac{\kappa_3(I)}{\kappa_1(I)} \right)^{\kappa_1(I)} \le \kappa_3(I)^{\kappa_3(I)}$$

many ways, this method yields a solution in $O(\kappa_3(I)^{\kappa_3(I)} \cdot |I|)$ time. □

Theorem 5.1.1 immediately raises the question of whether MaxSMTI is FPT if the parameter is the number of ties ($\kappa_1$). As claimed by Theorem 5.1.2, this problem turns out to be hard. The proof of this theorem can be found in Section 5.2. The reason for this is that the proof of Theorems 5.1.2 relies on the same construction as the proof of Theorem 5.2.1, so we will prove them simultaneously.

**Theorem 5.1.2.** *The decision version of* MaxSMTI *is W[1]-hard with parameterization $\kappa_1$, even if only women can be indifferent.*

## 5.2 Local search

In this section, we investigate the possibility of giving an efficient permissive local search algorithm for MaxSMTI. Recall that the objective function to be maximized in the MaxSMTI problem is the size of the stable matching. We define the distance of two stable matchings $M_1$ and $M_2$ for $I$ as the number of persons $p$ in $I$ such that $M_1(p) \neq M_2(p)$. We denote this value by $d(M_1, M_2)$. Accordingly, the task of a permissive local search algorithm for MaxSMTI, as defined in Chapter 1.1, is the following: given an instance $I$ of MaxSMTI, a stable matching $M_0$ for $I$, and an integer $\ell$, if there is a stable matching $M$ for $I$ with $|M| > |M_0|$ and $d(M_0, M) \le \ell$, then find any stable matching $M'$ for $I$ with $|M'| > |M|$.

Theorem 5.2.1 shows that no permissive local search algorithm can run in FPT time (assuming W[1] $\neq$ FPT), even if we regard not only the number of ties but also $\ell$ as a parameter for some input $(I, S_0, \ell)$.

**Theorem 5.2.1.** *If W[1] $\neq$ FPT, then there is no permissive local search algorithm for the* MaxSMTI *problem that runs in FPT time with combined parameters $(\kappa_1(I), \ell)$, even if only women can be indifferent.*

*Proof of Theorems 5.1.2 and 5.2.1.* Let $G(V,E)$ be the input graph and $k$ be the parameter for the Clique problem. We are going to construct an SMTI instance $I = (X,Y,L)$ with $\kappa_1(I) = \binom{k}{2} + k + 1$ ties, each being in the preference list of a woman, together with a stable matching $M_0$ for $I$ of size $|X| - 1$ such that the following statements are equivalent:

(1) $I$ has a stable matching of size at least $|M_0| + 1$, and

(2) there is a clique of size $k$ in $G$.

This immediately yields an FPT-reduction from Clique to MaxSMTI with parameterization $\kappa_1$, proving Theorem 5.1.2. Moreover, we will also show that every stable matching of size at least $|M_0| + 1$ must be $\ell$-close to $M_0$ for $\ell = 6\binom{k}{2} + 4k + 4$. Therefore, a permissive local search algorithm for MaxSMTI can be used to detect whether $I$ has a stable matching of size at least $|M_0| + 1$, i.e. whether $G$ has a clique of size $k$. Therefore, this construction also proves Theorem 5.2.1.

By the nature of the SMTI problem, the presented reduction is more complex than a typical reduction that proves hardness for some graph theoretic problem, since we have to describe the preference list for each person of the constructed instance. To ease the understanding, we illustrate the construction in Figure 5.1 by depicting the bipartite graph underlying the instance, where persons are represented by nodes and we connect two nodes if and only if the corresponding persons are acceptable for each other. Moreover, we use edge weights to represent ranks, and we use bold edges to mark the edges of a given matching.

We write $V(G) = \{v_1, v_2, \ldots, v_n\}$ and $m = |E(G)|$. To define $I = (X,Y,L)$, we construct a *node-gadget* $\mathcal{G}^i$ for each $i \in [k]$, an *edge-gadget* $\mathcal{G}^{i,j}$ for each $(i,j) \in \binom{[k]}{2}$, and a *path-gadget* $\mathcal{P}$. The node-gadget $\mathcal{G}^i$ consists of women $X^i \cup \{x_0^i\}$ with $X^i = \{x_u^i \mid u \in [n]\}$ and men $Y^i \cup \{y_0^i\}$ with $Y^i = \{y_u^i \mid u \in [n]\}$. Similarly, the edge-gadget $\mathcal{G}^{i,j}$ consists of women $X^{i,j} \cup \{x_0^{i,j}\}$ with $X^{i,j} = \{x_{u,z}^{i,j} \mid u < z, v_u v_z \in E(G)\}$ and men $Y^{i,j} \cup \{y_0^{i,j}\}$ with $Y^{i,j} = \{y_{u,z}^{i,j} \mid u < z, v_u v_z \in E(G)\}$. The path-gadget contains women $\{p_i \mid i \in [\binom{k}{2} + 2]\}$ and men $\{q_i \mid i \in [\binom{k}{2} + 2]\}$. The set of all these women and men define $X$ and $Y$, respectively.

Let $M_0$ contain the pairs $x_u^i y_u^i$ and $x_{u,z}^{i,j} y_{u,z}^{i,j}$ for all possible $i,j,u$ and $z$, and also the pairs $p_h q_{h+1}$ for all $h \in [\binom{k}{2} + 1]$. Note that $|M_0| = |X| - 1$, since $p_{\binom{k}{2}+2}$ is the only unmatched woman. Let $\nu$ be a bijection from $[\binom{k}{2}]$ into the set $\binom{[k]}{2}$, and let $C(i,u) = \{x_{u,z}^{i,j} \mid i < j \leq k, u < z, v_u v_z \in E(G)\} \cup \{x_{z,u}^{j,i} \mid 1 \leq j < i, z < u, v_z v_u \in E(G)\}$ for all $i \in [k], u \in [n]$. We define the precedence list $L(a)$ for each person $a$ below. A tie $T = \{t_1, \ldots, t_i\}$ w.r.t. $a$ is denoted by $(t_1, \ldots, t_i)$ in $L(a)$, and we use $[s_1, \ldots, s_i]$ to denote an arbitrary ordering of $s_1, \ldots, s_i$. (If it is not confusing, we will simply write $(S)$ or $[S]$ instead of listing the elements of $S$ in the brackets.) Observe that there are indeed $\binom{k}{2} + k + 1$ indifferent women, there is no indifferent man, and each indifferent woman has exactly one tie in her preference list. The indices $i, j, u$, and $z$ take all possible values in the lists, unless otherwise stated. For brevity, we write $k'$ for $\binom{k}{2}$.

| | |
|---|---|
| $L(x_u^i)\colon y_u^i, y_0^i$ | $L(y_u^i)\colon x_0^i, [C(i,u)], x_u^i$ |
| $L(x_0^i)\colon y_0^i, (Y^i)$ | $L(y_0^i)\colon [X^i], x_0^i$ |
| $L(x_{u,z}^{i,j})\colon y_{u,z}^{i,j}, [y_u^i, y_z^j], y_0^{i,j}$ | $L(y_{u,z}^{i,j})\colon x_0^{i,j}, x_{u,z}^{i,j}$ |
| $L(x_0^{i,j})\colon y_0^{i,j}, (Y^{i,j})$ | $L(y_0^{i,j})\colon [X^{i,j}], p_{\nu^{-1}(i,j)}, x_0^{i,j}$ |
| $L(p_h)\colon q_{h+1}, y_0^{\nu(h)}, q_h$ if $h \in [k']$ | $L(q_h)\colon p_h, p_{h-1}$ if $2 \leq h \leq k'+1$ |
| $L(p_{k'+1})\colon (q_{k'+1}, q_{k'+2})$ | $L(q_1)\colon p_1$ |
| $L(p_{k'+2})\colon q_{k'+2}$ | $L(q_{k'+2})\colon p_{k'+1}, p_{k'+2}$. |

Observe that $M_0$ assigns each woman in $X \setminus \{p_{k'+2}\}$ to a man that she prefers the most, so they cannot be in a blocking pair for $M_0$. As $p_{k'+2} q_{k'+2}$ is also not a blocking pair, $M_0$ is indeed stable. By $|X| = |Y| = O(\binom{k}{2}) + \binom{k}{2} O(m) + k O(n)$, the construction takes polynomial

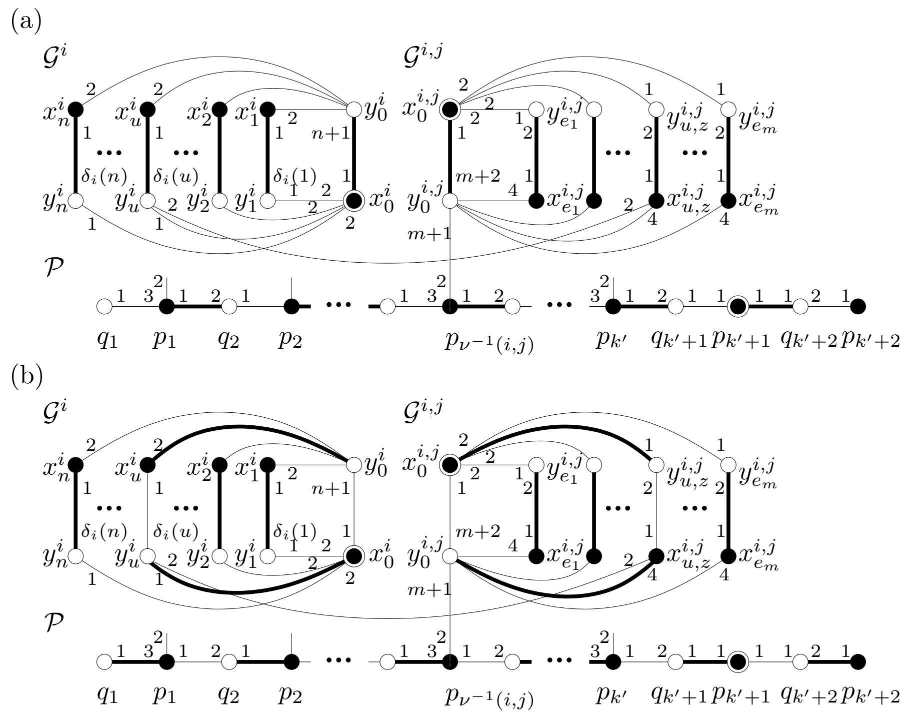


Figure 5.1: Illustration for the SMTI instance $I$ constructed in the proof of Theorem 5.2.1. White circles represent men, black circles represent women, and double black circles represent indifferent women. The bold edges in Figure (a) show $M_0$, and the bold edges in (b) show a possible stable matching $M$ that is larger than $M_0$. The small numbers on the edges represent ranks. We write $\delta_i(u)$ for $|C(i,u)|+2$, and also $e_1$ and $e_m$ for two pairs in $\{(a,b) \mid v_a v_b \in E(G)\}$.

time in $n$ and $m$ (using also $k \leq n$). Since $\kappa_1(I) \leq \binom{k}{2} + k + 1$ also holds, this yields an FPT-reduction.

The basic idea of the above construction is the following. It is easy to see that we can only get a matching $M$ larger than $M_0$ if we "swap" the matching $M_0$ along the path-gadget $\mathcal{P}$. However, the given ranks ensure that this can only result in a stable matching if we make a swap in each edge-gadget as well. (See Fig 5.1 (b). If the matching would include the edge $x_0^{i,j} y_0^{i,j}$, then $y_0^{i,j} p_{\nu^{-1}(i,j)}$ would be a blocking pair.) Such a swapping in the edge-gadget $\mathcal{G}^{i,j}$ can be done in $m$ ways, as we can swap $M_0$ along the cycles formed by $x_0^{i,j}$, $y_0^{i,j}$, $x_{u,z}^{i,j}$, and $y_{u,z}^{i,j}$ for each $u < z$ where $v_u v_z$ is an edge. But the connections between the edge- and node-gadgets ensure that swapping $M_0$ along the cycle in $\mathcal{G}^{i,j}$ corresponding to some edge $v_u v_z$ can only result in a stable matching if we also swap it along the cycles in the node-gadgets $\mathcal{G}^i$ and $\mathcal{G}^j$ corresponding to the vertices $v_u$ and $v_z$, respectively. As we can only make one swap in each gadget (because of the existence of $x_0^i$ and $y_0^i$ in the case of Figure 5.1), this ensures that the $\binom{k}{2}$ edges of $G$ that correspond to the swappings in the edge-gadgets have altogether at most $k$ endpoints, as these endpoints must correspond to the swappings made in the $k$ node-gadgets. Thus, we have a clique in $G$ if and only if we can improve $M_0$.

Before going into the details, we remark that ties are unavoidable in the construction. First, swapping a stable matching along an alternating path of the underlying graph can only result in a stable matching if at least one node of the path corresponds to a person who is indifferent between its two possible partners on the path. Second, if there are two non-disjoint cycles $C_1$ and $C_2$ in the underlying graph such that swapping some stable matching along $C_1$ and along $C_2$ both result in a stable matching, then at least one person corresponding to a node in $C_1$ or $C_2$ must be indifferent. Thus, we need ties both for constructing an instance with a possibly improvable solution, and also for leaving enough space for the possible improvements to map the different cliques of the graph to different solutions.

To detail the proof of the reduction, we first show that the following are equivalent for any matching $M$ for $I$:

- property (p1): $p_1q_1 \in M$ and $M$ is stable,
- property (p2): $|M| = |M_0| + 1$ and $M$ is stable, and
- property (p3): $|M| = |M_0| + 1$, $M$ is stable, and $M$ is $\ell$-close to $M_0$.

Property (p3) $\Longrightarrow$ (p2) is trivial, and (p2) $\Longrightarrow$ (p1) should also be clear. To prove (p1) $\Longrightarrow$ (p3), suppose that $M$ is a stable matching with $M(q_1) = p_1$. First, to prevent $p_1q_2$ from being a blocking pair, $M$ must assign $p_2$ to $q_2$. Applying this argument iteratively, we obtain that $M(q_h) = p_h$ for each $h \in [\binom{k}{2} + 1]$. Also, $q_{\binom{k}{2}+2}p_{\binom{k}{2}+2}$ must be contained in $M$, as otherwise this would be a blocking pair. Since $\rho^L(p_h, y_0^{\nu(h)}) > \rho^L(p_h, q_h)$ for each $h \in [\binom{k}{2}]$, we get that $M$ can only be stable if $y_0^{\nu(h)}$ has a partner in $M$ whom he prefers to $p_h$, implying $M(y_0^{i,j}) \in X^{i,j}$ for each $(i,j) \in \binom{k}{2}$. We denote by $\sigma(i,j)$ the pair $(u,z)$ if $M(y_0^{i,j}) = x_{u,z}^{i,j}$, and similarly we let $\sigma(i) = u$ if $M(y_0^i) = x_u^i$.

As $\rho^L(x_{u,z}^{i,j}, y_{u,z}^{i,j}) = 1$ for every possible $(u,z)$, we get $M(y_{\sigma(i,j)}^{i,j}) = x_0^{i,j}$, since otherwise $x_{\sigma(i,j)}^{i,j}y_{\sigma(i,j)}^{i,j}$ would be a blocking pair. Also, we obtain $M(x_{u,z}^{i,j}) = y_{u,z}^{i,j}$ for all $(u,z) \neq \sigma(i,j)$ for the same reason. Thus, each person in an edge-gadget $\mathcal{G}^{i,j}$ can only be assigned to a person in $\mathcal{G}^{i,j}$.

Suppose $\sigma(i,j) = (u^*, z^*)$. As $x_{\sigma(i,j)}^{i,j}$ prefers $y_{u^*}^i$ to $y_0^{i,j}$, and $y_{u^*}^i$ prefers $x_{\sigma(i,j)}^{i,j} \in C(i,u^*)$ to $x_{u^*}^i$, $M(x_{u^*}^i) = y_{u^*}^i$ is not possible, since then $y_{u^*}^i$ and $x_{\sigma(i,j)}^{i,j}$ would form a blocking pair. As $C(i,u^*)$ is a subset of persons in $\mathcal{G}^{i,j}$, we get $M(y_{u^*}^i) \notin C(i,u^*)$ by the argument above. This implies $M(y_{u^*}^i) = x_0^i$. Using again the stability of $M$, we also obtain $M(y_u^i) = x_u^i$ for every $u \neq u^*$, and $M(x_{u^*}^i) = y_0^i$. Note that this latter means $\sigma(i) = u^*$. Using the same arguments again, we also obtain $M(y_{z^*}^j) = x_0^j$, $M(y_0^j) = x_{z^*}^j$, and $M(y_z^j) = x_z^j$ for each $z \neq z^*$. This yields $\sigma(j) = z^*$, so we have $\sigma(i,j) = (\sigma(i), \sigma(j))$ for each $(i,j) \in \binom{[k]}{2}$.

Observe also that $M$ covers each person of the instance, meaning $|M| = |M_0| + 1$. There are exactly 4 persons $a$ in each node-gadget and in each edge-gadget for which $M(a) \neq M_0(a)$ holds. As $M(a) \neq M_0(a)$ holds for every person $a$ in the path-gadget, we can conclude that $M$ is $(6\binom{k}{2} + 4k + 4) = \ell$-close to $M_0$. Thus, (p1) indeed implies (p3), so the properties (p1), (p2) and (p3) are equivalent.

Now, by $\sigma(i,j) = (\sigma(i), \sigma(j))$, the definition of $X^{i,j}$ implies that $v_{\sigma(i)}v_{\sigma(j)}$ is an edge in $G$ for each $(i,j) \in \binom{[k]}{2}$. Hence, we can conclude that $\{v_{\sigma(i)} \mid i \in [k]\}$ is a clique of size $k$ in $G$, proving (1) $\Longrightarrow$ (2).

Finally, we prove (2) $\Longrightarrow$ (1). Suppose $\{v_{\sigma(i)} \mid i \in [k]\}$ is a clique in $G$. We construct a stable matching $M$ of size $|M_0| + 1$ as follows (the indices take all the possible values):

$$M(x_0^i) = y_{\sigma(i)}^i \qquad M(y_0^i) = x_{\sigma(i)}^i$$
$$M(x_0^{i,j}) = y_{\sigma(i),\sigma(j)}^{i,j} \qquad M(y_0^{i,j}) = x_{\sigma(i),\sigma(j)}^{i,j}$$
$$M(q_h) = p_h.$$

By setting $M(a) = M_0(a)$ for every other person $a$, $|M| = |M_0| + 1$ is clear. It is straightforward to verify that $M$ is stable, proving the theorem. □

Observe that there is no bound on the length of the ties in the SMTI instance constructed in the proof of Theorem 5.2.1. Thus, we could hope that restricting $\kappa_2$ to be small yields an easier problem. But as already mentioned, NP-completeness has been shown [98] for the special case of MaxSMTI when $\kappa_2(I) = 2$ holds for every input $I$. Besides, $\kappa_3(I) \leq \kappa_1(I)\kappa_2(I)$, so Theorem 5.1.1 trivially implies that MaxSMTI is FPT with combined parameterization $(\kappa_1, \kappa_2)$.

This latter fact trivially gives us a permissive local search algorithm for MaxSMTI with FPT running time, assuming the combined parameterization $(\kappa_1, \kappa_2)$. Thus, it is natural to ask whether we can also give an FPT permissive local search algorithm by parameterizing the problem with only $\kappa_2$. The following theorem shows that no such algorithm can be given (supposing the standard assumption W[1] $\neq$ FPT holds). Moreover, the problem remains hard even if we restrict $\kappa_2 = 2$, and regard $\ell$ as a parameter.

**Theorem 5.2.2.** *If W[1] $\neq$ FPT, then there is no permissive local search algorithm for* MaxSMTI *that runs in FPT time with parameter $\ell$, even if $\kappa_2 = 2$ and only women can be indifferent.*

*Proof.* The proof will be very similar to the proof of Theorem 5.2.1, so we will reuse some of the definitions and arguments used there. Clearly, we have to eliminate long ties in the constructed instance. Note that the instance constructed in the proof of Theorem 5.2.1 only contains ties longer than two in the preference lists of $x_0^i$ and $x_0^{i,j}$ (where $i \in [k]$ and $(i,j) \in \binom{[k]}{2}$, respectively). Therefore, we break the ties in these lists. However, we must not narrow the number of possibilities for improving the initial matching. Thus, in order to avoid the presence of blocking pairs for the swapped solutions, we have to place indifferent persons on each of the alternating cycles that might take part in a possible swapping.

Let $G(V,E)$ be the input graph and $k$ be the parameter for the Clique problem. As before, we are going to construct an SMTI instance $I$ with $\kappa_2 = 2$ together with a stable matching $M_0$ for $I$ and the integer $\ell = 12\binom{k}{2} + 8k + 4$ such that the following three statements are equivalent:

(1) $I$ has a stable matching of size at least $|M_0| + 1$,

(2) $I$ has a stable matching $M$ of size at least $|M_0| + 1$ that is $\ell$-close to $M_0$, and

(3) there is a clique of size $k$ in $G$.

Since the construction will take polynomial time, this clearly proves our theorem. Note that (2) $\Longrightarrow$ (1) is trivial.

Figure 5.2 shows an illustration for the construction. Let $V(G) = \{v_i \mid i \in [n]\}$ and $m = |E(G)|$. The instance $I$ consists of *node-gadgets* $\mathcal{G}^i$ for each $i \in [k]$, *edge-gadgets* $\mathcal{G}^{i,j}$ for each $(i,j) \in \binom{[k]}{2}$, and a *path-gadget* $\mathcal{P}$. The node-gadget $\mathcal{G}^i$ consists of women $A^i \cup C^i \cup \{x_0^i, x_1^i\}$ and men $B^i \cup D^i \cup \{y_0^i, y_1^i\}$, where $A^i = \{a_u^i \mid u \in [n]\}$, and $B^i, C^i, D^i$ are defined analogously to $A^i$. The edge-gadget $\mathcal{G}^{i,j}$ consists of women $A^{i,j} \cup C^{i,j} \cup \{x_0^{i,j}, x_1^{i,j}\}$ and men $B^{i,j} \cup D^{i,j} \cup \{y_0^{i,j}, y_1^{i,j}\}$, where $A^{i,j} = \{a_{u,z}^{i,j} \mid u < z, v_u v_z \in E(G)\}$, and $B^{i,j}, C^{i,j}, D^{i,j}$ are defined similarly. The path-gadget $\mathcal{P}$ is defined in the same way as in the proof of Theorem 5.2.1. Note that the number of men and women in $I$ is $O(\binom{k}{2}) + \binom{k}{2}O(m) + kO(n)$, so the construction takes polynomial time in the size of $G$.

For each $i \in [k]$ we let $M_0(a_u^i) = b_u^i$ and $M_0(c_u^i) = d_u^i$ for each $u \in [n]$, and $M_0(x_h^i) = y_h^i$ for $h \in \{0,1\}$. Similarly, for each $(i,j) \in \binom{[k]}{2}$ we let $M_0(a_{u,z}^{i,j}) = b_{u,z}^{i,j}$ and $M_0(c_{u,z}^{i,j}) = d_{u,z}^{i,j}$ for each possible $u$ and $z$, and $M_0(x_h^{i,j}) = y_h^{i,j}$ for $h \in \{0,1\}$. We define the preference lists for $I$ by using the notation of the proof of Theorem 5.2.1. For each person $p$ in $\mathcal{P}$ we let both $M_0(p)$ and its preference list $L(p)$ be defined as in the proof of Theorem 5.2.1. We also

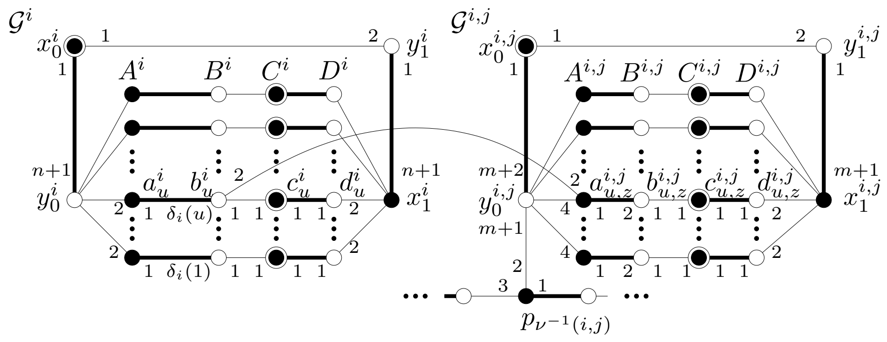


Figure 5.2: The modified gadgets of the SMTI instance $I$ constructed in the proof of Theorem 5.2.2. The bold edges in Figure (a) represent $M_0$, and the bold edges in (b) show a possible stable matching $M$ that is larger than $M_0$.

define $C(i,u) = \{a_{u,z}^{i,j} \mid i < j \leq k, u < z, v_u v_z \in E(G)\} \cup \{a_{z,u}^{j,i} \mid 1 \leq j < i, z < u, v_z v_u \in E(G)\}$ for all $i \in [k], u \in [n]$.

| | |
|---|---|
| $L(a_u^i)$: $b_u^i, y_0^i$ | $L(b_u^i)$: $c_u^i, [C(i,u)], a_u^i$ |
| $L(c_u^i)$: $(b_u^i, d_u^i)$ | $L(d_u^i)$: $c_u^i, x_1^i$ |
| $L(x_0^i)$: $(\{y_0^i, y_1^i\})$ | $L(y_0^i)$: $[A^i], x_0^i$ |
| $L(x_1^i)$: $[D^i], y_1^i$ | $L(y_1^i)$: $x_1^i, x_0^i$ |
| $L(a_{u,z}^{i,j})$: $b_{u,z}^{i,j}, [b_u^i, b_z^j], y_0^{i,j}$ | $L(b_{u,z}^{i,j})$: $c_{u,z}^{i,j}, a_{u,z}^{i,j}$ |
| $L(c_{u,z}^{i,j})$: $(b_{u,z}^{i,j}, d_{u,z}^{i,j})$ | $L(d_{u,z}^{i,j})$: $c_{u,z}^{i,j}, x_1^{i,j}$ |
| $L(x_0^{i,j})$: $(y_0^{i,j}, y_1^{i,j})$ | $L(y_0^{i,j})$: $[A^{i,j}], p_{\nu^{-1}(i,j)}, x_0^{i,j}$ |
| $L(x_1^{i,j})$: $[D^{i,j}], y_1^{i,j}$ | $L(y_1^{i,j})$: $x_1^{i,j}, x_0^{i,j}$. |

It is easy to see that $M_0$ is indeed a stable matching for $M$, and covers every woman except for $p_{\binom{k}{2}+2}$. Observe that $\kappa_2(I) = 2$ indeed holds, but $\kappa_1(I)$ is not bounded.

To prove (1) $\Longrightarrow$ (3), suppose that $M$ is a stable matching that covers every man and woman. Using the same arguments as in the proof of Theorem 5.2.1, we obtain $M(q_h) = p_h$ for each $h \in [\binom{k}{2}+2]$ and $M(y_0^{i,j}) \in A^{i,j}$ for each $(i,j) \in \binom{[k]}{2}$. Following that argument and exploiting the stability of $M$, after defining $\sigma(i,j)$ to be $(u,z)$ if $M(y_0^{i,j}) = a_{u,z}^{i,j}$ and $\sigma(i)$ to be $u$ if $M(y_0^i) = a_u^i$, we can easily obtain $\sigma(i,j) = (\sigma(i), \sigma(j))$ proving that $\{v_{\sigma(i)} \mid i \in [k]\}$ is a clique in $G$. This proves (1) $\Longrightarrow$ (3).

To prove (3) $\Longrightarrow$ (2), let $\{v_{\sigma(i)} \mid i \in [k]\}$ be a clique in $G$. We define a stable matching $M$ covering each person in $I$ as follows.

| | |
|---|---|
| $M(y_0^i) = a_{\sigma(i)}^i$ | $M(y_0^{i,j}) = a_{\sigma(i),\sigma(j)}^{i,j}$ |
| $M(b_{\sigma(i)}^i) = c_{\sigma(i)}^i$ | $M(b_{\sigma(i),\sigma(j)}^{i,j}) = c_{\sigma(i),\sigma(j)}^{i,j}$ |
| $M(d_{\sigma(i)}^i) = x_1^i$ | $M(d_{\sigma(i),\sigma(j)}^{i,j}) = x_1^{i,j}$ |
| $M(y_1^i) = x_0^i$ | $M(y_1^{i,j}) = x_0^{i,j}$ |
| $M(q_h) = p_h$. | |

For every other person $p$ in $I$ we let $M(p) = M_0(p)$. It is straightforward to check that $M$ is a stable matching for $I$ that is $\ell$-close to $M_0$. □

## 5.3 Inapproximability results

Theorem 5.1.2 shows that a stable matching for an instance $I$ of maximum size is hard to find even if $\kappa_1(I)$ is small. By [98], we know that the case where $\kappa_2(I) = 2$ is NP-hard.

Theorems 5.2.1 and 5.2.2 prove that improving an initial stable matching also remains hard in these cases, even if we can restrict our attention to solutions that are close to the initial solution. However, we can still try to find stable matchings that may not be of maximum size, but have some other useful properties.

If $M$ is a stable matching for an SMTI instance $I = (X, Y, L)$, then the *cost for $p$ w.r.t. $M$*, denoted by $c_M(p)$, is defined to be $\rho^L(p, M(p))$ if $M(p) \neq \oslash$, and $1 + \rho^L(p, q^*)$ otherwise, where $q^*$ has maximum rank according to $p$ among all acceptable partners for $p$. Now, the *weight* of $M$ is $w(M) = \sum_{p \in X \cup Y} c_M(p)$, and the *regret* of $M$ is $r(M) = \max_{p \in X \cup Y} c_M(p)$. An *egalitarian* (*minimum regret*) stable matching for $I$ is a stable matching for $I$ that has the minimum weight (regret, respectively) among all stable matchings for $I$. The task of the Egalitarian Stable Marriage with Ties and Incomplete Lists (or EgalSMTI) problem is to find an egalitarian stable matching for the given SMTI instance, and the Minimum Regret Stable Marriage with Ties and Incomplete Lists (or MinregSMTI) problem is defined analogously.

If P $\neq$ NP and $\varepsilon > 0$ then there is no polynomial-time approximation algorithm with ratio $N(I)^{1-\varepsilon}$ for these problems, even if only women can be indifferent, each preference list has at most one tie, and $\kappa_2(I) = 2$ [98]. Here, $N(I)$ is the number of men in $I$. Moreover, it has also been shown by Halldórsson [75] that for some $\delta > 0$ it is NP-hard to approximate EgalSMTI and MinregSMTI within a ratio of $\delta N(I)$. However, if there are no ties, then a minimum regret or an egalitarian stable matching can be found in polynomial time [81, 71]. As Theorem 5.3.1 shows, this can be exploited to give an FPT algorithm for both EgalSMTI and MinregSMTI if we parameterize it by $\kappa_3$, or equivalently, by $(\kappa_1, \kappa_2)$. By contrast, Theorems 5.3.2 and 5.3.3 present some bounds on the approximability of these problems that hold even if we allow the approximation algorithm to run not in polynomial time but in FPT time with parameterization $\kappa_1$. Note that such an approximation algorithm would have a tractable running time if $\kappa_1$ is a small integer, even if the length of the ties is unbounded.

**Theorem 5.3.1.** EgalSMTI *and* MinregSMTI *are FPT with parameterization $\kappa_3$.*

*Proof.* We will use the method described in the proof of Theorem 5.1.1 for breaking ties, and we also adopt the notation $T_i, P_i$ and $\pi_i^j$ for some $i \in [\kappa_1(I)]$ and $j \in [|T_i|]$. Denoting the elements of $P_1 \times \cdots \times P_{\kappa_1(I)}$ by $p_1, \ldots, p_t$, we write $I_i$ for the instance obtained by breaking ties according to $p_i$. Note that the instances $I_1, \ldots, I_t$ differ from $I$ only by the ranks assigned to the persons that are contained in a tie.

In order to find a minimum regret or an egalitarian matching, we have to define the ranking functions of the instances $I_1, \ldots, I_t$ in a special way, using the ranking function $\rho$ of $I$. More precisely, if $M_i$ is an egalitarian (minimum regret) stable matching for $I_i$ ($i \in [t]$), then we need the following to be true: the weight of an egalitarian (minimum regret) stable matching for $I$ equals $\min_{i \in [t]}\{w(M_i)\}$ ($\min_{i \in [t]}\{r(M_i)\}$, respectively). To enforce this property, which we will call *weight conserving property*, we define the ranking function $\rho^i$ of $I_i$ for some $p_i = (\pi_1, \ldots, \pi_{\kappa_1(I)})$ as follows (allowing $b = \oslash$ also):

$$\rho^i(a, b) = \begin{cases} \rho(a, b) + \frac{\pi_k(b) - 1}{|T_k|} & \text{if } T_k \text{ is a tie w.r.t. } a \text{ that contains } b, \\ \rho(a, b) & \text{if no such tie exists.} \end{cases}$$

First, suppose that $M$ is an egalitarian matching for $I$. Let $\pi_i$ ($i \in [\kappa_1(I)]$) be such that $\pi_i(M(a)) = 1$ if $T_i$ is a tie w.r.t. $a$ containing $M(a)$, otherwise $\pi_i$ is chosen arbitrarily. Observe that $M$ is a stable matching of $I_{(\pi_1, \ldots, \pi_t)}$, and the cost for any person w.r.t. $M$ is the same in $I$ and in $I_{(\pi_1, \ldots, \pi_t)}$, by the definition of $\rho^i$. Second, if $M_i$ is a stable matching in $I_i$, then it is also stable in $I$, and the cost for any person w.r.t. $M_i$ in $I$ is at most its cost in $I_i$. Thus, we get that this tie breaking method indeed fulfills the weight conserving property.

Note that the instances obtained have a strict ranking for each person, but the ranks may be not only be integers but rational numbers as well. However, finding an egalitarian or a minimum regret stable matching for such instances can be done in $O(n^4 \log n)$ time by [81, 71] for an instance of size $n$, so this does not cause any problems. Hence, breaking ties as above, finding the egalitarian or minimum regret stable matching for the instances obtained, and choosing a stable matching among them having minimum weight or regret yields an algorithm for both problems with running time $\prod_{i \in [\kappa_1(I)]} |T_i| O(|I|^4 \log |I|) = O(\kappa_3(I)^{\kappa_3(I)} |I|^4 \log |I|)$. $\square$

We remark that if $\kappa_1(I)$ is a fixed constant, then both the egalitarian and the minimum regret matching can be found in polynomial time, as the algorithm of Theorem 5.3.1 runs in $\prod_{i \in [\kappa_1(I)]} |T_i| O(|I|^4 \log |I|) \leq |I|^{\kappa_1(I)} O(|I|^4 \log |I|) = O(|I|^{\kappa_1(I)+4} \log |I|)$. Theorems 5.3.2 and 5.3.3 show that if $\kappa_1$ is not a constant but a parameter, then we have strong lower bounds on the ratio of any FPT approximation algorithm, for both of these problems.

**Theorem 5.3.2.** *There is a $\delta > 0$ such that if W[1] $\neq$ FPT, then there is no FPT-approximation with parameterization $\kappa_1$ for* EgalSMTI *that has ratio $\delta N(I)$, even if only women can be indifferent.*

*Proof.* We show that the theorem holds for $\delta = \frac{1}{14}$. Suppose that an FPT-approximation algorithm $\mathcal{A}$ with parameterization $\kappa_1$ and ratio $\delta N(I)$ exists for EgalSMTI. We will show that this yields an FPT algorithm for the Clique problem.

We are going to construct an SMTI instance $I'$ by adding some new persons to the instance $I$ constructed in the proof of Theorem 5.2.1. See Figure 5.3 for an illustration. The basic idea is that we complement the path-gadget $\mathcal{P}$ in a way such that the weight of the solution is mainly determined by the choices made for the persons in the path-gadget. Thus, we can only decrease the weight of the initial solution by swapping it along the path-gadget, which can be done exactly if there is a clique in the given graph (as we have already seen). Moreover, we ensure that swapping the initial solution along the path-gadget results in a decrease of the weight that is large enough, so that even an FPT-approximation algorithm can detect whether this is possible.

Clearly, these ideas are also useful for considering the MinregSMTI problem. Therefore, we describe the construction in a general way, by using some parameters in the classical sense ($J$ and $K$) that can be used to "tune" the weights that appear in the weight or in the regret of the solutions.

Now, we describe the details of the proof. Let $G(V, E)$ and $k$ be the input graph and the parameter for Clique, with $V(G) = \{v_i \mid i \in [n]\}$ and $|E(G)| = m$. Let $X$ and $Y$ denote the set of women and men in the instance $I$. To construct $I'$ from $I$, we add new women $A \cup S$ and men $B \cup T$, where $A = \{a_u \mid u \in \{0\} \cup [J]\}$, $B = \{b_u \mid u \in [J]\}$, $S = \{s_u \mid u \in [K]\}$, and $T = \{t_u \mid u \in [K]\}$. The value of the integers $J$ and $K$ will be specified later. The preference lists for the newly introduced person and for $q_1$ are given below, and we define the preference list $L(p)$ for each remaining person $p$ as in $I$.

$L(a_u)\colon b_u, b_{u+1}$ if $u \in [J-1]$
$L(a_0)\colon q_1, b_1$
$L(a_L)\colon b_J$
$L(b_u)\colon a_{u-1}, [S], a_u$.

$L(q_1)\colon p_1, [S], a_0$
$L(s_u)\colon t_u, [\{q_1\} \cup B]$
$L(t_u)\colon s_u$

Note that $N(I') = K + J + |Y|$, where $|Y| = \binom{k}{2} + 2 + k(n+1) + \binom{k}{2}(m+1)$. Let $M_{\mathcal{A}}$ be the output of $\mathcal{A}$ on input $I'$, and let $M_E$ be an egalitarian stable matching for $I'$. First, suppose there is a clique of size $k$ in $G$. Let $M$ be a stable matching containing $p_1 q_1$ for the instance $I$, such an $M$ can be defined as in the last paragraph of the proof of Theorem 5.2.1. By adding the pairs $\{s_u t_u \mid u \in [K]\}$ and $\{b_u a_{u-1} \mid u \in [J]\}$ to $M$, we clearly obtain a stable matching $M'$ for $I'$.

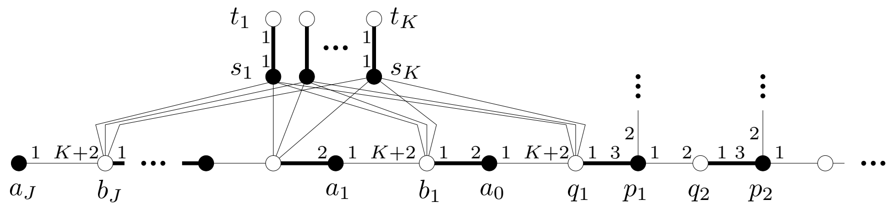


Figure 5.3: The modified SMTI instance $I'$ constructed in the proof of Theorem 5.3.2. Bold edges represent $M'$.

Observe that the regret of $M$ in $I$ is at most the maximum rank $R$ in $I$, which is at most $\max\{m+2, n+1, (k-1)\Delta(G)+2\}$, with $\Delta(G)$ denoting the highest degree in $G$. Thus we get $w(M) \leq (|X|+|Y|)R = 2|Y|R$, implying $w(M') = 2+(2+1)J+(1+1)K+w(M) \leq 3J+2K+2R|Y|+2$. Since $w(M_E) \leq w(M')$ and the ratio of $\mathcal{A}$ is $\delta N(I')$, we obtain that $w(M_\mathcal{A}) \leq \delta N(I')w(M_E) \leq \delta(K+J+|Y|)(3J+2K+2R|Y|+2) =: w^1_\mathcal{A}$.

Now, if there is no clique of size $k$ in $G$, then no stable matching for $I'$ can contain $p_1q_1$. To see this, observe that the restriction of such a matching on $I$ would also be stable. Now, recall that if $p_1q_1$ is contained in some stable matching for $I$, then $G$ contains a clique of size $k$, by the claims in the proof of Theorem 5.2.1. Thus, $q_1p_1 \notin M_\mathcal{A}$. Observe also that every stable matching must include the pairs $\{s_ut_u \mid u \in [K]\}$, as $s_u$ and $t_u$ prefer each other the most. Using this, we get that $q_1a_0$ must be in $M'$, as otherwise it would be a blocking pair. Applying this argument repeatedly, we can easily see that $a_ib_i$ is in $M'$ for $i = 1, 2, \ldots, J$. These observations together imply $w(M_\mathcal{A}) \geq (K+2+1)(J+1)+2K = KJ+3J+3K+3 =: w^2_\mathcal{A}$.

Now, if $w^1_\mathcal{A} < w^2_\mathcal{A}$ holds, then algorithm $\mathcal{A}$ can decide whether there is a clique of size $k$ in $G$, by outputting 'Yes' if $w(M_\mathcal{A}) \leq w^1_\mathcal{A}$ and outputting 'No' if $w(M_\mathcal{A}) \geq w^2_\mathcal{A}$. Setting $K = 2J$ and $J = 2R|Y|+2$ guarantees $w^1_\mathcal{A} < w^2_\mathcal{A}$, because $w^1_\mathcal{A} < \delta \cdot 3.5J \cdot 8J = 28\delta J^2 = 2J^2 < w^2_\mathcal{A}$. Finally, observe that $R = O(m+nk)$ and $|Y| = O(k^2m+nk)$ implies $N(I') = K+J+|Y| = 3(2R|Y|+2)+|Y| = O(k^2(m^2+n^2)+k^3nm)$, hence the instance $I'$ can be created in polynomial time. So by $\kappa_1(I') = \binom{k}{2}+k+1$, the presented algorithm for the Clique problem indeed runs in FPT time.

We remark that the theorem also holds for any $\delta < \frac{1}{5+2\sqrt{6}}$. This can be proven by an easy calculation similar to the one above, by setting $K = \lceil\sqrt{\alpha^2+5\alpha+3}\rceil J$ and $J = \lceil(R|Y|+1)/\alpha\rceil$ for some $\alpha$ that is close enough to 0. □

**Theorem 5.3.3.** *If W[1] $\neq$ FPT and $\varepsilon > 0$, then there is no FPT-approximation with parameterization $\kappa_1$ for* MinregSMTI *that has ratio $N(I)^{1-\varepsilon}$.*

*Proof.* Again, as in the proof of Theorem 5.3.2, we suppose that there exists such an FPT-approximation algorithm $\mathcal{A}$, and we show that this can be used to give an FPT algorithm for the Clique problem. Let $G$ and $k$ be the input graph and the parameter given for the Clique problem. We will use the construction $I'$, introduced in the proof of the Theorem 5.3.2, with the restriction that we set $J = 0$ and $K = \max\{|Y|, \lceil(2R)^{1/\varepsilon}\rceil\}$. Recall that $N(I') = K+J+|Y|$, $R = O(m+nk)$ and $Y = O(k^2m+nk)$. Since $\varepsilon$ is a constant, $I'$ can again be constructed from $G$ and $k$ in polynomial time.

Let $M_\mathcal{A}$ denote the output of $\mathcal{A}$ on input $I'$. Suppose that $G$ has a clique of size $k$. If we define $M'$ as in the proof of Theorem 5.3.2, then $p_1q_1 \in M'$, and $r(M') \leq R$ is easy to see. By the ratio of $\mathcal{A}$, and using also $K \geq |Y|$ we get $r(M_\mathcal{A}) \leq N(I')^{1-\varepsilon}R = (K+|Y|)^{1-\varepsilon}R \leq 2K^{1-\varepsilon}R$.

For the other direction, if there is no clique of size $k$ in $G$, then $p_1q_1$ cannot be contained $M_\mathcal{A}$, as we have already shown in the proof of Theorem 5.2.1 that this would imply

| | Parameters | | |
|---|---|---|---|
| | $\kappa_2 = 2$ (and $\ell$) | $\kappa_1$ (and $\ell$) | $\kappa_1, \kappa_2$ |
| MaxSMTI | NP-hard ([98]) | W[1]-hard (Theorem 5.1.2) | FPT (Theorem 5.1.1) |
| Local Search for MaxSMTI | No FPT alg. (Theorem 5.2.2) | No FPT alg. (Theorem 5.2.1) | FPT (Theorem 5.1.1) |
| Approximation for EgalSMTI | No poly. alg. has ratio $N^{1-\varepsilon}$ if $\varepsilon > 0$ ([98]) | No FPT alg. has ratio $\delta N$ for some $\delta > 0$ (Theorem 5.3.2) | FPT, exact (Theorem 5.3.1) |
| Approximation for MinregSMTI | No poly. alg. has ratio $N^{1-\varepsilon}$ if $\varepsilon > 0$ ([98]) | No FPT alg. has ratio $N^{1-\varepsilon}$ if $\varepsilon > 0$ (Theorem 5.3.3) | FPT, exact (Theorem 5.3.1) |

Table 5.1: Summary of the results of Chapter 5 (assuming W[1] $\neq$ FPT and P $\neq$ NP). The parameter $\ell$ is only defined in the local search problem for MaxSMTI, and $N$ denotes the number of men.

a clique of size $k$ in $G$. Since $s_u$ must be assigned to $t_u$ in any stable matching of $I'$, we get $q_1 a_0 \in M_{\mathcal{A}}$, implying $c_{M_{\mathcal{A}}}(q_1) = K + 2$. Thus, $r(M_{\mathcal{A}}) \geq K + 2$ in this case.

It is easy to check that $K + 2 > K \geq K^{1-\varepsilon} 2R$ holds by the choice of $K$. Therefore, $\mathcal{A}$ indeed can decide whether $G$ has a clique of size $k$, and by $\kappa_1(I') = \binom{k}{2} + k + 1$, this yields an FPT algorithm for Clique. □

## 5.4 Summary

We have shown that MaxSMTI remains W[1]-hard if we parameterize the problem with the number of ties, but becomes FPT if parameterized with the total length of ties. We have also shown that if W[1] $\neq$ FPT, then no local search algorithm with FPT running time can be given for this problem when the radius of the neighborhood to be examined is considered as parameter, even if each tie has length 2, or even if the number of ties is also considered as a parameter.

We also proved that no FPT algorithm can approximate the EgalSMTI or the MinregSMTI problem, when the parameter is the number of ties, unless W[1] = FPT. In contrast with this, both problems can be solved in FPT time if we parameterize them by the total length of ties.

A summary of the results of Chapter 5 is shown in Table 5.1.

# CHAPTER 6

# Stable matching with couples

In this chapter, we investigate the Hospitals/Residents with Couples problem. After discussing its parameterized complexity with the parameter being the number of couples, we also investigate the parameterized complexity of the neighborhood search problem for the maximization variant of HRC, considering different parameterization functions. We also present results concerning the simplified variant of this problem where preferences are ignored, and only an assignment of maximum size is sought for.

We will call an instance of the HRC problem a *couples' market with preference*, or shortly *cmp*. A cmp consists of a set $S$ of singles, a set $C$ of couples, a set $H$ of hospitals together with a *capacity* $f(h)$ for each hospital $h$, and a *preference list* $L(a)$ for each $a \in S \cup C \cup H$. Each couple $c$ is a pair $(c(1), c(2))$, and we call the elements of the set $R = \bigcup_{c \in C}\{c(1), c(2)\} \cup S$ *residents*. The set $A = S \cup C \cup H$ is called the set of *agents*.

The preference lists can be incomplete, but cannot involve ties, so if $s \in S$ then $L(s)$ is a strictly ordered set of hospitals, if $c \in C$ then $L(c)$ is a strictly ordered subset of $\widetilde{H}$, and if $h \in H$ then $L(h)$ is a strictly ordered set of residents. Here $\widetilde{H} = (H \cup \{\oslash\}) \times (H \cup \{\oslash\}) \setminus \{(\oslash, \oslash)\}$ where $\oslash$ is a special symbol indicating that someone is unemployed. If $f \equiv f_0$ for some $f_0 \in \mathbb{N}$, then we say that the cmp is $f_0$-*uniform*.

The set of elements appearing in the preference list $L(a)$ is $A^L(a)$, and we say that $x$ is *acceptable* for $a$ if $x \in A^L(a)$. Clearly, we may assume that acceptance is mutual, so $h \in A^L(s)$ holds if and only if $s \in A^L(h)$ for each $s \in S$ and $h \in H$, and $(h_1, h_2) \in A^L(c)$ implies $c(i) \in A^L(h_i)$ or $h_i = u$ for both $i \in \{1, 2\}$, for each $c \in C$. For some person $x$, the *ranking function* $\rho^L$ determines the *rank* of $x$ w.r.t. $a$, denoted by $\rho^L(a, x)$. If $x \in A^L(a)$ then $\rho^L(a, x)$ is $r \in \mathbb{N}$ if $x$ is the $r$-th element in $L(a)$. If $x \notin A^L(a)$, then we let $\rho^L(a, x) = \infty$ for all meaningful $x$. Sometimes we may leave the superscript $L$, if its clear from the context.

An *assignment* for a cmp $(S, C, H, f, L)$ is a function $M : R \to H \cup \{\oslash\}$ such that $M(s) \in A^L(s) \cup \{\oslash\}$ for each $s \in S$, $M(c) \in A^L(c) \cup \{(\oslash, \oslash)\}$ for each $c \in C$, and the number of residents assigned to a hospital $h$ is at most its capacity $f(h)$. Here, $M(c)$ denotes the pair $(M(c(1)), M(c(2)))$, and the set of residents assigned to $h$ in $M$ is the set $\{r | r \in R, M(r) = h\}$, denoted by $M(h)$. We say that an assignment $M$ *covers* a resident $r$ if $M(r) \neq \oslash$, and $M$ covers a couple $c$, if it covers $c(1)$ or $c(2)$. We define the *size* of $M$, denoted by $|M|$, to be the number of residents covered by $M$.

We say that $x$ is *beneficial* for the agent $a$ with respect to an assignment $M$ if $x \in A^L(a)$

and one of the following cases holds: (1) $a \in S \cup C$ and either $a$ is not covered by $M$ or $\rho(a,x) < \rho(a, M(a))$, (2) $a \in H$ and either $|M(a)| < f(a)$ or there exists a resident $r' \in M(h)$ such that $\rho(a,x) < \rho(a,r')$. A *blocking pair* for $M$ can be of three types:

- it is either a pair formed by a single $s$ and a hospital $h$ such that both $s$ and $h$ are beneficial for each other w.r.t. $M$,
- or a pair formed by a couple $c$ and a pair $(h_1,h_2)$ with $h_1 \neq h_2$ such that $(h_1,h_2)$ is beneficial for $c$ w.r.t. $M$, and for both $i \in \{1,2\}$ it holds that if $h_i \neq \oslash$ then either $c(i)$ is beneficial for $h_i$ w.r.t. $M$ or $c(i) \in M(h_i)$,
- or a pair formed by a couple $c$ and a hospital $h$ such that $(h,h)$ is beneficial for $c$ w.r.t. $M$, and *the couple $c$ is beneficial for $h$*. If $h$ prefers $c(1)$ to $c(2)$, this latter means that either $|M(h)| \leq f(h) - 2$, or $|M(h)| \leq f(h) - 1$ and $\rho(h,c(1)) < \rho(h,r)$ for some $r \in M(h)$, or $\rho(h,c(1)) < \rho(h,r_1)$ and $\rho(h,c(2)) < \rho(h,r_2)$ for some $r_1 \neq r_2$ in $M(h)$. [1]

An assignment $M$ for $I$ is *stable* if there is no blocking pair for $M$.

The input of the Hospitals/Residents with Couples problem is a cmp $I$, and the task is to determine a stable assignment for $I$, if such an assignment exists. If no couples are involved, then a stable assignment can always be found in linear time with the Gale-Shapley algorithm [60]. In the case when couples are present, a stable assignment may not exist, as first proved by Roth [124]. Here we also give a simple example.

Let $H = \{h_1,h_2,h_3\}$, $S = \emptyset$, $C = \{(a,b),(c,d)\}$ and $f \equiv 1$. The preference lists are defined below. It is straightforward to verify that no stable assignment exists for this cmp which will be denoted by $I_0$. For example, $M(a) = h_1$, $M(b) = h_2$ and $M(c) = M(d) = \oslash$ is not stable, because $(c,d)$ and $(h_1,h_3)$ form a blocking pair.

| | |
|---|---|
| $L((a,b))\colon (h_1,h_2),(h_2,h_3),(h_3,h_1)$ | $L(h_1) = L(h_2) = L(h_3)\colon c,a,b,d$ |
| $L((c,d))\colon (h_1,h_3),(h_2,h_1),(h_3,h_2)$ | |

Ronn proved that it is NP-complete to decide whether a stable assignment exists for a given cmp [123]. As the following example shows, an instance of the Hospitals/Residents with Couples problem may admit stable assignments of different sizes. The example contains a single $s$, a couple $c = (c_1,c_2)$ and hospitals $h_1$ and $h_2$ with capacities $f(h_1) = 2$ and $f(h_2) = 1$. The preference lists are the following:

| | |
|---|---|
| $L(s)\colon h_2,h_1$ | $L(h_1)\colon s,c_1,c_2$ |
| $L(c)\colon (h_1,h_1),(\oslash,h_2)$ | $L(h_2)\colon c_2,s$ |

In this instance, assigning $s$ to $h_1$ and $c$ to $(\oslash,h_2)$ yields a stable assignment of size 2, whilst assigning $s$ to $h_2$ and $c$ to $(h_1,h_1)$ results in a stable assignment of size 3. We denote by Maximum Hospitals/Residents with Couples (or MaxHRC) the optimization problem where the task is to determine a stable assignment of maximum size for a given cmp. Note that this problem is trivially NP-hard, as it contains the Hospitals/Residents with Couples problem.

When considering the HRC problem, the number of couples in an instance can be a natural parameter. We deal with the parameterized complexity of this problem in Section 6.1, where we prove the negative result of Theorem 6.1.1 stating that the decision version of HRC is W[1]-hard.

In Section 6.3.2, we present results concerning the applicability of local search for the Maximum Hospitals/Residents with Couples problem. In Theorem 6.2.2, we give a strict local search algorithm for MaxHRC with FPT running time, where the parameters are the number of couples and the radius $\ell$ of the neighborhood search. In contrast with this,

[1] We thank David Manlove for pointing out this case.

| Task: | Existence problem | Maximum problem | Local search algorithm with FPT running time | |
|---|---|---|---|---|
| Parameter: | $\|C\|$ | | $\ell$ | $(\|C\|, \ell)$ |
| Without preferences | P (trivial) | randomized FPT (Theorem 6.3.1) | No permissive alg. (Theorem 6.3.5) | Permissive alg. (Theorem 6.3.1) |
| With preferences | W[1]-hard (Theorem 6.1.1) | W[1]-hard (Theorem 6.1.1) | No permissive alg. (Theorem 6.2.1) | Strict alg. (Theorem 6.2.2) |

Table 6.1: Summary of the results of Chapter 6 (assuming W[1] $\neq$ FPT).

if we only regard $\ell$ as a parameter, then Theorem 6.2.1 shows that no permissive local search algorithm can have FPT running time, unless W[1] = FPT.

Most of the questions examined in his chapter are also worth studying in a model that does not involve preferences. This simplification leads to a matching problem that we call MAXIMUM MATCHING WITH COUPLES. We study this problem in Section 6.3. On the one hand, letting $|C|$ be the parameter, a randomized FPT algorithm is presented in Theorem 6.3.1 that finds a matching of maximum size, so this problem becomes easier without preferences. On the other hand, the local search problem still remains hard to solve (Theorem 6.3.5). For a summary of our results see Table 6.1.

The results of this chapter appear in [107].

## 6.1 Parameterized complexity

The main result of this section is Theorem 6.1.1, showing that the HOSPITALS/RESIDENTS WITH COUPLES problem is W[1]-hard with parameter $|C|$, denoting the number of couples in the given instance. As a consequence, the optimization problem MAXIMUM HOSPITALS/RESIDENTS WITH COUPLES is also W[1]-hard with parameter $|C|$.

However, supposing that a stable assignment has already been determined by some method, it is a valid question whether we can increase its size. We will denote this problem INCREASE HOSPITALS/RESIDENTS WITH COUPLES. Formally, its input is a cmp $I$ and a stable assignment $M_0$ for $I$, and the task is to find a stable assignment with size at least $|M_0|+1$. If no couples are involved, then all stable assignments for the instance have the same size, so this problem is trivially polynomial-time solvable. Theorem 6.1.1 shows that INCREASE HOSPITALS/RESIDENTS WITH COUPLES is also W[1]-hard with parameter $|C|$.

**Theorem 6.1.1.** *(1) The decision version of* HOSPITALS/RESIDENTS WITH COUPLES *is W[1]-hard with parameter $|C|$, even in the 1-uniform case.*
*(2) The decision version of* INCREASE HOSPITALS/RESIDENTS WITH COUPLES *is W[1]-hard with parameter $|C|$, even in the 1-uniform case.*

Before proving Theorem 6.1.1, we introduce a special construction that will be very useful in the proof. For a graph $G$ and an integer $k$, we construct a cmp $I^{G,k} = (S, C, H, f, L)$ as follows. See Figure 6.1 for an illustration.

Let $V(G) = \{v_i \mid i \in [n]\}$, $|E(G)| = m$ and let $\nu$ be a bijection from $[m]$ into the set $\{(x, y) \mid v_x v_y \in E(G), x < y\}$. First, we construct a *node-gadget* $\mathcal{G}^i$ for each $i \in [k]$ and an *edge-gadget* $\mathcal{G}^{i,j}$ for each pair $(i, j) \in \binom{[k]}{2}$. The node-gadget $\mathcal{G}^i$ contains hospitals $H^i \cup G^i \cup \{f^i\}$, singles $S^i \cup T^i$ and a couple $a^i$. Analogously, the edge-gadget $\mathcal{G}^{i,j}$ contains hospitals $H^{i,j} \cup G^{i,j} \cup \{f^{i,j}\}$, singles $S^{i,j} \cup T^{i,j}$ and a couple $a^{i,j}$. Here $T^i = \{t^i_j \mid j \in [n-1]\}$

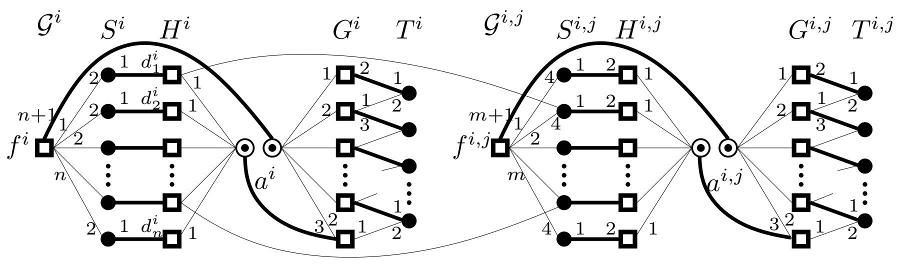


Figure 6.1: A node-gadget and an edge-gadget of $I^{G,k}$. Hospitals are represented by rectangles, singles by black circles, and members of couples by double circles. A hospital $h$ is connected to some resident $r$ if $r \in A^L(h)$. The numbers on the edges represent ranks, bold edges represent $M_0^{G,k}$ from Lemma 6.1.2, and $d_x^i$ is for $|Q_x^i| + 2$.

and $T^{i,j} = \{t_e^{i,j} \mid e \in [m-1]\}$, $H^i = \{h_j^i \mid j \in [n]\}$ and $H^{i,j} = \{h_e^{i,j} \mid e \in [m]\}$, and we define $G^i, S^i$ and $G^{i,j}, S^{i,j}$ similarly to $H^i$ and $H^{i,j}$. Observe that $|C| = k + \binom{k}{2}$.

We let $f \equiv 1$, so $I^{G,k}$ is 1-uniform. The precedence lists for each agent in $I^{G,k}$ are defined below. The notation $[X]$ for some set $X$ in a preference list denotes an arbitrary ordering of the elements of $X$. We write $Q_x^i$ for the set $\{s_e^{i,j} \mid i < j \leq k, \exists y : \nu(e) = (x,y)\} \cup \{s_e^{j,i} \mid 1 \leq j < i, \exists y : \nu(e) = (y,x)\}$ and $Q_e^{i,j}$ for $\{h_x^i, h_y^j\}$ where $\nu(e) = (x,y)$. The indices in the precedence lists take all possible values if not stated otherwise, and the symbol $\alpha$ can be any index in $[k]$ or a pair of indices in $\binom{[k]}{2}$. If $\alpha$ takes a value in $[k]$ then $N(\alpha) = n$, otherwise $N(\alpha) = m$. (This notation will be used again later on.)

| | |
|---|---|
| $L(g_x^\alpha)$: $t_{x-1}^\alpha, a^\alpha(2), t_x^\alpha$ if $1 < x < N(\alpha)$ | $L(h_x^i)$: $a^i(1), [Q_x^i], s_x^i$ |
| $L(g_1^\alpha)$: $a^\alpha(2), t_1^\alpha$ | $L(h_e^{i,j})$: $a^{i,j}(1), s_e^{i,j}$ |
| $L(g_{N(\alpha)}^\alpha)$: $t_{N(\alpha)-1}^\alpha, a^\alpha(2), a^\alpha(1)$ | $L(s_x^i)$: $h_x^i, f^i$ |
| $L(t_x^\alpha)$: $g_x^\alpha, g_{x+1}^\alpha$ | $L(s_e^{i,j})$: $h_e^{i,j}, [Q_e^{i,j}], f^{i,j}$ |
| $L(f^\alpha)$: $s_1^\alpha, s_2^\alpha, \ldots, s_{N(\alpha)}^\alpha, a^\alpha(2)$ | |
| $L(a^\alpha)$: $(g_{N(\alpha)}^\alpha, f^\alpha), (h_1^\alpha, g_{N(\alpha)}^\alpha), (h_2^\alpha, g_{N(\alpha)-1}^\alpha), \ldots, (h_{N(\alpha)}^\alpha, g_1^\alpha)$ | |

**Lemma 6.1.2.** *For a graph $G$ and $k \in \mathbb{N}$, $I^{G,k}$ has a stable assignment $M_0^{G,k}$ that covers each resident. Moreover, statements (1), (2) and (3) are equivalent:*

*(1) There is a clique in $G$ of size $k$.*

*(2) There is a stable assignment $M$ for $I^{G,k}$ with the following property, which we will call property $\pi$: $M(f^{i,j}) \subseteq S^{i,j}$ for each $(i,j) \in \binom{[k]}{2}$.*

*(3) There is a stable assignment for $I^{G,k}$ with property $\pi$ covering each resident.*

*Proof.* To see the first claim, we define an assignment $M_0$ by letting $M_0(a^\alpha) = (g_{N(\alpha)}^\alpha, f^\alpha)$, $M_0(t_x^\alpha) = g_x^\alpha$, and $M_0(s_x^\alpha) = h_x^\alpha$ for all possible values of $\alpha$ and $x$. As each single and couple is assigned to his or their best choice, $M_0$ is stable and covers each resident.

To prove (2) $\Rightarrow$ (1), suppose that $I^{G,k}$ has a stable assignment $M$ with property $\pi$. Let us define $\sigma(i,j)$ for each $(i,j) \in \binom{[k]}{2}$ such that $M(f^{i,j}) = \{s_{\sigma(i,j)}^{i,j}\}$. Since $s_{\sigma(i,j)}^{i,j}$ prefers $h_{\sigma(i,j)}^{i,j}$ to $f^{i,j}$, the stability of $M$ implies $M(h_{\sigma(i,j)}^{i,j}) = \{a^{i,j}(1)\}$. From this, we get that $M(s_e^{i,j}) = h_e^{i,j}$ must hold for each $e \in [m] \setminus \{\sigma(i,j)\}$ since otherwise $s_e^{i,j}$ and $h_e^{i,j}$ would form a blocking pair. Note that each single in $S^{i,j}$ is assigned to a hospital in $H^{i,j} \cup \{f^{i,j}\}$. As this holds for each $(i,j) \in \binom{[k]}{2}$, we get that $M(h_x^i) \subseteq S^i \cup \{a^i(1)\}$ holds for each $i \in [k], x \in [n]$.

Let $\nu(\sigma(i,j)) = (x,y)$ for some $(i,j) \in \binom{[k]}{2}$. Since $s^{i,j}_{\sigma(i,j)}$ prefers the hospitals in $Q^{i,j}_{\sigma(i,j)} = \{h^i_x, h^j_y\}$ to $f^{i,j}$, $M$ can only be stable if both $h^i_x$ and $h^j_y$ prefer their partner in $M$ to $s^{i,j}_{\sigma(i,j)}$. This implies $M(h^i_x) = \{a^i(1)\}$ and $M(h^j_y) = \{a^j(1)\}$. Thus, by defining $\sigma(i)$ to be $x$ if $M(a^i) = (h^i_x, g^i_{n+1-x})$ for each $i \in [k]$, we obtain $\nu(\sigma(i,j)) = (\sigma(i), \sigma(j))$. From the definition of $\nu$, this implies that $v_{\sigma(i)}$ and $v_{\sigma(j)}$ are adjacent in $G$. As this holds for every $(i,j) \in \binom{[k]}{2}$, we get that $\{v_{\sigma(i)} \mid i \in [k]\}$ is a clique in $G$.

Now, we prove (1) $\Rightarrow$ (3). If $v_{\sigma(1)}, v_{\sigma(2)}, \ldots, v_{\sigma(k)}$ form a clique in $G$, then define $\sigma(i,j)$ such that $\sigma(i,j) = \nu^{-1}(\sigma(i), \sigma(j))$. We define a stable assignment $M$ fulfilling property $\pi$ and covering every resident as follows.

$M(s^\alpha_{\sigma(\alpha)}) = f^\alpha$
$M(s^\alpha_x) = h^\alpha_x \quad$ if $x \in [N(\alpha)] \setminus \{\sigma(\alpha)\}$
$M(a^\alpha) = (h^\alpha_{\sigma(\alpha)}, g^i_{N(\alpha)+1-\sigma(\alpha)})$
$M(t^\alpha_x) = g^\alpha_x \quad$ if $1 \le x < N(\alpha) + 1 - \sigma(\alpha)$
$M(t^\alpha_x) = g^\alpha_{x+1} \quad$ if $N(\alpha) + 1 - \sigma(\alpha) \le x < N(\alpha)$

It is not hard to verify the stability of $M$ by simply checking all possibilities to find a blocking pair. (We note that many of the agents are only contained in $I^{G,k}$ to assure that a clique in $G$ indeed implies a stable assignment with the required properties.) As (3) $\Rightarrow$ (2) is trivial, this finishes the proof. □

Now, it is easy to prove Theorem 6.1.1.

*Proof of Theorem 6.1.1.* Let $G$ be an arbitrary graph and $k \in \mathbb{N}$. We construct two 1-uniform cmps $I_1$ and $I_2$, together with a stable assignment $M_2$ for $I_2$ such that the following three statements are equivalent:

(a) $G$ has a clique of size $k$,
(b) $I_1$ has a stable assignment,
(c) $I_2$ has a stable assignment of size greater than $|M_2|$.

Furthermore, the construction will take FPT time, and there will be $k + 3\binom{k}{2}$ couples in $I_1$, and $k + \binom{k}{2} + 1$ couples in $I_2$. Thus, (a) $\iff$ (b) yields an FPT reduction from Clique to Hospitals/Residents with Couples, and (a) $\iff$ (c) yields an FPT reduction from Clique to Increase Hospitals/Residents with Couples.

To get $I_1$, we simply combine the cmp $I_0$ having no stable assignment with the cmp $I^{G,k}$. This is done by introducing new couples $b^{i,j}$ and $c^{i,j}$, and new hospitals $\bar{f}^{i,j}_1$ and $\bar{f}^{i,j}_2$ for each $(i,j) \in \binom{[k]}{2}$, and adding these agents to $I^{G,k}$. We preserve the preference lists of $I^{G,k}$, except for hospitals $\{f^{i,j} \mid (i,j) \in \binom{[k]}{2}\}$, and we give the missing preference lists below.

$L(b^{i,j})\colon (f^{i,j}, \bar{f}^{i,j}_1), (\bar{f}^{i,j}_1, \bar{f}^{i,j}_2), (\bar{f}^{i,j}_2, f^{i,j})$
$L(c^{i,j})\colon (f^{i,j}, \bar{f}^{i,j}_2), (\bar{f}^{i,j}_1, f^{i,j}), (\bar{f}^{i,j}_2, \bar{f}^{i,j}_1)$
$L(\bar{f}^{i,j}_1) = L(\bar{f}^{i,j}_2)\colon c^{i,j}(1), b^{i,j}(1), b^{i,j}(2), c^{i,j}(2)$
$L(f^{i,j})\colon s^{i,j}_1, s^{i,j}_2, \ldots, s^{i,j}_m, c^{i,j}(1), b^{i,j}(1), b^{i,j}(2), c^{i,j}(2)$

Observe that if we restrict $I_1$ to contain only the hospitals $f^{i,j}, \bar{f}^{i,j}_1$ and $\bar{f}^{i,j}_2$ and the couples $b^{i,j}$ and $c^{i,j}$ for some $(i,j) \in \binom{[k]}{2}$, we obtain a cmp isomorphic to $I_0$, having no stable assignment. Therefore, any stable assignment $M$ must assign a single in $S^{i,j}$ to $f^{i,j}$, for each $(i,j) \in \binom{[k]}{2}$. The restriction of such an $M$ on the agents of $I^{G,k}$ must also be stable, because agents of $I^{G,k}$ cannot be assigned by $M$ to agents outside $I^{G,k}$. Thus, by Lemma 6.1.2, $G$ has a $k$-clique.

For the other direction, if there is a $k$-clique in $G$, then we can construct a stable assignment $M'_1$ for $I_1$ by setting $M'_1(b^{i,j}) = (\bar{f}^{i,j}_1, \bar{f}^{i,j}_2)$, $M'_1(c^{i,j}) = (\oslash, \oslash)$ for each $(i,j) \in \binom{[k]}{2}$,

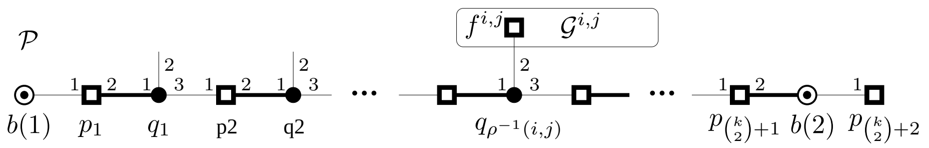


Figure 6.2: The path-gadget $\mathcal{P}$ in $I_2$. Bold edges represent $M_2$.

and $M_1'(r) = M_\pi^{G,k}(r)$ for the residents in $I^{G,k}$, where $M_\pi(G,k)$ is the stable assignment for $I^{G,k}$ with property $\pi$, guaranteed by Lemma 6.1.2. It is easy to see that $M_1'$ is stable, by using the stability of $M_\pi(G,k)$. This finishes the proof of the first claim.

To construct $I_2$, we add a *path-gadget* $\mathcal{P}$ to $I^{G,k}$ that contains the newly introduced hospitals $\{p_i \mid i \in [\binom{k}{2}+2]\}$, singles $\{q_i \mid i \in [\binom{k}{2}]\}$ and a couple $b$. See Figure 6.2 for an illustration. As before, we only modify the preferences of the hospitals $\{f^{i,j} \mid (i,j) \in \binom{[k]}{2}\}$, and we give the missing preference lists below. The notation $\rho$ used there denotes a bijection from $[\binom{k}{2}]$ into $\binom{[k]}{2}$.

| | |
|---|---|
| $L(p_1)\colon b(1), q_1$ | $L(p_i)\colon q_{i-1}, q_i$ if $1 < i \le \binom{k}{2}$ |
| $L(p_{\binom{k}{2}+1})\colon q_{\binom{k}{2}}, b(2)$ | $L(p_{\binom{k}{2}+2})\colon b(2)$ |
| $L(q_i)\colon p_i, f^{\rho(i)}, p_{i+1}$ | $L(f^{i,j})\colon s_1^{i,j}, s_2^{i,j}, \ldots, s_m^{i,j}, q_{\rho^{-1}(i,j)}, a^{i,j}(2)$ |
| $L(b)\colon (\oslash, p_{\binom{k}{2}+1}), (p_1, p_{\binom{k}{2}+2})$ | |

We also let $M_2(q_i) = p_i$ for each $i \in [\binom{k}{2}]$, $M_2(b) = (\oslash, p_{\binom{k}{2}+1})$, and $M_2(r) = M_0^{G,k}(r)$ for the residents in $I^{G,k}$, where $M_0^{G,k}$ is the stable assignment for $I^{G,k}$, provided by Lemma 6.1.2. Note that $M_2$ is indeed stable.

Suppose, there is a stable assignment $M$ for $I_2$ with $|M| > |M_2|$. Observe that $M_2$ covers each resident except for $b(1)$, so $M$ must cover every resident, implying $M(b) = (p_1, p_{\binom{k}{2}+2})$. Also, since $M(h)$ cannot be empty for any hospital $h$, we get $M(p_i) = \{q_{i-1}\}$ for each $i = \binom{k}{2}+1, \binom{k}{2}, \ldots, 2$. Thus, $f^{\rho(i)}$ is beneficial for $q_i$ for each $i \in [\binom{k}{2}]$, so by the stability of $M$ we obtain $M(f^{i,j}) \subseteq S^{i,j}$ for each $(i,j) \in \binom{[k]}{2}$. Again, the restriction of $M$ on the agents of $I^{G,k}$ must be stable, and so Lemma 6.1.2 implies that $G$ has a clique of size $k$.

Conversely, if there is a $k$-clique in $G$, then we can define a stable assignment $M_2'$ for $I_2$, covering each resident, as follows. We let $M_2'(q_i) = p_{i+1}$ for each $i \in [\binom{k}{2}]$, $M_2'(b) = (p_1, p_{\binom{k}{2}+2})$, and $M_2'(r) = M_\pi^{G,k}(r)$ for the residents in $I^{G,k}$. Again $M_2'$ is stable, and has size greater than $|M_2|$, proving the second claim. □

## 6.2 Local search

Here we examine the applicability of the local search approach for the Maximum Hospitals/Residents with Couples problem. Theorem 6.2.1 shows that no permissive local search algorithm is likely to exist for this problem running in FPT time with parameter $\ell$. However, if we regard the combined parameterization $(\ell, |C|)$, then even a strict local search algorithm with FPT running time can be given, as presented in Theorem 6.2.2.

**Theorem 6.2.1.** *There is no permissive local search algorithm for the 1-uniform* Maximum Hospitals/Residents with Couples *that runs in FPT time with parameter $\ell$, unless W[1] = FPT.*

*Proof.* Let $G$ be a graph and $k$ an integer. First, recall the cmp $I_2$ defined in the proof of Theorem 6.1.1, and observe that the assignment $M_2$ and the assignment $M_2'$, constructed

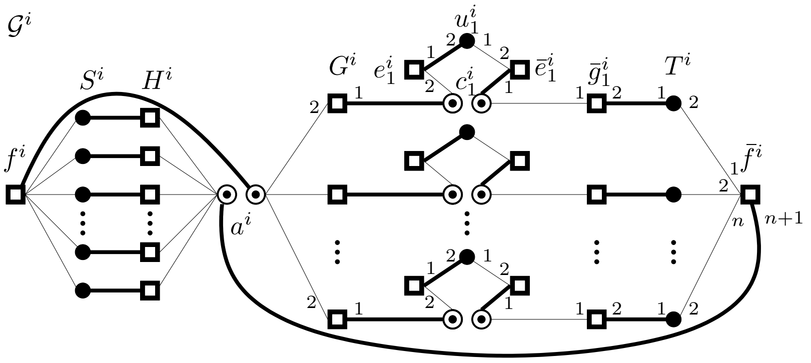


Figure 6.3: The modified node-gadget in the proof of Theorem 6.2.1. Bold edges represent $M_3$.

when a $k$-clique is present in $G$, may not be close to each other. Thus, in order to present an FPT-reduction here, we need to modify the node- and edge-gadgets of $I_2$. We are going to construct a cmp $I_3$ together with a stable assignment $M_3$ for it such that the following statements are equivalent:

(a) $G$ has a clique of size $k$.
(b) There is a stable assignment for $I_3$ with size at least $|M_3|+1$.
(c) There is a stable assignment for $I_3$ with size at least $|M_3|+1$ that is $\ell$-close to $M_3$ where $\ell = 8\binom{k}{2} + 7k + 2$.

The construction will take FPT time, hence a permissive local search algorithm for Maximum Hospitals/Residents with Couples that runs in FPT time with parameter $\ell$ can be used to solve Clique in FPT time.

See Figure 6.3 for an illustration of the modifications applied to $I_2$ in order to get $I_3$. For each node- or edge-gadget $\mathcal{G}^\alpha$, we take new singles $\{u_x^\alpha \mid x \in [N(\alpha)]\}$ and the single $t_{N(\alpha)}^\alpha$, new couples $\{c_x^\alpha \mid x \in [N(\alpha)]\}$, and new hospitals $\bigcup_{x\in[N(\alpha)]}\{\bar{g}_x^\alpha, e_x^\alpha, \bar{e}_x^\alpha\} \cup \{\bar{f}^\alpha\}$. For most of the agents we preserve the preferences originally defined for $I_2$. The modifications and the preference lists of the newly defined agents are as follows.

| | |
|---|---|
| $L(g_x^\alpha)\colon c_x^\alpha(1), a^\alpha(2)$ | $L(t_x^\alpha)\colon \bar{g}_x^\alpha, \bar{f}^\alpha$ |
| $L(e_x^\alpha)\colon u_x^\alpha, c_x^\alpha(1)$ | $L(u_x^\alpha)\colon \bar{e}_x^\alpha, e_x^\alpha$ |
| $L(\bar{e}_x^\alpha)\colon c_x^\alpha(2), u_x^\alpha$ | $L(c_x^\alpha) = (e_x^\alpha, \bar{g}_x^\alpha), (g_x^\alpha, \bar{e}_x^\alpha)$ |
| $L(\bar{g}_x^\alpha)\colon c_x^\alpha(2), t_x^\alpha$ | $L(\bar{f}^\alpha)\colon t_1^\alpha, t_2^\alpha, \dots, t_{N(\alpha)}^\alpha, a^\alpha(1)$ |

$L(a^\alpha)\colon (\bar{f}^\alpha, f^\alpha), (h_1^\alpha, g_{N(\alpha)}^\alpha), (h_2^\alpha, g_{N(\alpha)-1}^\alpha), \dots, (h_{N(\alpha)}^\alpha, g_1^\alpha)$

We also define $M_3(a^\alpha) = (\bar{f}^\alpha, f^\alpha)$, $M_3(c_x^\alpha) = (g_x^\alpha, \bar{e}_x^\alpha)$, $M_3(u_x^\alpha) = e_x^\alpha$ and $M_3(t_x^\alpha) = \bar{g}_x^\alpha$ for all possible values of $\alpha$ and $x$, and for each remaining resident $r$ let $M_3(r) = M_2(r)$. It is easy to observe that $M_3$ is stable, and covers each resident except for $b(1)$.

Supposing that there is a stable assignment $M$ with size greater than $|M_3|$ and using exactly the same arguments as in the proof of Theorem 6.1.1, we get $M(b) = (p_1, p_{\binom{k}{2}+2})$, $M(q_i) = (p_{i+1})$ for each $i \in [\binom{k}{2}]$, and $M(f^{i,j}) \subseteq S^{i,j}$ for each $(i,j) \in \binom{[k]}{2}$. By following the argument proving (2) $\Rightarrow$ (1) in Lemma 6.1.2, we again obtain that $G$ must have a $k$-clique. (The modifications of the gadgets in $I_3$ to do not affect that reasoning.) This proves (b) $\Rightarrow$ (a).

Clearly, (c) $\Rightarrow$ (b) is trivial, so we only have to prove (a) $\Rightarrow$ (c). Suppose that $G$ has a clique $\{v_{\sigma(i)} \mid i \in [k]\}$. We again let $\sigma(i,j) = \nu^{-1}(\sigma(i), \sigma(j))$, and we write $\sigma'(\alpha)$ for $N(\alpha) + 1 - \sigma(\alpha)$. We define a stable assignment $M_3'$ for $I$ in a very similar fashion as in the previous proofs:

$M_3'(b) = (p_1, p_{\binom{k}{2}+2})$ $\quad$ $M_3'(u^\alpha_{\sigma'(\alpha)}) = \bar{e}^\alpha_{\sigma'(\alpha)}$
$M_3'(q_i) = p_{i+1}$ for each $i \in [\binom{k}{2}]$ $\quad$ $M_3'(s^\alpha_{\sigma(\alpha)}) = f^\alpha$
$M_3'(a^\alpha) = (h^\alpha_{\sigma(\alpha)}, g^\alpha_{\sigma'(\alpha)})$ $\quad$ $M_3'(t^\alpha_{\sigma'(\alpha)}) = \bar{f}^\alpha$
$M_3'(c^\alpha_{\sigma'(\alpha)}) = (e^\alpha_{\sigma'(\alpha)}, \bar{g}^\alpha_{\sigma'(\alpha)})$

For each remaining resident $r$ we let $M_3'(r) = M_3(r)$. It is straightforward to verify that $M_3'$ is stable, and it is $\ell$-close to $M_0$. □

Before stating our last result, we describe the trick of "cloning" hospitals. For each hospital $h \in H$ in a given cmp, we take $f(h)$ copies of $h$ by replacing $h$ with new hospitals $h^1, \dots, h^{f(h)}$, each having capacity 1. The preference lists of these hospitals agree with the original preference list of $h$. For each single $s$ containing $h$ in its preference list, we replace $h$ in the list $L(s)$ by the series $h^1, \dots, h^{f(h)}$. For a couple $c$ containing a pair $(h, g)$ of two hospitals in $L(c)$, we replace $(h, g)$ by a series formed by the elements of $\{(h^i, g^j) : i \in [f(h)], j \in [f(g)]\}$ such that $(h^i, g^j)$ precedes $(h^{i'}, g^{j'})$ if $i < i'$, or $i = i'$ and $j < j'$. (We assume that the cases $h = \oslash$ and $g = \oslash$ are also clear.)

Now, if $M$ is an assignment for the original cmp $I$, then it defines an assignment $M^c$ for the cmp $I^c$ obtained by the above cloning process, as follows. If $M$ assigns $r$ to $h$ and there are $i-1$ residents in $M(h)$ that $h$ prefers to $r$, then let $M^c(r) = h^i$. If $M(r) = \oslash$ for some $r$, then we let $M^c(r) = \oslash$ as well. Observe that if $M$ is stable then $M^c$ is also stable. Conversely, it is not hard to see that a stable assignment for $I^c$ can be transformed in the straightforward way to a stable assignment for $I$.

**Theorem 6.2.2.** *There is a strict local search algorithm for* Maximum Hospitals/Residents with Couples *running in FPT time with combined parameter* $(\ell, |C|)$.

*Proof.* Let $I = (S, C, H, f, L)$ be given with the stable assignment $M_0$ and the integer $\ell$. W.l.o.g. we may assume that $f \equiv 1$, as otherwise we can apply the trick of cloning the hospitals, as argued above. Thus, if $M(r) = h$ for some resident $r$, then we will write $M(h) = r$ instead of $M(h) = \{r\}$.

Before describing the strict local search algorithm for Maximum Hospitals/Residents with Couples, we introduce some notation to capture the structure of the solution. The bipartite graph $G$ underlying $I$ has vertex set $H \cup R$ and edge set $E = \{hr \mid h \in H, r \in A^L(h)\}$. Clearly, an assignment $M$ for $I$ determines a matching $E(M)$ in $G$ in the natural way: $hr \in E(M)$ if and only if $M(r) = h$ for some resident $r$ and hospital $h$. Suppose that $M$ is a closest solution, i.e. a stable assignment for $I$ with $|M| > |M_0|$ and $d(M, M_0) \leq \ell$ that is the closest to $M_0$ among all such assignments. Let $A^\delta = \{a \in R \cup H \mid M(a) \neq M_0(a)\}$, and $E^\delta$ be the symmetric difference of $E(M_0)$ and $E(M)$. Note that $E^\delta$ covers exactly the vertices of $A^\delta$, and $G^\delta = (A^\delta, E^\delta)$ is the union of paths and cycles which contain edges from $M_0$ and $M$ in an alternating manner. It is well-known that for a cmp not containing couples, every stable assignment covers exactly the same agents [61]. Thus, it is easy to see that the stability of $M$ and $M_0$ imply that if a component of $G^\delta$ does not contain any resident from $R \setminus S$, then it must be a cycle. Let $\mathcal{K}_0$ denote the set of such cycles, and $\mathcal{K}_1$ the set of the remaining components of $G^\delta$. We write $C^\delta$ for $(R \setminus S) \cap A^\delta$, and we define $B(a) = \{b \mid a$ is beneficial for $b$ w.r.t. $M_0\}$ for every $a \in S \cup H$. We also let $S^+ = \{s \in S \mid M(s)$ is beneficial for $s$ w.r.t. $M_0\}$, and $S^- = \{s \in S \mid M_0(s)$ is beneficial for $s$ w.r.t. $M\}$. Note that $S^+ \cup S^- = S \cap A^\delta$. We define $H^+$ and $H^-$ analogously. We call agents in $A^+ = S^+ \cup H^+$ *winners* and agents in $A^- = S^- \cup H^-$ *losers*. For a simple illustration see Figure 6.4.

Now, we describe an algorithm that finds vertices of $A^\delta$. The algorithm first branches on guessing $|A^\delta|$ and a copy $\bar{G}$ of the graph $G^\delta$. Let $\varphi$ denote an isomorphism from $\bar{G}$ to $G^\delta$. The algorithm also guesses the vertex sets $\varphi^{-1}(C^\delta), \varphi^{-1}(H^+), \varphi^{-1}(H^-), \varphi^{-1}(S^+), \varphi^{-1}(S^-)$, and edge sets $\bar{E}_{M_0}$ and $\bar{E}_M$ denoting $\varphi^{-1}(E(M_0) \cap E^\delta)$ and $\varphi^{-1}(E(M) \cap E^\delta)$, respectively.

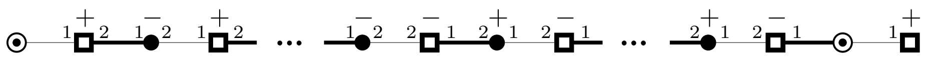


Figure 6.4: A possible component of $G^\delta$. Winners and losers are marked by '+' and '−' signs, respectively. Bold edges represent $M_0$, normal edges represent $M$.

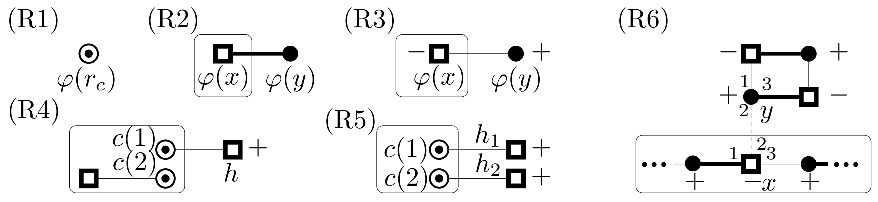


Figure 6.5: Figure (R$i$) shows a subgraph of $G^\delta$ illustrating Extension Rule $i$, for $i \in \{1,2,\ldots,6\}$. We represent agents of $\varphi(X)$ by enclosing them in a rectangular box. Bold edges represent $\bar{E}_{M_0}$ and normal edges represent $\bar{E}_M$.

Since $|A^\delta| \leq 2\ell$, it can be achieved by careful implementation that the algorithm branches into at most $(2\ell)6^{2\ell}$ directions in this phase.

Next, we apply the technique of color-coding [8], in order to help the localization of $A^\delta$. To this end, the algorithm colors the vertices of $G$ with $|A^\delta| \leq 2\ell$ colors randomly with uniform and independent distribution; $\gamma(a)$ denotes the color of $a$. The coloring $\gamma$ is *nice*, if $\gamma(\varphi(a)) = \Gamma(a)$ for each $a \in V(\bar{G})$, where $\Gamma$ is an arbitrary fixed ordering of $V(\bar{G})$, i.e. a bijection from $V(\bar{G})$ to $[|A^\delta|]$. From now on, we suppose that $\gamma$ is nice, which clearly holds with probability $|A^\delta|^{-|A^\delta|} \geq (2\ell)^{-2\ell}$.

Given a coloring, the algorithm grows a subset $X \subseteq V(\bar{G})$ on which $\varphi$ is already known. It applies the following extension rules repeatedly, until none of them is applicable. When Extension Rule 1 is applied, the algorithm branches into at most $2|C|$ branches, but no other branchings are involved. We write $\bar{X} = V(\bar{G}) \setminus X$. See Figure 6.5 for an illustration.

**Extension Rule 1 [guessing a member of a couple]**: applicable if $c \in \bar{X} \cap \varphi^{-1}(C^\delta)$. In this case, we simply branch on the vertices of $(R \setminus S) \cap \{a \mid \gamma(a) = \Gamma(c)\}$ to choose $\varphi(c)$. Note that this means at most $2|C|$ branches.

**Extension Rule 2 [finding pairs by $M_0$]**: applicable if $x \in X, y \in \bar{X}$ and $xy \in \bar{E}_{M_0}$ for some $x$ and $y$. Clearly, we get $\varphi(y) = M_0(\varphi(x))$, so we can extend $\varphi$ by adding $y$ to $X$.

**Extension Rule 3 [finding pairs by $M$ for losers]**: applicable if $x \in X \cap \varphi^{-1}(A^-)$, $y \in \bar{X} \cap \varphi^{-1}(A^+)$ and $xy \in \bar{E}_M$ for some $x$ and $y$. Let $y^*$ be the first element in the preference list $L(\varphi(x))$ contained in the set $B(\varphi(x))$ having color $\Gamma(y)$. We claim $y^* = \varphi(y)$. Clearly, $\varphi(y) \in B(\varphi(x))$ holds because $\varphi(y)$ is a winner, and its color must be $\Gamma(y)$ as $\gamma$ is nice. Now, suppose for contradiction that $y^*$ precedes $\varphi(y)$ in $L(\varphi(x))$. Since the only vertex in $A^\delta$ having color $\Gamma(y)$ is $\varphi(y)$, we get $M(y^*) = M_0(y^*)$ implying that $y^*$ and $\varphi(x)$ form a blocking pair for $M$. Thus, $\varphi(y) = y^*$ can be found in linear time, so we can extend $\varphi$ by adding $y$ to $X$.

**Extension Rule 4 [finding pairs by $M$ for couples with one winner hospital]**: applicable if $c(i) \in C^\delta \cap \varphi(X)$, $y \in \varphi^{-1}(H^+) \cap \bar{X}$, $\varphi^{-1}(c(i))y \in \bar{E}_M$, and $M(c(i'))$ is already known for some $c \in C$, $i \neq i'$ and $y$. W.l.o.g. we assume $i = 1$. Let $h$ be defined such that $(h, M(c(2)))$ is the first element in $L(c)$ for which $h \in B(c(1))$ and $h$ has color $\Gamma(y)$. We claim $\varphi(y) = h$. Observe that $\varphi(y) \in B(c(1))$ must hold because $\varphi(y)$ is a winner. As $\gamma$ is nice, $\varphi(y)$ indeed has color $\Gamma(y)$. Thus, if $h \neq \varphi(y)$ then $(h, M(c(2)))$ precedes $(\varphi(y), M(c(2)))$ in $L(c)$, but this implies that the couple $c$ and $(h, M(c(2)))$ form a blocking pair for $M$.

Therefore we get $\varphi(y) = h$, and we can extend $\varphi$ in linear time by adding $y$ to $X$.

**Extension Rule 5 [finding pairs by $M$ for couples with two winner hospitals]**: applicable if $c(i) \in C^\delta \cap \varphi(X)$, $y_i \in \varphi^{-1}(H^+) \cap \bar{X}$, and $\varphi^{-1}(c(i))y_i \in \bar{E}_M$ holds for both $i \in \{1,2\}$, for some $c \in C$, $y_1$ and $y_2$. We let $(h_1, h_2)$ be the first element in $L(c)$ such that $h_i \in B(c(i))$ and $\gamma(h_i) = \Gamma(y_i)$ for both $i \in \{1,2\}$. Using the same arguments as in the previous case, we can argue that $\varphi(y_1) = h_1$ and $\varphi(y_2) = h_2$ hold. Thus, in this case we can extend $\varphi$ in linear time by adding both $y_1$ and $y_2$ to $X$.

**Extension Rule 6 [dissolving a blocking pair]**: applicable if $M(a) \in \varphi(X)$ if and only if $a \in \varphi(X)$ for all $a \in A^\delta$, and $xy$ is a blocking pair for the *actual assignment* $M_X$. We define $M_X$ by setting $M_X(a) = M_0(a)$ if $a \notin \varphi(X)$ and $M_X(a) = M(a)$ if $a \in \varphi(X)$, for each agent $a$. Note that by our first condition, $M_X$ is indeed an assignment. Now, as $xy$ cannot be a blocking pair for $M$ or $M_0$, either $x \in \varphi(X)$ and $y \in A^\delta \setminus \varphi(X)$, or vice versa. W.l.o.g. we suppose the former. By defining $\bar{y} \in V(\bar{G})$ such that $\Gamma(\bar{y}) = \gamma(y)$, it can be seen that $\varphi(\bar{y}) = y$ must hold because $\gamma$ is nice. Thus, $\varphi$ can be extended by adding $\bar{y}$ to $X$.

**Lemma 6.2.3.** *If none of the extension rules is applicable, then $\varphi(X) = A^\delta$.*

*Proof.* First, $\varphi(X) \supseteq C^\delta$ is trivial, as Extension Rule 1 is not applicable.

**Claim 1:** $\varphi(X) \supseteq (H^- \cup S^+) \cap V(\mathcal{K}_1)$. Suppose $a \in (H^- \cup S^+) \cap V(\mathcal{K}_1) \setminus \varphi(X)$ is chosen such that the distance $d^C(a)$ is minimal, where $d^C(a)$ is the minimum length of a path $P$ in $G^\delta$ from $a$ to some $c \in C^\delta$ such that the first edge of $P$ is in $E(M_0)$ if $a \in H$ and it is in $E(M)$ if $a \in S$. If no such path exists then let $d^C(a) = \infty$.

First, if $a$ is a winner single, then $M(a) \neq \oslash$, and since $a$ and $M(a)$ cannot be a blocking pair for $M_0$, $M(a)$ must be a loser hospital. Now, if $M(a) \in \varphi(X)$ then Extension Rule 3 is applicable, a contradiction. Thus $M(a) \notin \varphi(X)$, but as $M(a)$ is on the path defining $d^C(a)$, we get $d^C(M(a)) < d^C(a)$ contradicting to the choice of $a$. (Note that $d^C(a) \neq \infty$ as $a \in V(\mathcal{K}_1)$.) Second, if $a$ is a loser hospital, then $M_0(a) \neq \oslash$. Observe that if $M_0(a) \in \varphi(X)$ then Extension Rule 2 is applicable, which cannot be the case, so $M_0(a)$ can only be a single in $S \setminus \varphi(X)$. If $M_0(a)$ were a loser, then $a$ and $M_0(a)$ would form a blocking pair for $M$, so we obtain $M_0(a) \in S^+ \setminus \varphi(X)$. But this implies $d^C(M_0(a)) < d^C(a)$, a contradiction. Thus, $\varphi(X)$ indeed contains $(H^- \cup S^+) \cap V(\mathcal{K}_1)$.

**Claim 2:** $\varphi(X) \supseteq V(\mathcal{K}_1)$. By Claim 1, we only have to prove that $(H^+ \cup S^-) \cap V(\mathcal{K}_1) \setminus \varphi(X)$ is empty. Analogously as in Claim 1, we choose $a \in (H^+ \cup S^-) \cap V(\mathcal{K}_1) \setminus \varphi(X)$ such that the distance $d'^C(a)$ is minimal, where $d'^C(a)$ is the minimum length of a path $P$ in $G^\delta$ from $a$ to some $c \in C^\delta$ such that the first edge of $P$ is in $E(M)$ if $a \in H$ and it is in $E(M_0)$ if $a \in S$. If no such path exists then let $d'^C(a) = \infty$. Note that $d'^C \neq d^C$, as the requirements for the first edge of the path $P$ are different.

First, if $a$ is a loser single, then $M_0(a) \neq \oslash$, and since $a$ and $M_0(a)$ cannot be a blocking pair for $M$, $M_0(a)$ must be a winner hospital. Now, if $M_0(a) \in \varphi(X)$ then Extension Rule 2 is applicable, a contradiction. Thus $M_0(a) \notin \varphi(X)$, but as $M_0(a)$ is on the path defining $d'^C(a)$, we get $d'^C(M_0(a)) < d'^C(a)$ contradicting to the choice of $a$. Again, $d'^C(a) \neq \infty$ as $a \in V(\mathcal{K}_1)$.

Second, if $a$ is a winner hospital, then $M(a) \neq \oslash$. Observe that if $M(a)$ is a member of some couple $c$, then if $M(c(i))$ is not known for some $i \in \{1,2\}$, then $M(c(i))$ can only be a winner hospital by Claim 1, so Extension Rule 4 or 5 is applicable. If $M(a)$ were a winner single, then $a$ and $M(a)$ would form a blocking pair for $M_0$, so we obtain $M(a) \in S^-$. Now, if $M(a) \in S^- \cap \varphi(X)$ then Extension Rule 3 is applicable. Thus, only $M(a) \in S^- \setminus \varphi(X)$ is possible. But this implies $d'^C(M(a)) < d'^C(a)$, which is a contradiction proving Claim 2.

**Claim 3:** $\varphi(X) \supseteq V(\mathcal{K}_0)$. As already mentioned, each component of $\mathcal{K}_0$ is a cycle, and it easy to see that it must contain vertices from $A^+$ and $A^-$ in an alternating manner. Thus, if neither of Extension Rule 2 and 3 is applicable, then each component of $\mathcal{K}_0$ is totally contained in either $A^\delta \setminus \varphi(X)$ or in $\varphi(X)$. Therefore, the first condition of Extension Rule 6 must hold. Now, if $\varphi(X) \neq A^\delta$ then clearly $M_X \neq M$. As $M_X$ is closer to $M_0$ than $M$, and $M$ is a closest solution, $M_X$ cannot be stable. Thus Extension Rule 6 is applicable, a contradiction.

Now, Claims 1, 2, and 3 together imply the lemma. □

If no extension rule is applicable, then we easily obtain the solution $M$ by Lemma 6.2.3. Each step takes linear time, the number of steps is at most $2\ell$, and the algorithm branches into at most $(2\ell)6^{2\ell}(2|C|)^\ell$ branches in total, thus the overall running time is $O(\ell(72|C|)^\ell|I|)$. The output is correct if the coloring $\gamma$ is nice, which holds with probability at least $(2\ell)^{-2\ell}$. To derandomize the algorithm, we can use the standard method of $k$-perfect hash functions [8] instead of randomly coloring $V(G)$. This yields a running time of $O(\ell^{O(\ell)}|C|^\ell|I|\log|I|)$. □

## 6.3 Maximum matching without preferences

In this section, we define a variant of the Hospitals/Residents problem that involve couples, but do not deal with preferences, using only a notion of acceptability instead.

A *couples' market with acceptance*, or shortly *cma* is a tuple $(S, C, H, f, A)$. Here, the sets $S, C$ and $H$ denote the set of singles, couples and hospitals, respectively, and $f(h)$ denotes the capacity of the hospital $h$ for each $h \in H$. But instead of describing the preferences, we only define a set $A(s) \subseteq H$ for each single $s \in S$ representing *acceptable hospitals* for $s$, and a set $A(c) \subseteq \widetilde{H}$ for each couple $c \in C$ representing *acceptable hospital pairs* for $c$. Here, $\widetilde{H}$ and the symbol $\oslash$ are defined the same way as before. We also keep the definitions of agents, resident, and $f_0$-uniformity as given for a couples market with preference. To define an *assignment* for a cma $(S, C, H, f, A)$, we also refer to the corresponding definition for a cmp. Note that this definition only relies on the concept of acceptability, and has no direct reference to the preference lists.

The Maximum Matching with Couples problem is an optimization problem, where given a cma $I$, the set of solutions is the set of assignments for $I$, and the task is to find an assignment for $I$ of maximum size. We investigate the parameterized complexity of this problem in Section 6.3.1. We also examine the possibility of finding a local search algorithm for it in Section 6.3.2.

### 6.3.1 Parameterized complexity

First, we investigate a slightly modified version of Maximum Matching with Couples. We say that an assignment for a cma $(S, C, H, f, A)$ is a $(k, n)$-*assignment* if it covers at least $k$ couples and at least $n$ singles. We denote the following problem as $(k, n)$-Matching with Couples: given a cma $I$ and two integers $k$ and $n$, find a $(k, n)$-assignment for $I$ if possible.

Clearly, if there are no couples in a given instance, then this problem is equivalent to finding a maximum matching in a bipartite graph, and can be solved by standard techniques. If couples are involved, the problem becomes hard. More precisely, the decision version of this problem is NP-complete [66, 12], even in the following special case: each hospital has capacity 2, and the acceptable hospital pairs for a couple are always of the form $(h, h)$ for some $h \in H$. However, if the number of couples is small, which is a reasonable assumption in many practical applications, $(k, n)$-Matching with Couples becomes tractable, as shown by Theorem 6.3.1.

**Theorem 6.3.1.** $(k, n)$-Matching with Couples *can be solved in randomized FPT time with parameter* $|C|$.

To prove Theorem 6.3.1, we need some results from [100] concerning matroids.

Although we only use basic concepts of matroid theory, here we give a brief outline of the main definitions used. For some set $U$ and collection $\mathcal{I} \subseteq 2^U$, the pair $(U, \mathcal{I})$ is a *matroid* if the followings hold: (1) $\emptyset \in \mathcal{I}$, (2) if $X \in \mathcal{I}$ and $X' \subseteq X$ then $X' \in \mathcal{I}$, and (3) if $X, Y \in \mathcal{I}$ and $|X| < |Y|$ then $X \cup \{y\} \in \mathcal{I}$ for some $y \in Y \setminus X$. The elements of $\mathcal{I}$ are called *independent sets.* A matrix $A$ over a field $F$ is a *linear representation* of a matroid $(\{u_i \mid i \in [n]\}, \mathcal{I})$, if for any set $J$ of indices in $[n]$, the set of columns in $A$ corresponding to the indices $J$ are independent over $F$ if and only if $\{u_j \mid j \in J\} \in \mathcal{I}$. A matroid is *linear* if it admits a linear representation. A maximal independent set of a matroid is called a *basis* of the matroid. The *dual* of a matroid $(U, \mathcal{I})$ with basis set $\mathcal{B}$ is the matroid with ground set $U$ whose basis set is $\{U \setminus B \mid B \in \mathcal{B}\}$. The *k-truncation* of $(U, \mathcal{I})$ is the matroid $(U, \mathcal{I}')$ where $I \in \mathcal{I}'$ if and only if $I \in \mathcal{I}$ and $|I| \leq k$. Given a bipartite graph $G(A, B; E)$, its *transversal matroid* has ground set $A$, and $X$ is defined to be independent if there is a matching in $G$ covering $X$.

**Theorem 6.3.2** ([100])**.** *Let $\mathcal{M}(U, \mathcal{I})$ be a linear matroid where the ground set $U$ is partitioned into blocks of size $b$. Given a linear representation $A$ of $\mathcal{M}$, it can be determined in $f(k, b) \cdot ||A||^{O(1)}$ randomized time whether there is an independent set that is the union of $k$ blocks. ($||A||$ denotes the length of $A$ in the input.)*

**Corollary 6.3.3.** *Let $\mathcal{M}(U, \mathcal{I})$ be a linear matroid and let $\mathcal{X} = \{X_1, X_2, \dots X_n\}$ be a collection of subsets of $U$, each of size $b$. Given a linear representation $A$ of $\mathcal{M}$, it can be determined in $f(k, b) \cdot ||A||^{O(1)}$ randomized time whether there is an independent set that is the union of $k$ disjoint sets in $\mathcal{X}$.*

*Proof.* First, let us make $n(u)$ copies for each $u \in U$, where $n(u)$ is the number of sets in $\mathcal{X}$ containing $u$, i.e. let $U' = \{u^i \mid u \in U, n(u) > 0, i \in [n(u)]\}$. Let $\mathcal{M}'(U', \mathcal{I}')$ be the matroid where $\mathcal{I}'$ contains those sets which can be obtained from some set $I \in \mathcal{I}$ by replacing each $u \in I$ with an arbitrary element from $\{u^i \mid i \in [n(u)]\}$. A representation $A'$ of $\mathcal{M}'$ can be obtained from $A$ by putting $n(u)$ copies of the column representing $u$ into $A'$ for each $u \in U$. For each $i \in [n]$, let $X'_i \subseteq U'$ be obtained by replacing each element $u$ in $X_i$ with $u^j$ if $X_i$ is the $j$-th set in $\mathcal{X}$ containing $u$. Clearly, by letting $X'_i$ to be a block (having size $b$) for each $i \in [n]$, we get a partition of $U'$.

The sets $\{X_{i_j} \mid j \in [k]\}$ satisfy the requirements (being disjoint and having an independent union in $\mathcal{M}$) if and only if the sets $\{X'_{i_j}) \mid j \in [k]\}$ are $k$ blocks whose union is independent in $\mathcal{M}'$, and thus the algorithm of Theorem 6.3.2 provides the solution. □

**Lemma 6.3.4** ([100])**.** *(1) Given a representation $A$ over a field $F$ of a matroid $\mathcal{M}$, a representation of the dual matroid $\mathcal{M}^*$ over $F$ can be found in polynomial time. (2) Given a representation $A$ over $\mathbb{N}$ of a matroid $\mathcal{M}$ and an integer $k$, a representation of the $k$-truncation of $\mathcal{M}^k$ can be found in randomized polynomial time. (3) Given a bipartite graph $G(A, B; E)$, a representation of its transversal matroid over $\mathbb{N}$ can be constructed in randomized polynomial time.*

Now, we are ready to prove Theorem 6.3.1.

*Proof of Theorem 6.3.1.* Let $(S, C, H, f, A)$ be the cma for which we have to find a $(k, n)$-assignment. W.l.o.g. we can assume that each hospital has capacity 1 as otherwise we can "clone" the hospitals, i.e. for each $h \in H$ we can substitute $h$ with the newly introduced hospitals $h^1, \dots, h^{f(h)}$, also modifying $A(p)$ for each $p \in S \cup C$ appropriately. (As $f(h) \leq |S| + 2|C|$ can be assumed, this increases the input size only polynomially.) Note that the case $k < |C|$ can be solved by finding a $(k, n)$-assignment for $(S, C', H, f, A')$ for every $C' \subseteq C$ where $|C'| = k$ and $A'$ is the restriction of $A$ on $S \cup C'$. As this increases the running time only with a factor of at most $2^{|C|}$, it is sufficient to give an FPT algorithm for the case $|C| = k$.

Moreover, we can assume $A(c) \subseteq H \times H$, since for each $c \in C$ we can eliminate each pair of the form $(h, \oslash)$ or $(\oslash, h)$ $(h \in H)$ in the set $A(c)$ by adding a new hospital $u_c$ to $H$ with capacity 1 and substituting the symbol $\oslash$ with $u_c$ in the pairs contained in $A(c)$.

Now, let $G(H, S; E)$ be the bipartite graph where a single $s \in S$ is connected with a hospital $h \in H$ if and only if $h \in A(s)$. We can assume w.l.o.g. that $G$ has a matching of size at least $n$ as otherwise no solution may exist, and this case can be detected easily in polynomial time. We define $\mathcal{M}(H, \mathcal{I})$ to be the matroid where a set $X \subseteq H$ is independent if and only if there is a matching in $G$ that covers at least $n$ singles but covers no hospitals from $X$. Observe that $\mathcal{M}$ is exactly the dual of the $n$-truncation of the transversal matroid of $G$, and thus it is indeed a matroid. By Lemma 6.3.4, we can find a linear representation $A$ of $\mathcal{M}$ in randomized polynomial time.

We define the matroid $\mathcal{M}'(U, \mathcal{I}')$ with ground set $U = H \cup C$ such that $X \subseteq U$ is independent in $\mathcal{M}'$ if $X \cap H$ is independent in $\mathcal{M}$. A representation of $\mathcal{M}'$ can be obtained by taking the direct sum of the matrices $A$ and $E_k$ where $E_k$ is the unit matrix of size $k \times k$. Let $\mathcal{X}$ be the collection of the sets that are of the form $\{c, h_1, h_2\}$ where $c \in C$ and $(h_1, h_2) \in A(c)$.

Observe that if $X_1, X_2, \ldots, X_k$ are $k$ disjoint sets in $\mathcal{X}$ whose union is independent in $\mathcal{M}'$, then we can construct a $(k, n)$-assignment as follows. For each $\{c, h_1, h_2\} \in \{X_1, \ldots, X_k\}$ we choose $M(c)$ from $\{(h_1, h_2), (h_2, h_1)\} \cap A(c)$ arbitrarily. The disjointness of the sets $X_1, \ldots, X_k$ guarantees that this way we assign exactly one resident to each hospital in $X = \bigcup_{i \in [k]} X_i \cap H$. Now, let $N$ be a matching in $G$ that covers at least $n$ singles, but no hospitals from $X$. Such a matching exists, as $X$ is independent in $\mathcal{M}$. Thus, letting $M(s)$ to be $N(s)$ if $s$ is covered by $N$ and $\oslash$ otherwise for each $s \in S$ yields that $M$ is a $(k, n)$-assignment. Conversely, if $M$ is a $(k, n)$-assignment then the sets $\{c, h_1, h_2\}$ for each $c \in C$ and $M(c) = (h_1, h_2)$ form a collection of $k$ disjoint sets in $\mathcal{X}$ whose union is independent in $\mathcal{M}'$. By Corollary 6.3.3, such a collection can be found in randomized FPT time when $k$ is the parameter, yielding a solution if existent. □ □

Theorem 6.3.1 shows that $(k, n)$-Matching with Couples can be solved by a randomized FPT algorithm with parameter $|C|$. We remark that our method can also be applied when there are groups of fixed size instead of couples in the given market, or when the task is to maximize (or minimize) some arbitrary function $f(k, n)$ where $k$ and $n$ are the number of covered couples and singles, respectively, in the solution. This latter can be done by simply searching for a $(k, n)$-assignment for all possible and relevant values of $k$ and $n$. Thus, as a consequence we get that Maximum Matching with Couples can also be solved in randomized FPT time with parameter $|C|$.

Theorem 6.3.1 also has a useful consequence in connection to the following scheduling problem. We are given a set of parallel machines, a set of independent jobs, and a set of job assignment restrictions describing for each job $j$ the set of machines which $j$ may be assigned to. The task is to find a minimum makespan assignment of the jobs to the machines respecting the given restrictions. Considering the remarks of the previous paragraph, we get that Theorem 6.3.1 yields a randomized FPT algorithm for the special case of this problem, where $k$ jobs have processing time $p \in \mathbb{N}$ and all other jobs have processing time 1, and we regard $k$ as a parameter.

### 6.3.2 Local search

Here, we investigate the applicability of the local search approach to handle the intractability of the Maximum Matching with Couples problem. We define the *distance* $d(M, M')$ of two assignments $M$ and $M'$ for some cma $I$ as the number of residents $r$ for which $M(r) \neq M'(r)$. The following theorem shows that no permissive local search algorithm is likely to run in FPT time, if the parameter is the radius of the explored neighborhood and we restrict the problem to the 2-uniform case.

**Theorem 6.3.5.** *There is no permissive local search algorithm for the 2-uniform* Maximum Matching with Couples *that runs in FPT time with parameter $\ell$ denoting the radius of the explored neighborhood, unless W[1] = FPT.*

*Proof.* Let $G$ be the input graph for the Clique problem and $k$ be the parameter given. We denote the vertices of $G$ by $v_1, v_2, \dots, v_n$. We claim that if there is a permissive local search algorithm $\mathcal{A}$ for Maximum Matching with Couples running in FPT time with parameter $\ell$, then we can use $\mathcal{A}$ to solve Clique in FPT time. To prove this, we construct an input $\Lambda = (I, M_0, \ell)$ of $\mathcal{A}$ with the following properties: every assignment for $I$ with size at least $|M_0| + 1$ is $\ell$-close to $M_0$, and there is such an assignment for $I$ if and only if $G$ has a clique of size $k$. Thus, $G$ has a clique of size $k$ if and only if $\mathcal{A}$ outputs an assignment for $I$ with size at least $|M_0| + 1$.

To construct $\Lambda$, we first define the cma $I$ together with the assignment $M_0$ for it. Let the set $H$ of hospitals be the union of $D = B \cup \bigcup\{H^{i,j} \mid i,j \in [k]\}$, $D' = B' \cup \bigcup\{H'^{i,j} \mid i,j \in [k]\}$ and $F = \{f_i \mid i \in [k]\}$, where $B = \{b_i \mid i \in [2k-1]\}$, $H^{i,i} = \{h^{i,i}_{j,j} \mid j \in [n]\}$ for each $i \in [k]$, $H^{i,j} = \{h^{i,j}_{x,y} \mid v_x v_y \in E(G)\}$ for each $i \neq j, \{i,j\} \subseteq [k]$, and for each hospital $h$ in $B$ ($H^{i,j}$, respectively) we also define a hospital $h'$ to be in $B'$ ($H'^{i,j}$, respectively). For brevity, we will use the notation $H^{i,j}_{h,\bullet} = \{h \mid \exists y : h = h^{i,j}_{h,y} \in H^{i,j}\}$ and $H^{i,j}_{\bullet,h} = \{h \mid \exists x : h = h^{i,j}_{x,h} \in H^{i,j}\}$. The capacity of each hospital is 2. For each hospital $h \in D$ we define a couple denoted by $c(h)$, and for each $h' \in D'$ we define two singles $s_1(h')$ and $s_2(h')$. Let $C = \{c(h) \mid h \in D\}$ and let $S = \{s_0\} \cup \{s_i(h') \mid h' \in D', i \in \{1,2\}\}$.

Before defining $A(p)$ for each $p \in S \cup C$, we define the assignment $M_0$ for $I$, as this will not cause any confusion. Let $M_0(s_0) = \oslash$, and let $M_0(p) = h$ where either $h \in D$ and $p$ is a member of the couple $c(h)$, or $h \in D'$ and $p \in \{s_1(h), s_2(h)\}$. Now, for each $p \in S \cup C$, we define the set of acceptable hospitals or pairs of hospitals $A(p)$ to be the union of $\{M_0(p)\}$ and the set $A'(p)$ of hospitals, defined below, that can be assigned to $p$ besides $M_0(p)$. We define $A'(p)$ for each $p \in S \cup C$ as follows.

$A'(c(h)) = \{(h', h')\}$ for each $h \in D$
$A'(s_0) = \{b_1\}$
$A'(s_1(b'_i)) = H^{1,i}$ for each $i \in [k]$
$A'(s_2(b'_i)) = \{b_{i+1}\}$ for each $i \in [k]$
$A'(s_1(b'_{k+i})) = H^{i,1}$ for each $i \in [k-1]$
$A'(s_2(b'_{k+i})) = \{b_{k+i+1}\}$ for each $i \in [k-2]$
$A'(s_2(b'_{2k-1})) = H^{k,1}$
$A'(s_1(h'^{i,j}_{x,y})) = H^{i,j+1}_{x,\bullet}$ for each $i \in [k], j \in [k-1]$ and every possible $x$ and $y$
$A'(s_1(h'^{i,k}_{x,y})) = \{f_i\}$ for each $i \in [k]$ and every possible $x$ and $y$
$A'(s_2(h'^{i,j}_{x,y})) = H^{i+1,j}_{\bullet,y}$ for each $i \in [k-1], j \in [k]$ and every possible $x$ and $y$
$A'(s_2(h'^{k,i}_{x,y})) = \{f_i\}$ for each $i \in [k]$ and every possible $x$ and $y$

This completes the definition of the cma $I = (S, C, H, f, A)$. Observe that $M_0$ indeed is an assignment for $I$. Finally, setting $\ell = 4k^2 + 8k - 3$ finishes the definition of the instance $\Lambda = (I, M_0, \ell)$. Figure 6.6 shows an illustration.

First, suppose that $M$ is an assignment for $I$ such that $|M| > |M_0|$. We do not require $M$ to be $(4k^2 + 8k - 3)$-close to $M_0$, but we will actually prove that this is necessary. Observe that $M_0$ covers each resident except for $s_0$, so $M$ must cover all residents to satisfy $|M| > |M_0|$. As $A(s_0) = \{b_1\}$, $M$ must assign $b_1$ to $s_0$. This implies $M(c(b_1)) = (b'_1, b'_1)$, and therefore we also have $M(s_2(b'_1)) = b_2$, implying $M(c(b_2)) = (b'_2, b'_2)$, and so on. Following this argument, it can be seen that $M(c(b_i)) = (b'_i, b'_i)$ for every $i \in [2k-1]$, and $M(s_2(b'_i)) = b_{i+1}$ for every $i \in [2k-2]$.

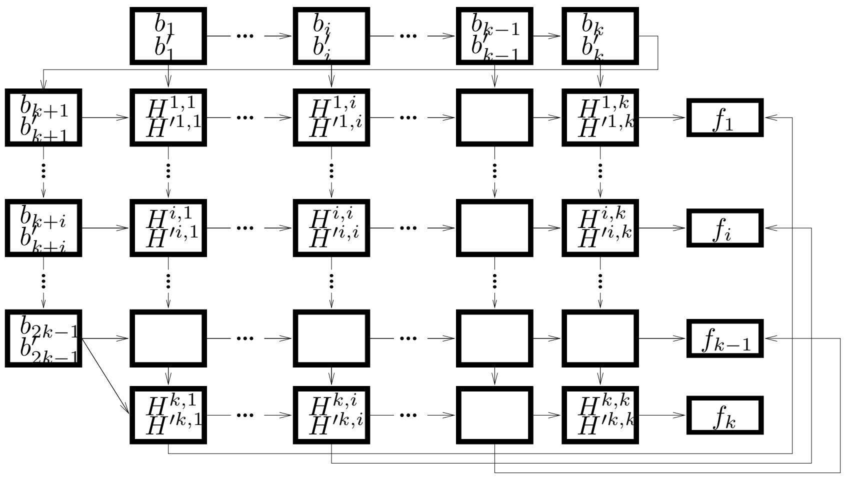


Figure 6.6: A block diagram showing the hospitals in the proof of Theorem 6.3.5. For two sets $H_1, H_2$ of hospitals, $(H_1, H_2)$ is an arc if $A'(s) \subseteq H_2$ for some $s \in S$ with $M_0(s) \in H_1$.

We say that a single $s$ *enters* $H^{i,j}$ if $M(s) \in H^{i,j}$ but $M_0(s) \notin H^{i,j}$, and *leaves* $H'^{i,j}$ if $M_0(s) \in H'^{i,j}$ but $M(s) \notin H'^{i,j}$. A couple $c$ *moves* from a hospital $h$ if $M_0(c) = (h,h) \neq M(c)$, and we say that $c$ moves from a set $J \subseteq H$ of hospitals if it moves from a hospital in $J$. Observe that if $c$ moves from $H^{i,j}$, then two singles leave $H'^{i,j}$, one of them entering $H^{i+1,j}$ if $i \neq k$, and the other entering $H^{i,j+1}$ if $j \neq k$. If a single $s$ leaves $H'^{i,j}$ but does not enter $H^{i+1,j}$ or $H^{i,j+1}$, then $M(s) \in F$ must hold, and therefore there can exist at most $2k$ such single $s$. Moreover, if a set of $m$ singles enter $H^{i,j}$ then at least $\lceil m/2 \rceil$ couples have to move from $H^{i,j}$. For each $i \in [k]$, exactly one single from $\{s_1(b'_1), s_1(b'_2), \dots, s_1(b'_k)\}$ enters $H^{1,i}$, and exactly one single from $\{s_1(b'_{k+1}), s_1(b'_{k+2}), \dots, s_1(b'_{2k-1}), s_2(b'_{2k-1})\}$ enters $H^{i,1}$. These altogether imply that exactly one couple moves from $H^{i,j}$ for each $i,j \in [k]$, and that if $s$ and $s'$ enter $H^{i,j}$ then $M(s) = M(s')$ must hold.

Suppose that $c$ moves from the hospital $h^{i,j}_{x,y}$. Observe that if $j < k$ then a couple must move from $H^{i,j+1}_{x,\bullet}$, and similarly, if $i < k$ then a couple must move from $H^{i+1,j}_{\bullet,y}$. For each $i \in [k]$, letting $\sigma_h(i)$ to be $x$ if for some $j$ and $y$ a couple moves from $h^{i,j}_{x,y}$, and $\sigma_v(i)$ to be $y$ if for some $j$ and $x$ a couple moves from $h^{i,j}_{x,y}$, we obtain that $\sigma_h(i)$ and $\sigma_v(i)$ are well-defined. Observe that by the definition of $H^{i,i}$ we get $\sigma_h(i) = \sigma_v(i) := \sigma(i)$, and from the definition of $H^{i,j}$ we get that if $\sigma(i) = x$ and $\sigma(j) = y$ for some $i \neq j$, then $v_x v_y$ must be an edge in $G$. Thus, the set $\{v_{\sigma(i)} \mid i \in [k]\}$ forms a clique of size $k$ in $G$.

Remember that exactly one couple moves from $H^{i,j}$ for each $i,j \in [k]$, which (considering also the size of $F$) forces exactly two singles to leave $H'^{i,j}$ for each $i,j \in [k]$. Taking into account the couples $c(b_i)$ and the singles $s_1(b'_i), s_2(b'_i)$ for each $i \in [2k-1]$ and the single $s_0$, we get that $M$ is $4k^2 + 4(2k-1) + 1 = (4k^2 + 8k - 3) = \ell$-close to $M_0$.

For the other direction, suppose $v_{\sigma(1)}, v_{\sigma(2)}, \dots, v_{\sigma(k)}$ form a clique in $G$. By defining $M$ as below, it is straightforward to verify that $M$ is an assignment for $(S, C, H, f, A)$ which covers every resident, and is $\ell$-close to $M_0$.

$M(c(b_i)) = (b'_i, b'_i)$ for each $i \in [2k-1]$
$M(c(h^{i,j}_{\sigma(i),\sigma(j)})) = (h'^{i,j}_{\sigma(i),\sigma(j)}, h'^{i,j}_{\sigma(i),\sigma(j)})$ for each $i, j \in [k]$
$M(s_0) = b_1$
$M(s_1(b'_i)) = h^{1,i}_{\sigma(1),\sigma(i)}$ for each $i \in [k]$
$M(s_1(b'_{k+i})) = h^{i,1}_{\sigma(i),\sigma(1)}$ for each $i \in [k-1]$
$M(s_2(b'_{2k-1})) = h^{k,1}_{\sigma(k),\sigma(1)}$
$M(s_2(b'_i)) = b_{i+1}$ for each $i \in [2k-2]$
$M(s_1(h'^{i,j}_{\sigma(i),\sigma(j)})) = h^{i,j+1}_{\sigma(i),\sigma(j+1)}$ for each $i \in [k], j \in [k-1]$
$M(s_2(h'^{i,j}_{\sigma(i),\sigma(j)})) = h^{i+1,j}_{\sigma(i+1),\sigma(j)}$ for each $i \in [k-1], j \in [k]$
$M(s_1(h'^{i,k}_{\sigma(i),\sigma(k)})) = f_i$ for each $i \in [k]$
$M(s_2(h'^{k,i}_{\sigma(k),\sigma(i)})) = f_i$ for each $i \in [k]$
$M(p) = M_0(p)$ for every $p \in S \cup C$ where $M(p)$ was not defined above.

□

# CHAPTER 7

# Conclusions

In this dissertation we considered the parameterized complexity of several graph modification and stable assignment problems. In Chapter 2 we discussed the parameterized $k$-Apex problem, where given a graph $G$ and a parameter $k$, the task is to decide whether there are $k$ vertices in $G$ whose removal yields a planar graph. Using treewidth-based techniques including results by Robertson and Seymour from graph minor theory and a theorem by Courcelle, we presented an FPT algorithm for this problem whose running time is quadratic in the number of vertices of $G$ for every fixed $k$.

In Chapters 3 and 4 we presented various results considering the parameterized complexity of deciding whether two graphs can be made isomorphic by deleting a few vertices from the larger one. To this end, we defined the Cleaning($\mathcal{H}, \mathcal{G}$) problem: given a pair of graphs $(H, G)$ with $H \in \mathcal{H}$ and $G \in \mathcal{G}$, find a set of vertices $S$ in $G$ such that $G - S$ is isomorphic to $H$. We investigated the complexity of this problem with a non-standard parameterization, where the parameter of an input $(H, G)$ is the difference of the number of vertices in $G$ and $H$. This parameterization has not been studied before in the literature for this problem.

In Section 3.1, we focused on the special case where both input graphs are planar and the smaller one is 3-connected. After showing the NP-hardness of this problem, we presented a quadratic time FPT algorithm for it. In Section 3.2, we dealt with the case where the smaller input graph is a tree and the larger one can be arbitrary. This case was already known to be NP-hard, and we settled the parameterized complexity of this problem by providing an FPT algorithm for it, running in cubic time for each fixed value of the parameter.

Chapter 3 contains the study of the Cleaning($\mathcal{H}, \mathcal{G}$) problem for the case where both input graphs are interval graphs. We gave an FPT algorithm for this problem, when parameterized by the non-standard parameterization used also in Chapter 3. We also proved the NP-hardness of this problem.

In Chapters 5 and 6, we turned our attention to stable assignment problems. None of the problems discussed in these chapters have been studied from a parameterized viewpoint before. In Chapter 5, we showed that finding a maximum stable matching in the Stable Marriage with Ties and Incomplete Lists problem is W[1]-hard, when parameterized by the number of ties. In contrast with this, we showed that the problem becomes FPT, when parameterized by the total length of ties.

We also obtained results concerning the applicability of local search for this problem. We studied the problem of deciding for a given instance $I$ of SMTI, a stable matching $M$ for $I$, and an integer $\ell$, whether $I$ admits a stable matching $M'$ larger than $M$ such that the number of persons having different partners in $M$ and in $M'$ is at most $\ell$. We showed that no algorithm for this problem can run in FPT time (unless W[1] = FPT) if we regard $\ell$ as a parameter, and besides, either the number of ties is a parameter as well, or the maximum length of the ties is at most 2. In addition, we investigated two optimization problems, namely the Egalitarian Stable Marriage with Ties and Incomplete Lists and the Minimum Regret Stable Marriage with Ties and Incomplete Lists. On the one hand, we gave an FPT algorithm for them with the parameterization where the parameter is the total length of the ties. On the other hand, we showed strong FPT-inapproximability results for both of these problems concerning the case when the parameter is only the number of ties.

In Chapter 6 we studied the parameterized complexity of the Hospitals/Residents with Couples problem. We proved that finding a stable assignment is W[1]-hard, if the parameter is the number of couples in the instance. We also proved that no permissive local search algorithm for the problem of finding a stable assignment of maximum size can run in FPT time (unless W[1] = FPT), if parameterized by only the radius of the explored neighborhood. By contrast, we described a strict local search algorithm for this problem that runs in FPT time, if both the radius of the neighborhood and the number of couples are regarded as parameters.

Additionally, we also investigated a variant of the Hospitals/Residents with Couples problem, called Maximum Matching with Couples, where no preferences are involved, and the task is to find an acceptable assignment having maximum size. We described a randomized FPT algorithm for this problem with the parameter being the number of couples, by using FPT results from matroid theory. We also showed that no permissive local search algorithm can run in FPT time for this problem (unless W[1] = FPT), if the parameter is the radius of the explored neighborhood.

There are several possible directions for further research. First, considering applicability in practice, it is a relevant question whether the running times of the presented algorithms can be improved. In the case of the three algorithms proposed for the Cleaning problem, we believe that an effective implementation would result in much better running times for practical instances than the proven upper bounds suggest. Thus, an empirical study of these algorithms would be interesting.

Regarding the Cleaning problem, another natural direction for future research is to investigate further graph classes. A possible candidate for such research could be the class of permutation graphs, since the Graph Isomorphism problem is solvable for such graphs [34]. An important open question is whether the Cleaning problem on planar graphs can be solved by an FPT algorithm. We conjecture that the answer is yes. Also, it would be interesting to examine whether unit interval graphs are different from general interval graphs in the context of Induced Subgraph Isomorphism.

Yet another possibility for future investigations is the generalization of the Cleaning problem, where vertices can be deleted from both graphs. The resulting problem is the Maximum Common Induced Subgraph problem, where the task is to delete some vertices from both input graphs in a way that the obtained graphs are isomorphic to each other. Here, both the size of the desired common subgraph and the number of deletions could be relevant parameters. Naturally, allowing also edge deletions raises interesting questions as well.

Considering the area of stable matchings, a possible issue that could be studied in the parameterized framework is whether "almost" stable matchings (of a certain size) are easier to find than stable matchings. We can consider a matching almost stable if it admits only a certain number of blocking pairs. This question has already been studied from the classical complexity perspective [4]. From the parameterized viewpoint, the number of blocking pairs allowed is a natural parameter. Such questions may also be relevant in connection with the HOSPITALS/RESIDENTS problem.

Finally, the most important and appealing research direction for future work is the area of kernelization algorithms (see e.g. [70, 20]). None of the FPT algorithms proposed in the dissertation provide a non-trivial kernel directly, though such kernelizations seem possible in connection with the CLEANING problem. Hence, it is an interesting open question whether polynomial-time kernels exist for the FPT problems discussed here. Using newly developed techniques [21, 58], the non-existence of such kernels might also be proven. Altogether, this area can be a fruitful direction for future research.

# Bibliography

[1] E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization.* Princeton University Press, Princeton, NJ, 2003. Reprint of the 1997 original [Wiley, Chichester]. (Page 10.)

[2] A. Abdulkadiroğlu, P. A. Pathak, and A. E. Roth. The New York City high school match. *American Economic Review*, 95(2):364–367, May 2005. (Pages 2 and 14.)

[3] A. Abdulkadiroğlu, P. A. Pathak, A. E. Roth, and T. Sönmez. The Boston public school match. *American Economic Review*, 95(2):368–371, May 2005. (Pages 2 and 14.)

[4] D. J. Abraham, P. Biró, and D. F. Manlove. "Almost stable" matchings in the roommates problem. In *WAOA 2005: Proceedings of the Third International Workshop on Approximation and Online Algorithms*, volume 3879 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006. (Page 107.)

[5] I. Adler, M. Grohe, and S. Kreutzer. Computing excluded minors. In *SODA 2008: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 641–650, 2008. (Page 18.)

[6] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974. (Page 43.)

[7] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Appl. Math.*, 123(1-3):75–102, 2002. (Page 11.)

[8] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. (Pages 97 and 99.)

[9] S. Arnborg, A. Proskurowski, and D. Seese. Monadic second order logic, tree automata and forbidden minors. In *CSL 1990: Proceedings of the 4th Workshop on Computer Science Logic*, volume 533 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1991. (Page 20.)

[10] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994. (Page 17.)

[11] P. Biró. Student admissions in Hungary as Gale and Shapley envisaged. Technical Report TR-2008-291, University of Glasgow, Department of Computing Science, 2008. (Pages 2, 13, and 14.)

[12] P. Biró and E. J. McDermid. Matching with sizes (or scheduling with processing set restrictions). Technical Report TR-2010-307, University of Glasgow, Department of Computing Science, 2010. (Page 99.)

[13] C. Blair. The lattice structure of the set of stable matchings with multiple partners. *Math. Oper. Res.*, 13(4):619–628, 1988. (Page 14.)

[14] M. Blum. A machine-independent theory of the complexity of recursive functions. *J. ACM*, 14(2):322–336, 1967. (Page 1.)

[15] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial $k$-trees. *J. Algorithms*, 11(4):631–643, 1990. (Pages 32 and 40.)

[16] H. L. Bodlaender. On disjoint cycles. *Int. J. Found. Comput. Sci.*, 5(1):59–68, 1994. (Page 18.)

[17] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. (Pages 20 and 27.)

[18] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *MFCS 1997: Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 1997. (Pages 6 and 19.)

[19] H. L. Bodlaender. A cubic kernel for feedback vertex set. In *STACS 2007: Proccedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science*, volume 4393 of *Lecture Notes in Computer Science*, pages 320–331. Springer, 2007. (Page 41.)

[20] H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In *IWPEC 2009: 4th International Workshop on Parameterized and Exact Computation*, volume 5917 of *Lecture Notes in Computer Science*, pages 17–37, Berlin, 2009. Springer. (Page 107.)

[21] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. (Page 107.)

[22] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13:335–379, 1976. (Page 46.)

[23] A. Borodin. Computational complexity and the existence of complexity gaps. *J. ACM*, 19(1):158–174, 1972. (Page 1.)

[24] L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996. (Pages 17 and 18.)

[25] L. Cai, S. M. Chan, and S. O. Chan. Random separation: A new method for solving fixed-cardinality optimization problems. In *IWPEC 2006: Proceedings of the 2nd International Workshop on Parameterized and Exact Computation*, volume 4169 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2006. (Page 32.)

[26] D. Cantala. Matching markets: the particular case of couples. *Economics Bulletin*, 3(45):1–11, 2004. (Page 16.)

[27] J. Chen, I. A. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In *MFCS 2006: Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science*, volume 4162 of *Lecture Notes in Computer Science*, pages 238–249, Berlin, 2006. Springer. (Page 7.)

[28] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):1–19, 2008. (Page 18.)

[29] Y. Chen and J. Flum. On parameterized path and chordless path problems. In *CCC 2007: 22nd Annual IEEE Conference on Computational Complexity*, pages 250–263. IEEE Computer Society, 2007. (Pages 33 and 41.)

[30] Y. Chen, M. Thurley, and M. Weyer. Understanding the complexity of induced subgraph isomorphisms. In *ICALP 2008: Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part I*, pages 587–596, Berlin, 2008. Springer. (Pages 32 and 33.)

[31] A. Church. A note on the Entscheidungsproblem. *J. Symb. Log.*, 1(1):40–41, 1936. (Page 1.)

[32] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics,*, 58:345–363, 1936. (Page 1.)

[33] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*, pages 24–30, Amsterdam, 1965. North-Holland. (Page 1.)

[34] C. J. Colbourn. On testing isomorphism of permutation graphs. *Networks*, 11:13–21, 1981. (Pages 32 and 106.)

[35] C. J. Colbourn and K. S. Booth. Linear time automorphism algorithms for trees, interval graphs, and planar graphs. *SIAM J. Comput.*, 10(1):203–225, 1981. (Page 46.)

[36] S. A. Cook. The complexity of theorem-proving procedures. In *STOC 1971: Proceedings of the Third annual ACM Symposium on Theory of Computing*, pages 151–158. ACM Press, 1971. (Page 1.)

[37] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, second edition, 2001. (Page 43.)

[38] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. Elsevier and MIT Press, 1990. (Page 19.)

[39] B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 313–400. World Scientific, 1997. (Page 27.)

[40] J. Díaz and D. M. Thilikos. Fast FPT-algorithms for cleaning grids. In *STACS 2006: Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 361–371. Springer, 2006. (Pages 32 and 33.)

[41] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, Heidelberg, 2005. (Pages 4, 5, 6, 19, 20, 22, and 35.)

[42] Y. Dinitz, A. Itai, and M. Rodeh. On an algorithm of Zemlyachenko for subtree isomorphism. *Inf. Process. Lett.*, 70(3):141–146, 1999. (Page 55.)

[43] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87:161–187, 1992. (Page 18.)

[44] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer-Verlag, New York, 1999. (Page 5.)

[45] B. Dutta and J. Massó. Stability of matchings when individuals have preferences over colleagues. *J. Econom. Theory*, 75(2):464–475, 1997. (Page 16.)

[46] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Springer-Verlag, second edition, 1994. (Page 27.)

[47] F. Echenique and J. Oviedo. A theory of stability in many-to-many matching markets. *Theoretical Economics*, 1(2):233–273, June 2006. (Page 14.)

[48] J. Edmonds. Maximum matching and a polyhedron with $0, 1$-vertices. *J. Res. Nat. Bur. Standards*, 69B:125–130, 1965. (Page 1.)

[49] J. Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:449–467, 1965. (Page 1.)

[50] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3):1–27, 1999. (Pages 17, 32, and 33.)

[51] M. Farber. Domination, independent domination, and duality in strongly chordal graphs. *Discrete Appl. Math.*, 7(2):115–130, 1984. (Page 45.)

[52] M. R. Fellows, F. V. Fomin, D. Lokshtanov, F. A. Rosamond, S. Saurabh, and Y. Villanger. Local Search: Is brute-force avoidable? In *IJCAI 2009: Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 486–491, 2009. (Page 12.)

[53] M. R. Fellows and M. A. Langston. On search, decision, and the efficiency of polynomial-time algorithms. *J. Comput. Syst. Sci.*, 49(3):769–779, 1994. (Page 18.)

[54] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *J. of Global Optimization*, 6:109–133, 1995. (Page 11.)

[55] I. S. Filotti and J. N. Mayer. A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus (working paper). In *STOC 1980: Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 236–243. ACM, 1980. (Page 32.)

[56] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002. (Page 27.)

[57] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, New York, 2006. (Page 5.)

[58] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *STOC 2008: Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 133–142. ACM, 2008. (Page 107.)

[59] H. N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *STOC 1983: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 448–456. ACM, 1983. (Page 41.)

[60] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *Amer. Math. Monthly*, 69(1):9–15, 1962. (Pages 2, 9, 12, 13, 78, and 90.)

[61] D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Appl. Math.*, 11(3):223–232, 1985. (Page 96.)

[62] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, 1979. A Series of Books in the Mathematical Sciences. (Pages 1, 4, 31, and 33.)

[63] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM J. Algebraic Discrete Methods*, 4:312–316, 1983. (Page 17.)

[64] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.*, 1:180–187, 1972. (Page 45.)

[65] P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canad. J. Math.*, 16:539–548, 1964. (Page 46.)

[66] C. A. Glass and H. Kellerer. Parallel machine scheduling with job assignment restrictions. *Naval Research Logistics*, 54(3):250–257, 2007. (Page 99.)

[67] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, Norwell, MA, 1997. (Page 11.)

[68] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931. (Page 1.)

[69] M. Grohe. Computing crossing numbers in quadratic time. *J. Comput. Syst. Sci.*, 68(2):285–302, 2004. (Pages 18 and 27.)

[70] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007. (Page 107.)

[71] D. Gusfield. Three fast algorithms for four problems in stable marriage. *SIAM J. Comput.*, 16(1):111–128, 1987. (Pages 85 and 86.)

[72] D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms.* Foundations of Computing Series. MIT Press, Cambridge, MA, 1989. (Pages 9, 13, and 78.)

[73] F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM J. Comput.*, 4(3):221–225, 1975. (Page 17.)

[74] M. Hajiaghayi and N. Nishimura. Subgraph isomorphism, log-bounded fragmentation, and graphs of (locally) bounded treewidth. *J. Comput. Syst. Sci.*, 73(5):755–768, 2007. (Page 32.)

[75] M. M. Halldórsson, R. W. Irving, K. Iwama, D. Manlove, S. Miyazaki, Y. Morita, and S. Scott. Approximability results for stable marriage problems with ties. *Theor. Comput. Sci.*, 306(1-3):431–447, 2003. (Page 85.)

[76] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.*, 117:285–306, 1965. (Page 1.)

[77] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973. (Page 35.)

[78] J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974. (Pages 2, 9, 23, 32, and 43.)

[79] R. W. Irving. An efficient algorithm for the "stable roommates" problem. *J. Algorithms*, 6(4):577–595, 1985. (Page 14.)

[80] R. W. Irving. Stable marriage and indifference. *Discrete Appl. Math.*, 48(3):261–272, 1994. (Page 14.)

[81] R. W. Irving, P. Leather, and D. Gusfield. An efficient algorithm for the "optimal" stable marriage. *J. ACM*, 34(3):532–543, 1987. (Pages 85 and 86.)

[82] R. W. Irving and D. F. Manlove. The stable roommates problem with ties. *J. Algorithms*, 43(1):85–105, 2002. (Page 14.)

[83] R. W. Irving and D. F. Manlove. Approximation algorithms for hard variants of the stable marriage and hospitals/residents problems. *J. Comb. Optim.*, 16(3):279–292, 2008. (Page 14.)

[84] K. Iwama, D. Manlove, S. Miyazaki, and Y. Morita. Stable marriage with incomplete lists and ties. In *Automata, Languages and Programming (Prague, 1999)*, volume 1644 of *Lecture Notes in Computer Science*, pages 443–452. Springer, Berlin, 1999. (Pages 14 and 78.)

[85] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. (Page 1.)

[86] S. Khuller, R. Bhatia, and R. Pless. On local search and placement of meters in networks. *SIAM J. Comput.*, 32(2):470–487, 2003. (Page 11.)

[87] Z. Király. Better and simpler approximation algorithms for the stable marriage problem. In *ESA '08: Proceedings of the 16th annual European symposium on Algorithms*, pages 623–634, Berlin, Heidelberg, 2008. Springer-Verlag. (Page 14.)

[88] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983. (Page 11.)

[89] B. Klaus and F. Klijn. Stable matchings and preferences of couples. *J. Econom. Theory*, 121(1):75–106, 2005. (Page 16.)

[90] A. Krokhin and D. Marx. On the hardness of losing weight. In *ICALP 2008: 35rd International Colloquium on Automata, Languages and Programming*, volume 5125 of *Lecture Notes in Computer Science*, pages 662–673, Berlin, 2008. Springer. (Pages 11 and 12.)

[91] L. A. Levin. Universal sorting problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973. In Russian. (Page 1.)

[92] J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. (Page 17.)

[93] A. Lingas. Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theor. Comput. Sci.*, 63(3):295–302, 1989. (Page 31.)

[94] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980. (Page 17.)

[95] D. Lokshtanov. Wheel-free deletion is W[2]-hard. In *IWPEC 2008: Proceedings of the Third International Workshop on Parameterized and Exact Computation*, volume 5018 of *Lecture Notes in Computer Science*, pages 141–147, Berlin, 2008. Springer. (Page 18.)

[96] G. S. Lueker and K. S. Booth. A linear time algorithm for deciding interval graph isomorphism. *J. ACM*, 26(2):183–195, 1979. (Pages 2, 32, 46, 55, and 69.)

[97] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982. (Page 32.)

[98] D. F. Manlove, R. W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theor. Comput. Sci.*, 276(1-2):261–279, 2002. (Pages 78, 83, 84, 85, and 88.)

[99] D. Marx. Local search. Parameterized Complexity News, pages 7–8, volume 3, 2008. (Page 11.)

[100] D. Marx. A parameterized view on matroid optimization problems. In *ICALP 2006: 33rd International Colloquium on Automata, Languages and Programming*, volume 4051 of *Lecture Notes in Computer Science*, pages 656–667. Springer, Berlin, 2006. (Page 100.)

[101] D. Marx. Chordal deletion is fixed-parameter tractable, 2008. To appear in *Algorithmica*. (Conference version: *WG 2006, LNCS 4271*, 37–48, 2006.). (Page 18.)

[102] D. Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008. (Page 78.)

[103] D. Marx. Searching the $k$-change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.*, 36(1):31–36, 2008. (Page 11.)

[104] D. Marx and I. Schlotter. Obtaining a planar graph by vertex deletion. In *WG 2007: 33nd International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 4769 of *Lecture Notes in Computer Science*, pages 292–303, Berlin, 2007. Springer. (Pages 4 and 19.)

[105] D. Marx and I. Schlotter. Parameterized complexity and local search approaches for the stable marriage problem with ties, 2009. To appear in *Algorithmica.* (Pages 4 and 78.)

[106] D. Marx and I. Schlotter. Parameterized graph cleaning problems. *Discrete Applied Mathematics*, 157(15):3258–3267, 2009. (Pages 4 and 32.)

[107] D. Marx and I. Schlotter. Stable assignment with couples: parameterized complexity and local search. In *IWPEC 2009: 4th International Workshop on Parameterized and Exact Computation*, volume 5917 of *Lecture Notes in Computer Science*, pages 300–311, Berlin, 2009. Springer. (Pages 4 and 91.)

[108] D. Marx and I. Schlotter. Cleaning interval graphs. *CoRR*, abs/1003.1260, 2010. arXiv:1003.1260 [cs.DS]. (Pages 4, 32, and 45.)

[109] D. Marx and I. Schlotter. Parameterized complexity of the arc-preserving subsequence problem, 2010. Manuscript. (Page 45.)

[110] J. Matoušek and R. Thomas. On the complexity of finding iso- and other morphisms for partial $k$-trees. *Discrete Math.*, 108(1-3):343–364, 1992. Topological, algebraical and combinatorial structures. Frolík's memorial volume. (Page 40.)

[111] D. W. Matula. Subtree isomorphism in $o(n^{5/2})$. *Ann. Discrete Math.*, 2:91–106, 1978. (Pages 31, 32, and 33.)

[112] E. J. McDermid and D. F. Manlove. Keeping partners together: Algorithmic results for the Hospitals / Residents problem with couples. *J. Comb. Optim.*, 19(3):279–303, 2010. (Page 16.)

[113] G. L. Miller. Isomorphism testing for graphs of bounded genus. In *STOC 1980: Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 225–235. ACM, 1980. (Page 32.)

[114] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications.* Oxford University Press, Oxford, 2006. (Page 5.)

[115] L. Perkovic and B. A. Reed. An improved algorithm for finding tree decompositions of small width. *Int. J. Found. Comput. Sci.*, 11(3):365–371, 2000. (Pages 20 and 27.)

[116] E. L. Post. Finite combinatory processes – formulation 1. *J. Symb. Log.*, 1(3):103–105, 1936. (Page 1.)

[117] B. A. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. (Page 18.)

[118] P. A. Robards. *Applying Two-Sided Matching Processes to the United States Navy Enlisted Assignment Process.* PhD thesis, Naval Postgraduate School Monterey CA, 2001. (Pages 2 and 13.)

[119] N. Robertson and P. D. Seymour. Graph minors. V. Excluding a planar graph. *J. Comb. Theory, Ser. B*, 41(1):92–114, 1986. (Page 20.)

[120] N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Combin. Theory Ser. B*, 63(1):65–110, 1995. (Pages 2 and 18.)

[121] N. Robertson and P. D. Seymour. Graph minors. XX. Wagner's conjecture. *J. Combin. Theory Ser. B*, 92(2):325–357, 2004. (Pages 2 and 18.)

[122] N. Robertson, P. D. Seymour, and R. Thomas. Quickly excluding a planar graph. *J. Comb. Theory, Ser. B*, 62(2):323–348, 1994. (Pages 18, 20, and 27.)

[123] E. Ronn. NP-complete stable matching problems. *J. Algorithms*, 11(2):285–304, 1990. (Pages 15 and 90.)

[124] A. E. Roth. The evolution of the labor market for medical interns and residents: A case study in game theory. *Journal of Political Economy*, 92:991–1016, 1984. (Pages 2, 13, 15, and 90.)

[125] A. E. Roth. A natural experiment in the organization of entry-level labor markets: Regional markets for new physicians and surgeons in the United Kingdom. *American Economic Review*, 81(3):415–440, June 1991. (Page 14.)

[126] A. E. Roth and E. Peranson. The redesign of the matching market for American physicians: Some engineering aspects of economic design. *American Economic Review*, 89(4):748–780, September 1999. (Page 16.)

[127] A. E. Roth and M. Sotomayor. *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*, volume 18 of *Econometric Society Monographs*. Cambridge University Press, Cambridge, MA, 1990. (Pages 2, 13, and 16.)

[128] W. J. Savitch. Relationship between nondeterministic and deterministic tape classes. *J. Comput. Syst. Sci.*, 4:177–192, 1970. (Page 1.)

[129] S. Szeider. The parameterized complexity of $k$-flip local search for SAT and MAX SAT. In *SAT 2009: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, volume 5584 of *Lecture Notes in Computer Science*, pages 276–283. Springer, 2009. (Page 11.)

[130] C. Thomassen. The graph genus problem is NP-complete. *J. Algorithms*, 10(4):568–576, 1989. (Page 17.)

[131] B. A. Trakhtenbrot. Turing computations with logarithmic delay. *Algebra i Logika*, 3(4):33–48, 1964. (Page 1.)

[132] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc. (2)*, 42:230–265, 1937. (Page 1.)

[133] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. A correction. *Proc. London Math. Soc. (2)*, 43:544–546, 1937. (Page 1.)

[134] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985. (Page 11.)

[135] V. N. Zemlyachenko. Canonical numbering of trees. In *Proc. Seminar on Comb. Anal. at Moscow State University*, 1970. (In Russian). (Page 55.)

[136] V. N. Zemlyachenko. Determining tree isomorphism. In *Voprosy Kibernetiki, Proc. of the Seminar on Combinatorial Mathematics, Moscow, 1971*, pages 54–60. Akad. Nauk SSSR, Scientific Council on the Complex Problem "Cybernetics", 1973. (In Russian). (Page 55.)