

Select, Allocate, and Manipulate via Multivariate Analysis

By

Sanjukta Roy

MATH10201504009

The Institute of Mathematical Sciences, Chennai

A thesis submitted to the

Board of Studies in Mathematical Sciences

In partial fulfillment of requirements

for the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE



March, 2020

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at Homi Bhabha National Institute (HBNI) and is deposited in the Library to be made available to borrowers under rules of the HBNI.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the Competent Authority of HBNI when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

Sanjukta Roy

DECLARATION

I hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and has not been submitted earlier as a whole or in part for a degree / diploma at this or any other Institution / University.

Sanjukta Roy

LIST OF PUBLICATIONS ARISING FROM THE THESIS

Publications in Refereed Journal:

- [1] D. Adil, S. Gupta, S. Roy, S. Saurabh, and M. Zehavi. “Parameterized algorithms for stable matching with ties and incomplete lists”. In: *Theoretical computer science* 723 (2018), pp. 1–10.
- [2] S. Gupta and S. Roy. “Stable matching games: manipulation via subgraph isomorphism”. In: *Algorithmica* 80.9 (2018), pp. 2551–2573.
- [3] S. Gupta, S. Roy, S. Saurabh, and M. Zehavi. “Parameterized algorithms and kernels for rainbow matching”. In: *Algorithmica* 81.4 (2019), pp. 1684–1698.
- [4] S. Gupta, S. Roy, S. Saurabh, and M. Zehavi. “Quadratic vertex kernel for rainbow matching”. In: *Algorithmica* (2020).

Conference Publications:

- [1] S. Gupta, S. Roy, P. Jain, S. Saurabh, and M. Zehavi. “Gehrlein stability in committee selection: parameterized hardness and algorithms”. In: *Proceedings of the 18th international conference on autonomous agents and multiagent systems, AAMAS 2019, may 13-17, 2019, montreal, canada*. 2019, pp. 511–519.
- [2] S. Gupta, S. Roy, S. Saurabh, and M. Zehavi. “Balanced stable marriage: how close is close enough?”. In: *Algorithms and data structures - 16th international symposium, WADS 2019, edmonton, ab, canada, august 5-7, 2019, proceedings*. 2019, pp. 423–437.
- [3] S. Gupta, S. Roy, S. Saurabh, and M. Zehavi. “Group activity selection on graphs: parameterized analysis”. In: *Algorithmic game theory - 10th international symposium, SAGT 2017, l’aquila, italy, september 12-14, 2017, proceedings*. 2017, pp. 106–118.
- [4] S. Gupta, S. Roy, S. Saurabh, and M. Zehavi. “When rigging a tournament, let greediness blind you”. In: *Proceedings of the twenty-seventh international joint conference on artificial intelligence, IJCAI 2018, july 13-19, 2018, stockholm, sweden*. 2018, pp. 275–281.
- [5] S. Gupta, S. Roy, S. Saurabh, and M. Zehavi. “Winning a tournament by any means necessary”. In: *Proceedings of the twenty-seventh international joint*

conference on artificial intelligence, IJCAI 2018, july 13-19, 2018, stockholm, sweden. 2018, pp. 282–288.

Sanjukta Roy

DEDICATIONS

To my parents.

ACKNOWLEDGEMENTS

I would like to acknowledge the immense support, help, and guidance I have received from my advisor Prof. Saket Saurabh. I am grateful to him for his patience and constant imposition to achieve higher goals, throughout the metamorphosis of my Ph.D. work right from infancy. I have learned many things from him both academically and non-academically. Without his guidance, this Ph.D. thesis would not have been accomplished. I take this opportunity to extend my humble gratitude to my co-authors Prof. Sushmita Gupta, Prof. Meirav Zehavi, and Dr. Pallavi Jain for the countless discussions we have had. I would like to thank Prof. Sushmita Gupta for the diligence and rigor of correcting my writing for the umpteenth time.

I shall be failing in my duties if I do not acknowledge my doctoral committee members Prof. Venkatesh Raman, Prof. Meena Mahajan, Prof. R. Ramanujam, and Prof. G. Philip for their valuable feedback and attention during the review meetings. If I can mention an enduring framework I have always come to rely upon, it has to be the experience gained during my course work and TA work in IMSc. I have had the pleasure of learning various subjects from my advisor Prof. Saket Saurabh, Prof. Venkatesh Raman, Prof. Meena Mahajan, Prof. V. Arvind, and Prof. R. Ramanujam. The concepts infused has helped me during my Ph.D. and will be an endearing companion in my further ventures.

I would like to thank my seniors Pranabendu Mishra, Fahad Panolan, Akanksha Agrawal, and Sudeshna Koley who have helped me since the days of my graduate course work. In no small parts, my friends and colleagues PrafullKumar Tale, Roohani Sharma, Jayakrishnan M, Abhishek Sahu, and Lawqueen Kanesh have helped me to paint a complete picture in several aspects of my academic endeavor.

The entire ordeal would have been devoid of colors and spirit if it was not for my friends at IMSc, Lawqueen Kanesh, Oorna Mitra, Shivani Singh, and all the members of the Parameterized Complexity group. I would also like to thank my friend Niladri Ranjan Das for providing moral support and encouragement.

Last but not the least, my family: my parents, brother, and sister have encouraged me to pursue my dreams and have always provided me with immense support spiritually throughout my Ph.D. and life in general. If I were left awry of these, the journey would never have been complete.

Contents

Summary	i
List of Figures	v
1 Introduction	1
1.1 Organization of the thesis	5
2 Preliminaries	7
2.1 Parameterized Complexity	9
A: STABLE ALLOCATION PROBLEMS	12
3 Stable Matching with Ties and Incomplete Lists	15
3.1 Preliminaries	21
3.2 Algorithms for SMTI	22
3.2.1 Kernel for MAX-SMTI	22
3.2.2 Kernel for MIN-SMTI	26
3.3 Kernel for SRTI	27

3.4	SMTI and SRTI on planar graphs	28
3.5	Conclusions	37
4	Balanced Stable Matching	39
4.1	Preliminaries	48
4.1.1	Known Results	50
4.2	Hardness	52
4.2.1	Formal Description of the Reduction	54
4.2.2	The Parameter	55
4.2.3	Correctness	58
4.3	Kernel	63
4.3.1	Functional Balanced Stable Marriage	63
4.3.2	Balanced Stable Marriage	85
4.4	Parameterized Algorithm	88
4.4.1	Bounded Search Tree: An Overview	88
4.4.2	Description of the Algorithm	89
4.4.3	The Procedure Branch	90
4.4.4	Algorithm	94
4.5	Conclusion	95
5	Group Activity Selection Problem on Graphs(gGASP)	97
5.1	Preliminaries	104

5.1.1	NP-completeness of STEINER TREE* on graphs of maximum degree 4.	105
5.2	Hardness	106
5.2.1	NP-completeness of gNSGA	106
5.3	An FPT Algorithm for General Graphs	111
5.4	FPT Algorithm for Networks of Bounded Treewidth	118
5.4.1	Computation	121
5.4.2	Correctness	123
5.5	Conclusion	132
B:	SELECTION PROBLEMS	132
6	Rainbow Matching	135
6.1	Our Contribution	136
6.1.1	Overview	140
6.1.2	Related Work	141
6.1.3	Preliminaries	143
6.2	Algorithm for RAINBOW MATCHING on Paths	144
6.3	FPT Algorithm for RAINBOW MATCHING on General Graphs	150
6.4	Kernelization Algorithms	152
6.4.1	Kernelization on general graphs: Algorithm I	153
6.4.2	Kernelization on graphs of bounded degree	155

6.5	Improved Kernelization Algorithms	157
6.5.1	Decomposing the graph into a vertex cover and an independent set	158
6.5.2	Kernelization on Forests	161
6.5.3	Kernelization on General Graphs: Algorithm II	166
6.6	Conclusion, Discussion and Open Problems	172
7	Stable Committee Selection	175
7.1	Preliminaries	180
7.2	Structural Observations	181
7.3	Hardness	182
7.4	Exact Algorithms for GEHRLEIN STABLE COMMITTEE SELECTION	186
7.4.1	A polynomial time subcase	187
7.4.2	Exact exponential time algorithm	189
7.5	FPT Algorithms for GSCS	191
7.6	A linear vertex kernel for GSCS	196
7.7	Conclusion	201
C:	MANIPULATION PROBLEMS	201
8	Stable Extension of Partial Matching	205
8.1	Our problem and motivation	207
8.1.1	Our Contributions	210

8.2 Preliminaries	212
8.3 Generalization of Suitor Graph	213
8.4 Exact Algorithm for SEOPM	218
8.4.1 Universality of Universal Suitor Graph	220
8.4.2 Rooted Universal Suitor Graph and Valid Subgraphs	222
8.4.3 $2^{\mathcal{O}(n)}$ Algorithm for SEOPM	224
8.4.4 A Lower Bound under Exponential Time Hypothesis	228
8.5 Stable extension when lists may not be complete	229
8.5.1 Overview of Algorithm 8.5.1	232
8.6 Concluding thoughts	237
9 Tournament Fixing Problem	241
9.1 Our Contribution	246
9.2 Preliminaries	248
9.3 Greedy Algorithm for TFP	250
9.3.1 Phase I: Guessing	253
9.3.2 Phase II: Verification of Guesses	254
9.3.3 Phase III: Greedy Choice for Path Resolution	257
9.3.4 Phase IV: Greedy Choices for Subtree Resolution	266
9.4 Obfuscation Operations	272
9.5 Combinatorial Result	275

9.6 Characterization of YES-Instances	278
9.7 Exact Algorithms	281
9.8 Conclusion	284
Bibliography	284

Summary

This thesis studies parameterized complexity of some NP-hard optimization problems related to STABLE MATCHING, RAINBOW MATCHING, and a set of problems that comes under the umbrella name of Computational Social Choice (COMSOC). All these problems have been extensively studied from the perspective of classical complexity and are well known hard problems. We, mainly, explore various combinatorial structures for the problems. We identify structural parameters that are, in many cases, much smaller than the size of input of the problem, and develop efficient combinatorial algorithms when the parameters are small. Towards this, we delve deep into the problem and identify which algorithmic tools can be exploited for the problem.

Specifically, we study two optimization variants of STABLE MATCHING, they are (i) finding a maximum/ minimum size stable matching when the preference lists are incomplete and not strict (i.e., it contains ties); (ii) finding a stable matching that is “balanced” between the men optimal stable matching and the women optimal stable matchings. The balance of a matching μ is defined as, $balance(\mu) = \max\{\sum_{(m,w) \in \mu} p_m(w), \sum_{(m,w) \in \mu} p_w(m)\}$, where $p_x(y)$ denote the position of y in x ’s preference list. The goal of the problem is to find a stable matching μ that minimizes $balance(\mu)$. For both of these problems, we use marking process and pre-processing rules to produce an “equivalent” reduced instance of size polynomial in the parameter in time polynomial in the input size. For the former problem, we

develop a non-trivial (better than brute force algorithm on the reduced instance) “fixed parameter tractable” (FPT) algorithm. We develop an algorithm that runs in time exponential in treewidth. This result combined with the fact that planar graphs on n vertices have treewidth at most $\sqrt{n}$, gives a better algorithm when the acceptability graph is a planar graph. For the latter problem, we study two “above guarantee parameters”. We give kernel and use the branching technique to give an FPT algorithm with respect to one parameter. We show hardness with respect to another smaller parameter.

The third problem we study is the GROUP ACTIVITY SELECTION (GASP) problem on graphs (gGASP). Unlike STABLE MATCHING, here agents have preferences over not only other agents but also groups of agents that include themselves. Here we identify some natural parameters with respect to which the problem remains hard. Then we identify structural parameters of the input graph with respect to which the problem is FPT.

We study another classical problem in matching theory, the Rainbow Matching problem. Given an edge colored graph, the goal is to find a matching with edges of distinct colors and of size at least k , for some non-negative integer k . We reduce this problem to SET PRE-PACKING to give a randomized algorithm based on algebraic technique, running in time $\mathcal{O}(2^k)n^{\mathcal{O}(1)}$. We give a branching algorithm for path graphs which runs in time $\mathcal{O}(\phi^k)n^{\mathcal{O}(1)}$, where ϕ is the golden ratio. We develop a kernel of size k^3 , then further improve the size to k^2 using “expansion lemma”.

A central question in COMSOC is winner determination in voting. Here, we have a set of candidates and a set of votes which are total order over the candidates. We study it in the multi-winner setting. The goal is to find a subset of candidates that forms a Gehrlein-stable committee. We initiate a systematic study of finding a weakly Gehrlein-stable committee of size k , for some non-negative integer k in the realm of Parameterized Complexity. This opens the question of studying other

well-known notions of stable committees in the realm of Parameterized Complexity.

Another well-explored branch in COMSOC is the study of manipulation problems that arise due to the presence of strategic agents. We study manipulation in STABLE MATCHING in bipartite graphs. Here, we are given a partial matching and the preference lists (strict) of one side, the goal is to find preference lists for the other side such that the partial matching is contained in the output of the Gale-Shapley algorithm. To solve this problem we define “Universal Suitor Graphs”, a polynomial-size graph that encodes all possible matchings allowed under the manipulation scheme. We believe such a structure can be used to systematically search stable matchings in other problems. Finally, we apply the black-box algorithm for graph isomorphism when the target graph has bounded treewidth. This produces an algorithm running in time $2^{\mathcal{O}(n)}n^{\mathcal{O}(1)}$ where n is the number of vertices in the bipartite graph. Two other manipulation that we study are TOURNAMENT FIXING and TOURNAMENT BRIBERY problems. We study the combinatorial structures of the problem to give FPT algorithms.

List of Figures

3.1	After deleting Isolated vertices, mark edges and vertices. Red denotes marked vertices and edges. The left partition of the independent set I refers to the top $2k + 1$ preferred vertices that are marked for every vertex in X , the vertex cover. The right partition contains the remaining vertices.	25
3.2	Delete all unmarked edges	26
3.3	Apply Reduction Rule 1 again and now delete all the isolated vertices. This remaining graph is the <i>kernel</i> .	26
5.1	The construction of the underlying graph G .	108
5.2	Depiction of how a nice function f assigns $\{1, 2\}$ to the vertices of G . Orange colored parts are assigned 1 by f and the white enclosed parts are assigned 2 by f . $C_1, C_2, \dots, C_5$ are the components of $G[f^{-1}(1)]$. For $i \in [5]$, the concentric circle outside C_i is $N(C_i)$. It is guaranteed that f assigns 2 to those vertices.	112
7.1	Example: The blue vertices in the set S is a weakly Gehrlein stable committee of size 5 for the voting profile given in Table 7.1.	178

7.2	Construction of D . Here, $n = V(G) $, the green and blue vertices are the node vertices and edge vertices respectively, the vertices in the green set is the set of indicator vertices, and the orange dashed lines show the directed cycles of length k^2 in D .	184
7.3	An illustration of Algorithm 7.5.1 where vertices in the red sets are in the solution	192
8.1	(a) Suitor Graph, (b) Rooted Suitor Graph	214
8.2	(a) Universal suitor graph on vertices $M = \{A, B, C\} \uplus_{X \in \{D, \dots, I\}} \{X^i \mid 4 \leq i \leq 9\}$ and $W = \{1, \dots, 9\}$, (b) Rooted universal suitor graph [described later in Section 8.4.2] for the partial matching $\mu = \{(A, 1), (B, 2), (C, 3)\}$ with source $\{4, 9\}$. Blue vertices are copies of the unmatched male vertices for each unmatched female vertex. The outgoing edges from blue vertices are not shown. Black edges represent the matching edges in μ , red edges represent the preferences of men matched in μ , while the blue edges represent edges from an unmatched woman to her own copies of the unmatched men. The green ellipse represents the set of source vertices, $\{4, 9\}$, that are connected from the root r .	217
9.1	Partition of $V(D) \setminus V(K)$ into types. The arcs in K and the vertices in $V(K)$ are colored red. For all v_i, v_j with $i < j$ such that $(v_j, v_i) \notin K$, we suppose that $(v_i, v_j) \in A(D)$ (not displayed).	245
9.2	Binomial arborescences of sizes $2^0, 2^1, 2^2$ and 2^3 .	248
9.3	The topology T^Δ of $\{v^*, v_4, v_{10}, v_{12}, v_{15}, v_{17}\}$ in B (left), and the template T^* (right). LCA-vertices are colored blue.	251

9.4	An application of the subtree clean-up operation. In D , displayed arcs are backward arcs; all other arcs are forward arcs. The sets R^{rev} and R'^{rev} are the sets of blue arcs in T and T' , respectively. In T' , the operation is not applicable.	273
9.5	An application of the cyclic shift clean-up operation with D being the graph in Fig. 9.1 (add vertices to ensure its size is a power of 2, whose display is irrelevant here). Only parts of the trees T and T' are displayed. Reversed arcs are colored blue. Here, $P = v_6 \rightarrow v_7 \rightarrow v_8$, $p_0 = v_5$, $p_1 = v_6$, $p_2 = v_7$ and $p_3 = v_8$.	274
9.6	A 2-removed binomial arborescence on 2^4 vertices.	281

Chapter 1

Introduction

A matching in a graph is a set of edges without common endpoints. Finding a matching is one of the important ingredients that lead to defining the complexity class P. Computational problems that involve matching agents to one another come with various criteria. In many cases, the agents form two disjoint sets, and the goal is to allocate the agents in one set to those in the other. Examples include assigning applicants to colleges/jobs, resident doctors to hospitals, kidney patients to donors, etc. In this thesis, we focus on the case where agents have ordinal preferences over a subset of the others. That is, each agent has a first choice, second choice and so on. For example, an applicant applying for admission to university would not rank all available universities, but only a small subset of them. Similarly, the universities would form a ranking of their applicants. The preference lists might not be strictly ordered: for example, an applicant might prefer two universities equally. Typically, the universities would not like it if some students leave after admission, which would lead to vacant seats. So, while matching the students to universities we would like to have some conditions to prevent such scenarios. Classically, this is enforced by a “stability” condition. If there is a student university pair such that they are not matched to each other but prefer each other compared to their current matching,

then the pair is said to form a *blocking pair*. A matching is stable if it does not have a blocking pair. David Gale and Lloyd Shapley gave an algorithm for finding stable matchings between two groups of people with preferences over each other, in a seminal paper in 1962.

The problems in matching theory can be viewed as a part of a larger domain which is social choice theory. Social choice theorists from a range of different disciplines, including mathematics, economics, and political science are interested in the design and theoretical evaluation of voting rules. Assessing the computational difficulty of determining a winner of a voting process, or manipulating it, are two prominent examples of the importation of a concept from the field of social choice to the field of theoretical computer science. This interdisciplinary view on collective decision making defines computational social choice as a field. The computational aspects of social choice theory is a relatively new area. Finding matching between two sets of agents can be thought of as a precursor of problems that arise in computational social choice. In this thesis, the problems of this paradigm are discussed as a set of stable allocation problems. As mentioned before, the preferences of the agents might not be strict, that is, there can be ties. We study the computational complexity of finding a stable matching in the presence of ties. In some applications, there are other constraints in addition to stability. The algorithm given by Gale and Shapley finds a particular stable matching that is preferred by the agents in one side but disliked by the other side. Hence, another question that is studied is finding a stable matching that makes all the agents equally “happy”. Another type of allocation problem is where one wants to assign a group of agents to a set of activities. Here, every agent has preference not only over the activities but also over the group size. We study the following question: can we find an assignment of groups of agents to activities such that every agent in a group know at least one other agent in the group. This is modelled by a graph where the vertices are the agents and there is an edge between two agents if they know each other. The “friendship” property is inherent

in a connected set. Moreover, we wish to capture the property that when people are part of a club or play sports together it is likely that they form a “connected network” together. So we want each group to be connected. This guarantees that each member directly knows someone in the network. We also want the assignment to be individually rational and Nash stable.

Another variety of problems that we study in this thesis is finding a subset of agents that satisfy some combinatorial properties from a given set of agents. We categorize these problems as Selection problems. We study, apparently, two very different problems here. The first is, given a graph with colored edges, we want to find a k size subset of edges with distinct colors. The second Selection problem we study is more in the flavour of Computational Social choice but it can be formulated as a graph problem as well. Here, we are given a set of candidates and a set of voters who provide a linear ordering of the candidates. We want to find a subset of candidates such that each candidate in the subset is preferred by a (weak) majority of voters over each candidate who is not in the subset. In Social Choice, the problem of selecting a subset of candidates is known as the COMMITTEE SELECTION problem or MULTI-WINNER VOTING. We explore different computational aspects of this problem.

As mentioned previously, manipulation problems are studied in Computational Social Choice for voting rules and in various other contexts. It is arguably the second most studied question in COMSOC after winner determination. We study two different types of manipulation problems. This thesis started with the study of stable matching. In this part of the thesis again we go back to the STABLE MATCHING problem in a bipartite graph. We study how to manipulate the outcome of the Gale-Shapley algorithm. Particularly, we want a given partial matching to be part of the output of the Gale-Shapley algorithm.

The theory of social choice has developed an immense variety of voting rules,

ranging from simple rules such as plurality to more complicated rules. Many voting rules can be viewed as tournaments: competitions between the candidates that determine the winner using some rule based solely on the results of *matches*, that is, pairwise comparisons. Besides being used to implement voting rules, tournaments are also prevalent in many social settings. Few common areas include sports competitions, patent races [38, 110], hiring employees [143], and even drug trials (these are commonly referred to as “head-to-head” drug trials). The outcome of the matches in a round dictates (using some rule) which matches take place in the next round. In the final round, a winner is determined. The structure of the tournament indicates how matches in different rounds are conducted. This, in turn, decides how the final winner is determined. We study the balanced knockout tournament. In a knockout tournament, a player is out of the competition if s/he loses a game. A balanced knockout tournament is governed by an unordered balanced binary tree where the players are assigned to the leaves of the tree. In the first stage, every two players mapped to leaves with the same parent compete against each other, and the winner is mapped to the common parent, and the next round is conducted similarly. We study two types of manipulation problems in these tournaments. One is the TOURNAMENT FIXING problem where the fixing is done by a central authority. The other involves bribing individual players to fix a particular match in the competition. These are, basically, two flavours of manipulation problems that we see in COMSOC. The first type of problem is called control problems in the literature of Voting theory and the later is called bribery problem.

Although finding a matching is solvable in polynomial time, i.e., is in the complexity class **P**. The problems described above are computationally hard, that is they are **NP**-complete. The field of theoretical computer science deals with such problems in mainly two ways. Either try to find an approximate solution efficiently or try to find an exact solution in time that is polynomial in input size but depends exponentially on some other parameter. Studying a problem in the later realm

analyses the Parameterized Complexity of the problem as compared to the Classical Complexity. Over the last two decades, Parameterized Complexity has evolved to be a central field of research in theoretical computer science. However, the scope of this field had a strong focus on applications to NP-hard optimization problems *on graphs*. There is no inherent reason why this should be the case. Indeed, the main idea of Parameterized Complexity is very general—measure running time in terms of both input size and a parameter that captures the structural properties of the input instance. The idea of a multivariate analysis of algorithms holds the potential to address the need for a framework for refined algorithm analysis for all kinds of problems across all domains and subfields of computer science. Recently, techniques in Parameterized Complexity were successfully applied in the area of Computational Social Choice Theory. In particular, Voting has become a subject of intensive study from the viewpoint of Parameterized Complexity, for a few examples, see [149, Chapter 10, 11] and [12]; for more information on the current state-of-the-art, we refer to excellent surveys such as [13, 21, 20, 44]. However, Voting is only one topic under the rich umbrella of Computational Social Choice. Parameterized analysis of other topics has been few and far between; a few examples of notable recent developments concern rigging a winner of a tournament [135]. In the recent past, a collective effort to study matching under preferences through the lens of Parameterized Complexity was initiated [106, 23, 126, 69, 80, 77, 68, 124, 107, 116, 117, 17].

1.1 Organization of the thesis

The thesis is divided into three parts. In the first part, we study the stable allocation problems. In this part, we study two variants of the STABLE MATCHING problem and the GROUP ACTIVITY SELECTION problem on graphs. In the second part of the thesis, we study a set of problems where the objective is to select a subset of given elements that satisfy a given constraint. We call the problems in the second

part as selection problems. Here, we study RAINBOW MATCHING and COMMITTEE SELECTION problem. Finally, in the third part, we study manipulation problems. Manipulation of stable matching and tournaments are studied in the final part of the thesis.

Chapter 2

Preliminaries

Sets. The union of two disjoint sets X and Y is denoted by $X \uplus Y$. For sets X and Y , $X \setminus Y$ denote a set of elements of X that are not in Y ; $X \triangle Y$ denotes the symmetric difference between X and Y .

Function. We denote the set $\{1, 2, \dots, n\}$ by $[n]$. Let $f : A \rightarrow B$ be some function. Given $A' \subseteq A$, the notation $f(A') = b$ indicates that for all $a \in A'$, it holds that $f(a) = b$. An *extension* f' of the function f is a function whose domain A' is a superset of A and whose range is B , such that for all $a \in A$, it holds that $f'(a) = f(a)$. For any $A' \subset A$, the *restriction* $f|_{A'}$ of f is a function from A' to B such that for any $a \in A'$, $f|_{A'}(a) = f(a)$.

Graph Theory. Given a graph $G = (V, E)$, a *matching* is a subset of E without two (or more) edges with a common vertex. For pairwise-disjoint sets $S, S' \subseteq V$, let $E(S, S')$ denote the set of edges with one endpoint in S and the other endpoint in S' . Given a set $S \subseteq V$, let $E(S)$ denote the set of edges with both endpoints in S . Given a set of edges $Z \subseteq E$, let $V(Z)$ denote the set of endpoints of the edges in Z . Given a vertex $v \in V$, denote $N_G(v) = \{u \in V : \{u, v\} \in E\}$. If G is clear from context, we omit the subscript G . Given a set $S \subseteq V$, denote $N_S(v) = N(v) \cap S$. Furthermore, we denote $\deg_S(v) = |N_S(v)|$. Given a matching μ in G , let $\mu(v)$ denote the vertex

matched to v in μ . For $X \subseteq V(G)$, $G - X$ and $G[X]$ denote subgraphs of G induced on the vertex set $V(G) \setminus X$ and X , respectively. An edge $uv \in E(G)$ is present in $E(G[X])$ if and only if $u, v \in X$. For an edge subset $Z \subseteq E(G)$, by $G[Z]$ we mean the subgraph of G with vertex set $V(Z)$ and edge set Z . The graph $G[Z]$ is also called the graph induced on Z . For $v_1, v_t \in V(G)$, a directed path from v_1 to v_t is denoted by $P = (v_1, v_2, \dots, v_t)$, where $V(P) \subseteq V(G)$ and for each $i \in [t - 1]$, $(v_i, v_{i+1}) \in E(G)$. Given two undirected graph, H and G , we say that H is a *minor* of G , if H can be formed from G by executing a sequence of operations, where each operation either deletes a vertex/edge or *contracts* an edge. Here, a contraction of an edge $e = (u, v)$ is the operation that removes e from the graph and unifies the vertices u and v . A planar graph is a graph that can be embedded in the plane. The vertices are represented by points in the plane, its edges are represented by lines between these points, and it can be drawn on the plane in such a way that its edges intersect only at their endpoints. In other words, it can be drawn in such a way that no edges cross each other. The notations K_5 and $K_{3,3}$ refer to a complete graph on five vertices and complete bipartite graph with three vertices in each bipartition, respectively. By the well-know Wagner's theorem [152], *planar graph* is a graph that excludes K_5 and $K_{3,3}$ as minors. We point the reader to [35] for the definitions of a minor.

Let G be a directed graph. We denote an arc from u to v by an ordered pair (u, v) , and say that u is an in-neighbor of v and v is an out-neighbor of u . For $x \in V(G)$, $N_G^-(x) = \{y \in V(G) : (y, x) \in \mathcal{A}(G)\}$ and $N_G^+(x) = \{y \in V(G) : (x, y) \in \mathcal{A}(G)\}$. In a directed graph G , we say a vertex u is reachable from a vertex v , if there is directed path from v to u . A graph is called a *strongly connected component* if every vertex in the graph is reachable from every other vertex. Let $X \subseteq V(G)$. A strongly connected component, $G[X]$, is called maximal if there does not exist a vertex $v \in V(G) \setminus X$ such that $G[X \cup \{v\}]$ is also a strongly connected component. Let $x \in V(G)$. We define two sets $R_G^-(x)$ and $R_G^+(x)$ as follows. $R_G^-(x) = \{x\} \cup \{y \in V(G) : x \text{ is reachable from } y\}$

and $R_G^+(x) = \{x\} \cup \{y \in V(G) : y \text{ is reachable from } x\}$. We call $R_G^-(x)$ and $R_G^+(x)$ as *in-reachability* set and *out-reachability* set of x in G , respectively. For $S \subseteq V(G)$, $R_G^-(S) = \cup_{v \in S} R_G^-(v)$ and $R_G^+(S) = \cup_{v \in S} R_G^+(v)$. The subscript in the notation for the neighborhood and the reachability sets may be omitted if the graph under consideration is clear from the context. A *topological ordering* of a directed graph G is an ordering, denoted by τ , of the vertices of $V(G)$ such that for every arc $(u, v) \in \mathcal{A}(G)$, we have $\tau(u) < \tau(v)$. Given an undirected graph G , complement of G is a graph G' such that $V(G') = V(G)$ and $E(G') = \{uv : u, v \in V(G) \text{ and } uv \notin E(G)\}$.

2.1 Parameterized Complexity

A *parameterization* of a problem is the association of an integer k with each input instance, which results in a *parameterized problem*. For our purposes, we need to recall three central notions that define the parameterized complexity of a parameterized problem. The first one is the notion of a *kernel*. Here, an instance $(\mathcal{I}, k)$ of some parameterized problem Π is said to admit a *kernel* of size $f(k)$ for some function f that depends *only* on k if there exists a polynomial-time algorithm, called a *kernelization algorithm*, that translates the input $\mathcal{I}$ instance into an “equivalent”¹ instance $\mathcal{J}$ of the same problem, $|\mathcal{J}|$ is bounded by $f(k)$ and such that the value of the parameter does not increase. In case the function f is polynomial in k , the problem is said to admit a *polynomial kernel*. Hence, kernelization is a mathematical concept that aims to analyze the power of preprocessing procedures in a formal, rigorous manner.

The second notion that we use is the one of *fixed-parameter tractability* (FPT). Here, an instance $(\mathcal{I}, k)$ of some parameterized problem Π is said to be FPT if there is an algorithm that solves it in time $f(k) \cdot |\mathcal{I}|^{O(1)}$, where $|\mathcal{I}|$ is the size of

¹Two instances $\mathcal{I}$ and $\mathcal{J}$ are said to be equivalent if $\mathcal{I}$ is a YES-instance if and only if $\mathcal{J}$ is a YES-instance.

the input and f is a function that depends only on k . Such an algorithm is called a *parameterized algorithm*. In other words, the notion of FPT signifies that it is not necessary for the combinatorial explosion in the running time of an algorithm for Π to depend on the input size, but it can be confined to the parameter k . It is known that if a parameterized problem is FPT then it admits a kernelization algorithm. This implies that a decidable problem admits a kernel if and only if it is fixed-parameter tractable. Thus, kernelization can be another way of defining fixed-parameter tractability. A problem Π is called *above-guarantee (below-guarantee)* parameterization if given an instance $(\mathcal{I}, k)$ of the problem Π with a guarantee that a solution of size $g(\mathcal{I})$ exists, we want to decide whether $\mathcal{I}$ admits a solution of size at least (resp. at most) $k + g(\mathcal{I})$. Here, $g(\mathcal{I})$ is usually a lower bound (resp. upper bound) on the maximum (resp. minimum) size of a solution.

Finally, we recall that Parameterized Complexity also provides tools to refute the existence of polynomial kernels and parameterized algorithms for certain problems (under plausible complexity-theoretic assumptions), in which context the notion of **W[1]-hardness** is a central one. It is widely believed that a problem that is **W[1]-hard** is unlikely to be FPT, and we refer the reader to the books [29, 37] for more information on this notion in particular, and on Parameterized Complexity in general. The notation $\mathcal{O}^*$ is used to hide factors polynomial in the input size.

While designing our kernelization algorithm, we might be able to determine whether the input instance is a YES-instance or a NO-instance. For the sake of clarity, in the first case, we simply return YES, and in second case, we simply return NO. To properly comply with the definition of a kernel, the return of YES and NO should be interpreted as the return of a trivial YES-instance and a trivial NO-instance, respectively.

Reduction Rules. To design our kernelization algorithm, we rely on the notion of a *reduction rule*. A reduction rule is a polynomial-time procedure that replaces

an instance $(\mathcal{I}, k)$ of a parameterized problem Π by a new instance $(\mathcal{I}', k')$ of Π . The rule is said to be *safe* if $(\mathcal{I}, k)$ is a YES-instance if and only if $(\mathcal{I}', k')$ is a YES-instance. Given a list of reduction rules, we always apply the first reduction rule whose condition is true. Note that the same rule may be applied multiple times consecutively. Moreover, it is possible that after we exhaustively apply some rule, the condition of a rule that precedes it on the list would become true.

Kernelization Algorithm. A kernelization algorithm consecutively applies various reduction rules in order to shrink the instance size as much as possible. Thus, such an algorithm takes as input an instance $(\mathcal{I}, k)$ of Π , works in polynomial time, and returns an equivalent instance $(\mathcal{I}', k')$ of Π such that the output size is finite and bounded by a computable function of the parameter (see [29, 37]).

Treewidth. Treewidth is a structural parameter indicating how much a graph resembles a tree. Formally,

Definition 2.1.1. A tree decomposition of a graph G is a pair (T, β) of a tree T and $\beta : V(T) \rightarrow 2^{V(G)}$, such that

1. $\bigcup_{v \in V(T)} \beta(v) = V(G)$, and
2. for any edge $\{\rho, \rho'\} \in E(G)$ there exists a node $v \in V(T)$ such that $\rho, \rho' \in \beta(v)$,
and
3. for any vertex $\rho \in V(G)$, the subgraph of T induced by the set $T_\rho = \{v \in V(T) : \rho \in \beta(v)\}$ is a tree.

The width of (T, β) is $\max_{v \in V(T)} |\beta(v)| - 1$. The treewidth of G is the minimum width of a tree decomposition of G .

For $v \in V(T)$, we say that $\beta(v)$ is the *bag* of v , and $\gamma(v)$ denotes the union of the bags of v and the descendants of v in T . We let $\text{tw}(G)$ denote the treewidth of G .

We also define a form of a tree decomposition that simplifies the design of DP algorithms.

Definition 2.1.2 (Nice Tree Decomposition). *A tree decomposition (T, β) of a graph G is nice if for the root r of T , $\beta(r) = \emptyset$, and each node $v \in V(T)$ is of one of the following types.*

- **Leaf:** v is a leaf in T and $\beta(v) = \emptyset$.
- **Forget:** v has one child, u , and there is a vertex $x \in \beta(u)$ such that $\beta(v) = \beta(u) \setminus \{x\}$.
- **Introduce:** v has one child, u , and there is a vertex $x \in \beta(v)$ such that $\beta(v) \setminus \{x\} = \beta(u)$.
- **Join:** v has two children, u and w , and $\beta(v) = \beta(u) = \beta(w)$.

In polynomial time, we transform (T, β) into a *nice* tree decomposition of the same width [18]; the number of nodes in this decomposition is bounded by $\mathcal{O}(n)$. According to standard practice in Parameterized Complexity with respect to problems parameterized by **tw**, we assume that every input instance is given to us along with a tree decomposition (of the input graph G) of width $\mathcal{O}(\mathbf{tw})$ [2].

²Otherwise, such a decomposition can be computed in time $2^{\mathcal{O}(\mathbf{tw})} \cdot n$ using the means described in [19]. In fact, in FPT time, we can also obtain a tree decomposition of width exactly **tw** [29].

Stable Allocation Problems

Chapter 3

Stable Matching with Ties and Incomplete Lists

STABLE MATCHING (SM) is a classic problem in economics, computer science and mathematics. Several books have been dedicated to SM [71]. Moreover, the Nobel Prize in Economics was awarded to Shapley and Roth in 2012 “for the theory of stable allocations.” Algorithms for SM are routinely employed to handle a wide-variety of real-world situations such as the assignment of users to servers in a large distributed Internet service. In the presence of both incomplete lists and ties, the problem is called SM WITH TIES AND INCOMPLETE LISTS (SMTI). Informally, the input of SMTI consists of a set of people, each person ranking a *subset* of people of the opposite sex in a *non-strict* manner. The goal is to find a matching that is *stable* in the following sense: there does not exist a pair of a man and a woman that strictly prefer being matched to each other over their current “status”. Here, every person prefers being matched (to a ranked person) over being unmatched. Every instance of SMTI has a stable matching, and such a matching can be found in polynomial time [85]. However, there can be an *exponential* number of such matchings [71]. Depending on the application at hand, some of these matchings would be better than

others. The two (arguably) most natural objectives are to maximize or minimize the size of the matching as it might be desirable to maintain stability while either maximizing or minimizing the use of available “resources”. These objectives define the well-known NP-hard MAX-SMTI and MIN-SMTI problems [85].

In this chapter, we study MAX-SMTI and MIN-SMTI in the realm of parameterized complexity. Here, each problem instance is associated with a parameter k . In the context of MAX-SMTI and MIN-SMTI, we choose the size of the solution (namely, the number of matched pairs in a solution) as the parameter. In the absence of ties or when the preference lists are complete, the Rural Hospital Theorem [59] guarantees that the same set of people is matched by *every* stable matching. These guarantees do not hold in the presence of both ties and incomplete lists, thereby leading to the possibility that even the largest stable matching is *arbitrarily smaller than the size of the instance*. In this context, note that our parameterizations are extremely relevant to practical applications that (even occasionally) realize this possibility. Furthermore, parameterization by solution size is the classical, most common one in the field. Let us also remark that very often, the establishment of fixed-parameter tractability with respect to solution size is not only interesting on its own right, but it is essential to the study of other parameterizations as well (such as above-guarantee parameterizations).

Motivating Example. Let us motivate our study in the context of the well studied example of matching applicants to jobs, where an applicant is assigned to a job. In particular, we would thus argue that even when the objective is to maximize the size of the stable matching, solution size is a realistic parameter. First, it is generally not practical that applicants (or firms hiring applicants for jobs) should have to submit a ranking of all qualifying jobs (resp. applicants). Instead, a truncated list containing their top choices is desirable. Second, it is also likely that the agents (applicants or jobs) would be *tied* on some of their admitted choices, signifying that the choices

are equally good. Assuming that the ordering on tied groups is transitive, we have the practical realization of the theoretical model behind SMTI. Additionally, any given job is assigned to one applicant, and any applicant can only accept at most one position, but the total number of available jobs is likely to be significantly smaller than the number of applicants who are applying. Therefore, the maximum size of a matching can be significantly smaller than the size of the instance. To utilize the available resources, it is imperative for the matching agency (an employment bureau, headhunter, recruitment firm, etc) to assign as many available jobs to as many qualifying applicants. These considerations lead us to believe that the size of the largest stable matching is a realistic parameter for further analysis.

Formulation. We formulate SMTI via *bipartite graphs*: given a bipartite graph $G = (A, B, E)$, the men (women) are represented by the vertices in A (resp. B). Moreover, we have a family of preference lists, $\mathcal{L}^A$: for each vertex $a \in A$, $\mathcal{L}^A(a) \in \mathcal{L}^A$ is a (not necessarily strict) ranking over a subset of B . Symmetrically, we have a family $\mathcal{L}^B$. Note that we assume w.l.o.g. that $b \in \mathcal{L}^A(a)$ if and only if $a \in \mathcal{L}^B(b)$. Let $\mathcal{I} = (A, B, \mathcal{L}^A, \mathcal{L}^B)$ is an instance of SMTI.

Definition 3.0.1. *Given a matching μ of G , a blocking edge is an edge $\{a, b\} \in E$ where each vertex (a or b) is either unmatched by μ or strictly prefers the other vertex to its matched partner in μ . If there do not exist blocking edges, then μ is said to be (weakly) stable.*^[1]

In this setting, MAX-SMTI and MIN-SMTI are defined as follows.

¹In the presence of *ties*, there are two other notions of stability: super stability and strong stability (see [84, 115]).

MAX(RES. MIN)-SMTI

Input: A bipartite graph $G = (A \cup B, E)$, and two families of preference lists, $\mathcal{L}^A$ and $\mathcal{L}^B$ and a non-negative integer k .

Parameter: k

Task: Find (if there exists) a weakly stable matching of size at least k (resp. at most k).

In the broader setting of STABLE ROOMMATE WITH TIES AND INCOMPLETE LISTS (SRTI), we are given an *arbitrary* graph $G = (V, E)$, and a family of preference lists $\mathcal{L}$. For each vertex $v \in V$, $\mathcal{L}(v)$ is a (not necessarily strict) ranking over the set of the neighbors of v in G , denoted by $N(v)$. Clearly, the notion of stability is well defined also in this setting.

MAX-SRTI

Input: A graph $G = (V, E)$, the family of preference lists $\mathcal{L}$, the size of a maximum matching ℓ , and a positive integer k .

Parameter: ℓ

Task: Find (if there exists) a weakly stable matching of size at least k .

The Parameter ℓ . For the STABLE ROOMMATE problem even with complete lists and without ties, a stable matching may not always exist [83]. Given an instance of SRTI, merely testing whether there exists a stable matching is NP-hard [137]². Thus, there does not exist (unless P=NP) any algorithm for MAX-SRTI which runs in time of the form $f(k) \cdot |V|^{\mathcal{O}(1)}$ (or even $|V|^{f(k)}$) where f depends only on k . Indeed, by setting $k = 1$, we could employ such an algorithm to test the *existence* of a stable

²By breaking the ties of an instance of SRTI arbitrarily, we have that if a matching is stable in the new instance, then it is stable in the original instance. However, it is computationally hard to decide how to break the ties in order to test the existence of a stable matching.

matching in polynomial time. Hence, we introduce an alternative parameterization. To this end, we note that a stable matching, if one exists, is a *maximal matching* in the graph. Furthermore, the size of any maximal matching is at least half the value of ℓ . Thus, if a stable matching exists, then its size and the value of ℓ differ by a factor of at most 2. Clearly, this leads us directly to the parameterization of MAX-SRTI by ℓ .

Our Contribution. We employ the method of polynomial-time preprocessing, known as kernelization. Here, a problem admits a *kernel* of size $f(k)$ if there exists a *polynomial-time* algorithm, called a *kernelization algorithm*, translating any input instance into an “equivalent instance” of the same problem whose size is bounded by $f(k)$. By devising a nontrivial marking scheme, we show that all the above problems admit a polynomial kernel. For example, in the context of MAX-SMTI, given an instance $(\mathcal{I}, k)$ of MAX-SMTI, we output (in polynomial time) another instance $(\mathcal{I}', k)$ of MAX-SMTI where $|\mathcal{I}'| = \mathcal{O}(k^2)$ and $(\mathcal{I}, k)$ is a YES-instance of MAX-SMTI if and only if $(\mathcal{I}', k)$ is a YES-instance of MAX-SMTI.

Our kernels also result in the design of FPT algorithms: First obtain an equivalent instance by applying the kernelization algorithm, where the output graph $G' = (A' \cup B', E')$ has $\mathcal{O}(k^2)$ edges. If we solve MAX-SMTI, then we enumerate all subsets of edges of size q in G' , where $k \leq q \leq 2k$. Since $\sum_{q=k}^{2k} \binom{|E'|}{q} \leq \sum_{q=k}^{2k} \left(\frac{e|E'|}{q}\right)^q = |E'|^{\mathcal{O}(k)}$, the running time is $2^{\mathcal{O}(k \log k)}$. If we solve MIN-SMTI, then we enumerate all subsets of edges of size at most k in G' ; again, the running time is $\sum_{q=1}^k \binom{|E'|}{q} = 2^{\mathcal{O}(k \log k)}$. In both cases, for each subset of edges, we test whether it is a stable matching in polynomial time. Overall, we show that:

Theorem 1. MAX-SMTI admits a kernel of size $\mathcal{O}(k^2)$, and an algorithm with running time $2^{\mathcal{O}(k \log k)} + n^{\mathcal{O}(1)}$.

Theorem 2. MIN-SMTI admits a kernel of size $\mathcal{O}(k^2)$, and an algorithm with running time $2^{\mathcal{O}(k \log k)} + n^{\mathcal{O}(1)}$.

For MAX-SRTI, we derive the following theorem.

Theorem 3. MAX-SRTI admits a kernel of size $\mathcal{O}(\ell^2)$, and an algorithm with running time $2^{\mathcal{O}(\ell \log \ell)} + n^{\mathcal{O}(1)}$.

Finally, we restrict the input to *planar graphs*, which are extensively studied in real-life applications. Intuitively speaking, a representation of 3D objects on a 2D surface, such as construction plans and traffic maps, are planar. Furthermore, Peters [131] has recently explicitly asked to study graphical hedonic games (which subsume matching problems such as SM and STABLE ROOMMATE) on bipartite, planar and H -minor free graph topologies. Specifically, we obtain the following theorems.

Theorem 4. MAX-SMTI (MIN-SMTI) on planar graphs admits a kernel of size $\mathcal{O}(k)$ and an algorithm running in time $2^{\mathcal{O}(\sqrt{k} \log k)} + n^{\mathcal{O}(1)}$.

Theorem 5. MAX-SRTI on planar graphs admits a kernel of size $\mathcal{O}(\ell)$ and an algorithm running in time $2^{\mathcal{O}(\sqrt{\ell} \log \ell)} + n^{\mathcal{O}(1)}$.

To prove Theorems [4] and [5], we also obtain the following theorem, which might be of independent interest.

Theorem 6. MAX-SMTI, MIN-SMTI and MAX-SRTI, parameterized by the treewidth $\mathbf{tw}$ of the input graph, admit algorithms running in times $n^{\mathcal{O}(\mathbf{tw})}$.

Other Related Works. MAX-SMTI is NP-hard even if inputs are restricted to having ties only in the preference lists of men, preference lists of bounded length, and symmetry in preference lists [85]. Thus, it is natural to study MAX-SMTI from the perspectives of approximation algorithms and parameterized complexity. The best known approximation algorithm for MAX-SMTI is a 1.5-approximation algorithm [121]. Marx and Schlotter [116] studied MAX-SMTI with many different

parameters, such as the the maximum number of ties in an instance, the maximum length of ties in an instance, and the total length of the ties in an instance. Unfortunately, they showed that MAX-SMTI is unlikely to be FPT with respect to the first and third of these parameters, which further motivates our study of solution size. We remark that other results, whose survey is beyond the scope of our study, are known for the special cases of MAX-SMTI as well as hardness of approximation.

Experimental approaches have also been employed to provide algorithms for MAX-SMTI. Munera et.al [127] gave an algorithm based on local search. Gent and Prosser [65] formulated the problem as a constrained optimization problem and gave an algorithm via constrained programming for both decision and optimization version. Finally, by [76], it is known that MIN MAXIMAL MATCHING is NP-hard on planar cubic graphs. Hence, the reduction in [87] directly implies that MAX-SMTI, MIN-SMTI and MAX-SRTI are NP-hard on planar graphs.

3.1 Preliminaries

Preferences. Let $G = (V, E)$ be a graph where vertices are associated with preference lists. Given vertices $v, v', x \in V$, the notation $v >_x v'$ indicates that x prefers v over v' . Moreover, the notation $v \geq_x v'$ indicates that either $v >_x v'$ or v and v' are tied in x 's preference list. Given a vertex $v \in V$, we view the preference list of v as a strict ordering of ties. For example, a strict preference list is an ordering of ties such that each tie is of length 1 (length of a tie is the number of vertices in the tie). In other words, the ordering on ties is transitive. This ordering also defines the order in which we process the preference list of any vertex, where the internal order in which we process the vertices in each tie is arbitrary.

Operations Removing Edges/Vertices. Given a graph $G = (V, E)$ where vertices are associated with preference lists, the operations that delete edges and vertices

from G are defined as follows. To delete an edge $\{u, v\} = e \in E$, remove the edge e from G , remove the vertex u from v 's preference list, and remove the vertex v from u 's preference list. The order in which u and v rank their remaining neighbors is not altered. Now, to delete a vertex $v \in V$, first delete every edge incident to v (in an arbitrary order), and then remove v itself from G .

3.2 Algorithms for SMTI

In this section we design polynomial kernels for MAX-SMTI and MIN-SMTI.

3.2.1 Kernel for MAX-SMTI

We start by giving a kernel for MAX-SMTI with $\mathcal{O}(k^2)$ vertices and $\mathcal{O}(k^2)$ edges, where k is the solution size.

Decomposition Lemma: Given an instance $\mathcal{I}$ of MAX-SMTI, we arbitrarily break all ties. Then, we compute a stable matching μ in the new instance, $\mathcal{I}'$, in polynomial time by invoking the algorithm in [58]. Note that μ is a stable matching in $\mathcal{I}$. Indeed, if a pair $\{x, y\}$ is a blocking edge for μ in $\mathcal{I}$, then it is also a blocking edge for μ in $\mathcal{I}'$. Overall, we conclude that if $|\mu| \geq k$, then it is safe to output a trivial YES-instance. From now on, we assume that $|\mu| < k$.

For any maximal matching, the set of vertices saturated by the matching edges forms a *vertex cover* of the graph. Moreover, μ is a maximal matching, otherwise an edge outside of μ which is not adjacent to an edge in μ is a blocking edge. Let X denote the set of vertices saturated by the stable matching μ . Thus, X is a vertex cover for the input graph, and it holds that $|X| < 2k$. Furthermore, as the size of a maximum matching in a graph is at most twice the size of any maximal matching, $k \leq |X|$. Hence, if $|X| < k$, then we can output a trivial NO-instance, and conclude

our argument. From now on, we assume that $k \leq |X| < 2k$.

Overall, we partition $V = A \cup B$ into two sets: the vertex cover X and the independent set $I = V \setminus X$. The maximum number of edges inside X is $\mathcal{O}(k^2)$. However, to obtain a kernel for MAX-SMTI, we also need to bound the size of I and the number of edges in $E(X, I)$. We summarize our argument so far in the following “decomposition lemma”.

Lemma 3.2.1. *Given an instance of MAX-SMTI, in polynomial time we can either conclude that the instance is a NO-instance or a YES-instance, or decompose the vertex-set V into a vertex cover X and an independent set $I = V \setminus X$ such that $k \leq |X| < 2k$.*

Isolated Vertices: To bound $|I|$ and $|E(X, I)|$, we first present the following simple reduction rule.

Reduction Rule 3.2.1. *Delete an isolated vertex from G .*

Since isolated vertices do not participate in *any* matching or a blocking edge, this rule is safe. An exhaustive application of this rule takes $\mathcal{O}(n)$ time, where n is the number of vertices. We are now ready to present our base case.

Base Case: For any $x \in X$, consider $E(\{x\}, I)$, the subset of edges whose one endpoint is x and the other endpoint is a vertex in I . Our base case assumes that for every $x \in X$, it holds that $|E(\{x\}, I)| \leq 2k + 1$. Then, since there are no isolated vertices (due to Reduction Rule 3.2.1), it holds that $|I| \leq 2k(2k + 1) = \mathcal{O}(k^2)$. Furthermore, $|E(X, I)| < 2k(2k + 1) = \mathcal{O}(k^2)$. Thus, the total number of vertices is $|X| + |I| \leq 2k + |X|(2k + 1) = \mathcal{O}(k^2)$, and the total number of edges is $|E(X)| + |E(X, I)| \leq |X|^2 + |X|(2k + 1) = \mathcal{O}(k^2)$. Overall, the size of the instance is bounded by $\mathcal{O}(k^2)$. Hence, in our base case, it is safe to output $(\mathcal{I}, k)$. From now on, we assume that there exists at least one vertex $x \in X$ such that $|E(\{x\}, I)| > 2k + 1$.

Marking Edges: We proceed by *marking* some of the edges incident to the vertices in X . For each $x \in X$, consider the edges in $E(\{x\}, I)$ that are currently unmarked. If there are less than $2k + 1$ such edges, then mark all of them. Otherwise, mark the edges incident to the $2k + 1$ most preferred neighbors of x in I (where we break ties arbitrarily)³. The edges with both endpoints in X are considered marked. We say that a vertex in V that is incident to at least one marked edge is a *marked vertex*. Note that if a vertex in X has an unmarked neighbor, then it must have $2k + 1$ marked neighbors in I . Moreover, if a vertex in I is unmarked, then all the edges incident to it are unmarked. Refer to Fig. 1. Let us now present a reduction rule that utilizes our marking scheme.

Reduction Rule 3.2.2. *If there exists $x \in X$ with more than $2k + 1$ neighbors in I , then delete all the unmarked edges in $E(\{x\}, I)$. (Note that no edge in $E(\{x\}, X \setminus \{x\})$ is deleted.)*

Lemma 3.2.2. *Reduction Rule 3.2.2 is safe.*

Proof. Given an instance $(\mathcal{I}, k)$ of MAX-SMTI, let $(\mathcal{I}', k)$ denote the instance outputted by Reduction Rule 3.2.2 on $\mathcal{I}$.

($\Leftarrow$) Let μ' denote a stable matching in $\mathcal{I}'$ such that $|\mu'| \geq k$. For the sake of contradiction, let us assume that there is a blocking edge for μ' in $\mathcal{I}$. Clearly this must be an unmarked edge incident to x , since all other edges are present in $\mathcal{I}'$. We denote this edge by $\{x, y\}$ where $y \in I$. Since x has an unmarked neighbor, it must also have $2k + 1$ marked neighbors. Recall that the size of a maximum matching in $\mathcal{I}$ (and in $\mathcal{I}'$), is at most $|X|$. Hence, $|\mu'| \leq |X|$. Therefore, there exists at least one marked neighbor of x who is unmatched in μ' , denoted by y' . By our marking scheme, we know that $y' \succeq_x y >_x \mu'(x)$, so the marked edge $\{x, y'\}$ is a blocking edge for μ' ; a contradiction. Thus, ($\Leftarrow$) is proved.

³We remark that it is sufficient to mark $|X| + 1 \leq 2k + 1$ edges, but we mark $2k + 1$ for the sake of clarity of presentation.

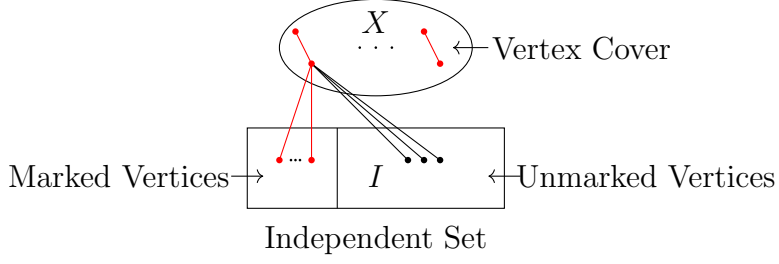


Figure 3.1: After deleting Isolated vertices, mark edges and vertices. Red denotes marked vertices and edges. The left partition of the independent set I refers to the top $2k + 1$ preferred vertices that are marked for every vertex in X , the vertex cover. The right partition contains the remaining vertices.

($\Rightarrow$) Let μ denote a stable matching in $\mathcal{I}$ such that $|\mu| \geq k$. If every edge in μ is also present in $\mathcal{I}'$, then μ is a matching in $\mathcal{I}'$. Since the graph G' in $\mathcal{I}'$ is a subgraph of the graph G in $\mathcal{I}$, we have that μ (if present in $\mathcal{I}'$) must also be stable in $\mathcal{I}'$. For the sake of contradiction, we assume that at least one of the edges in μ does not exist in $\mathcal{I}'$. All missing edges from $\mathcal{I}'$ are unmarked edges incident to x . Therefore, μ contains an unmarked edge incident to x , denoted by $\{x, y\}$, for some $y \in I$. Since $|\mu| \leq |X| \leq 2k$, there exists at least one marked neighbor x , denoted by y' , which is unmatched in μ . As before, we know that $y' \geq_x y$, where $y = \mu(x)$. Without loss of generality, we may assume that y' is the most preferred marked neighbor of x that is unmatched in μ , *i.e.* a more preferred neighbor is matched to some other vertex in μ . Since y' is a marked neighbor of x , we have that $\{y', x\}$ is present in $\mathcal{I}'$. It is easy to see that $\mu' = \mu \setminus \{\{x, y\}\} \cup \{\{x, y'\}\}$ is a stable matching in $\mathcal{I}'$. Thus, ($\Rightarrow$) is proved. $\diamond$

Each application of Reduction Rule [3.2.2](#) takes $\mathcal{O}(n)$ time (since we scan the neighborhood of a vertex in X). The rule can be applied at most $|X|$ times, hence the total time is $\mathcal{O}(kn)$. After applying this rule exhaustively, we obtain an instance $(\mathcal{I}, k)$ where every vertex in X has at most $2k + 1$ neighbors in I , all of which are marked. Thus, Theorem [1](#) is proved.

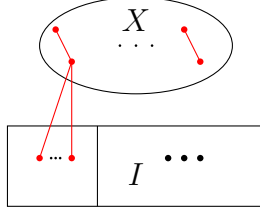


Figure 3.2: Delete all unmarked edges

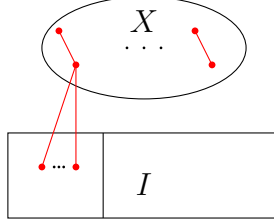


Figure 3.3: Apply Reduction Rule 1 again and now delete all the isolated vertices. This remaining graph is the *kernel*.

3.2.2 Kernel for MIN-SMTI

Here, we give a kernel for MIN-SMTI with $\mathcal{O}(k^2)$ vertices and $\mathcal{O}(k^2)$ edges. As in Section 3.2, we first obtain a (weakly) stable matching μ in $\mathcal{I}$. However, now we output a trivial YES-instance if $|\mu| \leq k$ rather than $|\mu| \geq k$.

Suppose that $|\mu| > k$, and μ is a maximal matching (otherwise it is not stable). Recall that the size of a maximum matching in G is at most twice the size of any maximal matching. Thus, if $\mathcal{I}$ is a YES-instance—that is, there exists a stable matching (which is maximal) of size at most k —then the size of any maximal matching in $\mathcal{I}$ cannot exceed $2k$. Hence, if $|\mu| > 2k$, then we output a trivial NO-instance and conclude our argument. From now on, we assume that $k < |\mu| \leq 2k$. As in Section 3.2, we obtain a decomposition lemma, using which we may assume that we can decompose the vertex set V into a vertex cover X and an independent set $I = V \setminus X$ such that $2k \leq |X| < 4k$. From here on, the construction of the kernel for MIN-SMTI is quite similar to the one presented for MAX-SMTI; we will discuss the main differences, but skip the details.

First, note that Reduction Rule [3.2.1](#) is applicable here, and it is evidently safe. Then, in our Base Case, we assume that for every $x \in X$, it holds that $|E(\{x\}, I)| \leq k + 1$. In this case, we conclude that the total number of vertices is bounded by $4k(k + 2) = \mathcal{O}(k^2)$, and the total number of edges is bounded by $20k^2 + 4k = \mathcal{O}(k^2)$. Next we apply the following reduction rule, the safeness of which follows from arguments almost identical to those in the proof of Lemma [3.2.2](#)—essentially, we replace occurrences of $2k$ by occurrences of k .

Reduction Rule 3.2.3. *Let $x \in X$. Mark the most preferred $k + 1$ neighbors of x in I (if they exist), or else mark as many as possible. Delete (if any) all the unmarked neighbors of x .*

After applying Reduction Rule [3.2.3](#) exhaustively, we obtain an instance handled by our Base Case. So, Theorem [2](#) is proved.

3.3 Kernel for SRTI

The analysis of MAX-SRTI differs from that of MAX-SMTI since a stable matching may not exist in an instance of the former.^{[4](#)} Here, we give a polynomial kernel for MAX-SRTI where the parameter ℓ is the size of a maximum matching of G . Our kernel has $\mathcal{O}(\ell^2)$ vertices and $\mathcal{O}(\ell^2)$ edges.

If $k > \ell$, then we can output a trivial NO-instance, because ℓ is the size of the maximum matching in the graph. Thus, we next focus on the case where $k \leq \ell$. Let μ denote any maximal matching of G . Let X denote a vertex cover obtained by taking the vertices matched by μ (hence, $|X| \leq 2|\mu| \leq 2\ell$), and let $I = V \setminus X$ be an independent set.

We begin our construction by applying Reduction Rule [3.2.1](#), the safeness of

⁴In this context, recall that unless $P=NP$, we cannot obtain an FPT algorithm for MAX-SRTI when the parameter is solution size.

which is self-evident. Next, we mark edges as follows. For $x \in X$, if $E(\{x\}, I)$ has less than $\ell + 1$ edges, then mark all of them. Otherwise, mark the edges incident on the $\ell + 1$ most preferred neighbors of x in I , breaking ties arbitrarily. The vertices in I that are incident to at least one marked edge are called *marked vertices*, and the others are called *unmarked vertices*.

Reduction Rule 3.3.1. *If $x \in X$ has unmarked neighbors in I , then delete all the unmarked edges in $E(\{x\}, I)$.*

Lemma 3.3.1. *Reduction Rule 3.3.1 is safe.*

Proof. Note that $k \leq \ell \leq |X|$, and obviously any matching in the instance obtained by applying Reduction Rule 3.3.1 has size at most ℓ . Thus, the proof of this lemma follows from the arguments presented for Lemma 3.2.2. $\diamond$

Each application of Reduction Rules 3.2.1 and 3.3.1 takes $\mathcal{O}(n)$ time, and each of these rules can be applied at most n times. Thus, the total running time is bounded by $\mathcal{O}(n^2)$, yielding a kernel where every vertex in X has at most $\ell + 1$ neighbors in the I , all of which are marked. Hence, the kernel has $|X| + |I| \leq 2\ell + (\ell + 1)2\ell = \mathcal{O}(\ell^2)$ vertices, and $|E(X)| + |E(X, I)| \leq |X|^2 + |X|(\ell + 1) = \mathcal{O}(\ell^2)$ edges. Therefore, we output the instance $(\mathcal{I}', k)$ and conclude our argument. With this we have proven Theorem 3.

3.4 SMTI and SRTI on planar graphs

Finally, when the input is restricted to planar graphs, we improve the kernels in Sections 3.2 and 3.3, as well as design faster FPT algorithms for MAX-SMTI, MIN-SMTI and MAX-SRTI. As the constructions are very similar to each other, we focus only on MAX-SMTI. Given an input instance $(\mathcal{I}, k)$ of MAX-SMTI, we first apply Lemma 3.2.1. Thus, we have a decomposition of V into a vertex cover X and an

independent set $I = V \setminus X$ such that $k \leq |X| < 2k$. We design a kernel such that both vertex and edge sets are of size $\mathcal{O}(k)$.

Towards our goal, partition $I = I_{\deg=1} \uplus I_{\deg=2} \uplus I_{\deg \geq 3}$. For $1 \leq k \leq 3$, define $I_{\deg=k} = \{v \in I \mid \deg_X(v) = k\}$ and $I_{\deg \geq k} = \{v \in I \mid \deg_X(v) \geq k\}$. Furthermore, define

$$Y_{\leq 1} = \{N(v) \mid v \in I \text{ and } \deg_X(v) \leq 1\}$$

$$Y_{=2} = \{N(v) \mid v \in I \text{ and } \deg_X(v) = 2\}$$

$$Y_{\geq 3} = \{N(v) \mid v \in I \text{ and } \deg_X(v) \geq 3\}$$

Since I is an independent set, all the neighbors of the vertices in I are in X , and thus for any $v \in I$, we have that $\deg_V(v) = \deg_X(v)$. To upper bound $|I_{\deg=1}|$, $|I_{\deg=2}|$ and $|I_{\deg \geq 3}|$, we employ the following result.

Lemma 3.4.1 ([89]). *For a planar graph G with vertex cover X , $|\{N(v) \mid v \in I\}| \leq 6|X| + 1$. In particular, $|Y_{\leq 1}| \leq |X| + 1$, $|Y_{=2}| \leq 3|X|$ and $|Y_{\geq 3}| \leq 2|X|$.*

We first give a rule to enable us to bound $|I_{\deg=1}|$.

Reduction Rule 3.4.1. *Let $x \in X$, and let u denote x 's most preferred neighbor (breaking ties arbitrarily) in $N_I(x) \cap I_{\deg=1}$. Then, delete all vertices in $N_I(x) \cap I_{\deg=1}$ except u .*

Lemma 3.4.2. *Reduction Rule 3.4.1 is safe.*

Proof. Let μ denote a stable matching in $\mathcal{I}$ of size at least k . If $\mu(x) \notin I_{\deg=1}$, then μ is a stable matching in $\mathcal{I}'$, since $\mathcal{I}'$ is a subgraph of $\mathcal{I}$. Whenever $\mu(x) = u$, then also μ is a stable matching in $\mathcal{I}'$. We are left with the case where $\mu(x) \neq u$ and $\mu(x) \in I_{\deg=1}$. In this case, u is unmatched in μ . If $u >_x \mu(x)$, then $\{u, x\}$ blocks μ in $\mathcal{I}$, a contradiction. Therefore, u and $\mu(x)$ must be in a tie in x 's preference list.

But then, the matching $\mu' = \mu \setminus \{\{x, \mu(x)\}\} \cup \{\{x, u\}\}$ is a matching in $\mathcal{I}$ and $\mathcal{I}'$, and is stable in I' . Hence, μ' (of size at least k) is also a stable matching in $\mathcal{I}'$. This proves the ($\Rightarrow$) direction.

Let μ' denote a stable matching in $\mathcal{I}'$ of size at least k . Suppose that there exists a blocking edge for μ' in $\mathcal{I}$. Clearly, it must have been deleted during the application of Reduction Rule [3.4.1](#). Let $\{x, w\}$ denote the blocking edge in $\mathcal{I}$, where $w \in N_I(x) \cap I_{\deg=1} \setminus \{u\}$ such that $w >_x \mu'(x)$. By the definition of u , we know that $u \geq_x w >_x \mu'(x)$. Since the edge $\{x, u\}$ exists in $\mathcal{I}'$, this implies that $\{x, u\}$ blocks μ' in $\mathcal{I}'$; a contradiction. Hence, ($\Leftarrow$) is proved. $\diamond$

The next rule enables us to bound $|I_{\deg=2}|$. We define an equivalence relation $\sim$ on the set $I_{\deg=2}$ as follows: for a pair of vertices $u, v \in I_{\deg=2}$, $u \sim v$ if and only if $N_G(u) = N_G(v)$. Let $\mathcal{E}_i$ denote the i^{th} equivalence class defined by the relation $\sim$. Let the total number of equivalence classes be $\mathcal{N}$.

Reduction Rule 3.4.2. *Let $x \in X$. For each equivalence class $\mathcal{E}_i$ ($1 \leq i \leq \mathcal{N}$) that contains a neighbor of x , mark the two most preferred neighbors (if they exist), breaking ties arbitrarily. Delete all edges between x and its unmarked neighbors in $\mathcal{E}_i$ ($1 \leq i \leq \mathcal{N}$).*

Lemma 3.4.3. *Reduction Rule [3.4.2](#) is safe.*

Proof. Let μ denote a stable matching in $\mathcal{I}'$ of size at least k . Clearly, μ is a matching in $\mathcal{I}$. Suppose that μ is not stable in $\mathcal{I}$, then the blocking edge(s) must be (one of) the deleted edge(s). Let $\{x, u\}$ denote a blocking edge for μ in $\mathcal{I}$, where u is an unmarked neighbor of x such that $u >_x \mu(x)$. The equivalence class containing u , say $\mathcal{E}_i$, must have two marked neighbors (denoted by u_{i_1} and u_{i_2}) that are both preferred by x at least as much as u . Thus, $\mu(x) \notin \{u_{i_1}, u_{i_2}\}$, and since they are in $I_{\deg=2}$, they have exactly two neighbors, one being x . Therefore, at least one of the

marked neighbors, say u_{i_1} , must be unmatched in μ . But then, $\{x, u_{i_1}\}$ is a blocking edge for μ in $\mathcal{I}'$, a contradiction. Hence, $(\Leftarrow)$ direction is proved.

Let μ denote a stable matching in $\mathcal{I}$ of size at least k . If μ is also a matching in $\mathcal{I}'$ then it is stable in $\mathcal{I}'$. Therefore, we assume that μ contains an edge incident on x that was deleted during the application of Reduction Rule [3.4.2](#). Let $\{x, u\}$ denote the edge in μ that does not exist in $\mathcal{I}'$. Let $\mathcal{E}_i$ denote the equivalence class containing u . As argued in the other direction, x has two marked neighbors in $\mathcal{E}_i$ who it prefers at least as much as u . Furthermore, at least one of the marked neighbors must be unmatched in μ , denoted by u_{i_1} . Since μ is stable in $\mathcal{I}$, so u and u_{i_1} must be in a tie in x 's preference list. Thus, we claim that $\mu' = \mu \setminus \{\{x, u\}\} \cup \{\{x, u_{i_1}\}\}$ is a stable matching in $\mathcal{I}$. Since μ' exists in $\mathcal{I}'$, it is also stable in $\mathcal{I}'$. This completes the proof of $(\Rightarrow)$ direction. $\diamond$

We apply Reduction Rules [3.2.1](#), [3.4.1](#), and [3.4.2](#) exhaustively. Since each of the reduction rules either deletes an edge or a vertex, the number of times these rules are applied is bounded by $\mathcal{O}(n)$. Furthermore, since each of the reduction rules can be applied in polynomial time, our kernelization algorithm runs in polynomial time.

Bounding the size. Let $(\mathcal{I}, k)$ denote the reduced instance of MAX-SMTI. That is, an instance on which Reduction Rules [3.2.1](#), [3.4.1](#), and [3.4.2](#) are no longer applicable.

Lemma 3.4.4. *MAX-SMTI parameterized by the solution size k admits $\mathcal{O}(k)$ sized kernel on planar graphs.*

Proof. Our objective is to show that

$$|I_{\deg=1}| \leq |X| + 1, |I_{\deg=2}| \leq 12|X| \text{ and } |I_{\deg \geq 3}| \leq 4|X|.$$

Once these relations are proved, an application of Reduction Rules [3.2.1](#), [3.4.1](#) and [3.4.2](#) yields an instance $(\mathcal{I}, k)$ of MAX-SMTI in which the independent set I

has size $|I_{\deg=1}| + |I_{\deg=2}| + |I_{\deg \geq 3}| = \mathcal{O}(|X|)$. Thus, the vertex set of the kernel has size $|X| + |I| = \mathcal{O}(k)$. As a planar graph on n vertices has $\mathcal{O}(n)$ edges [35], we conclude that the edge set of the kernel has size $\mathcal{O}(k)$.

The first relation follows from Reduction Rule [3.4.1] in conjunction with Lemma [3.4.1]. Recall, that each vertex in each equivalence class sees *exactly* two neighbours in X . This implies that after the application of Reduction Rule [3.4.2], each equivalence class has at most four vertices. Since the number of equivalence classes is upper bounded by $|Y_2|$, Lemma [3.4.1] yields $|I_{\deg=2}| \leq 4|Y_2| \leq 12|X|$. For the bound on $|I_{\deg \geq 3}|$, we note that for any $W \in Y_{\geq 3}$, there can be at most two vertices in I whose neighborhood is W . Otherwise, $K_{3,3}$ is a minor of G , a contradiction to its planarity. Thus, $|I_{\deg \geq 3}| \leq 2|Y_{\geq 3}| \leq 4|X|$, from Lemma [3.4.1]. $\diamond$

We provide details necessary to follow the proof of Theorem [6]. A *tree decomposition* of a graph G is a pair (T, β) of a tree T and $\beta : V(T) \rightarrow 2^{V(G)}$. We assume T is rooted. For $v \in V(T)$, $\beta(v)$ is the *bag* of v , and $\gamma(v)$ is the union of the bags of v and its descendants in T . We let $\text{tw}(G)$ denote the treewidth of G . For a detailed definition, see [29]. Next we define the notion of treewidth which is important in designing our subexponential algorithms. We recall the definition first.

Definition 3.4.1. A tree decomposition of a graph G is a pair (T, β) of a tree T and $\beta : V(T) \rightarrow 2^{V(G)}$, such that

1. $\bigcup_{v \in V(T)} \beta(v) = V(G)$, and
2. for any edge $\{\rho, \rho'\} \in E(G)$ there exists a node $v \in V(T)$ such that $\rho, \rho' \in \beta(v)$,
and
3. for any vertex $\rho \in V(G)$, the subgraph of T induced by the set $T_\rho = \{v \in V(T) : \rho \in \beta(v)\}$ is a tree.

The width of (T, β) is $\max_{v \in V(T)} \{|\beta(v)|\} - 1$. The treewidth of G is the minimum width of a tree decomposition of G .

For $v \in V(T)$, we say that $\beta(v)$ is the *bag* of v , and $\gamma(v)$ denotes the union of the bags of v and the descendants of v in T . We let $\text{tw}(G)$ denote the treewidth of G .

Given an n -vertex graph G and an integer q , in time $2^{\mathcal{O}(q)}n^{\mathcal{O}(1)}$, we can either output a tree decomposition of G of width at most $4q + 4$ or conclude that $\text{tw}(G) > q$ [29, 37]. The well-known results of Lipton and Tarjan [109] and Robertson and Seymour [136] imply that a n -vertex planar graph has treewidth $\mathcal{O}(\sqrt{n})$. Thus, using Lemma 3.4.4, we conclude that the reduced instance of MAX-SMTI has treewidth $\mathcal{O}(\sqrt{k})$. In time $2^{\mathcal{O}(\sqrt{k})}$, we then obtain a tree decomposition of $\mathcal{I}$ of width $\mathcal{O}(\sqrt{k})$. Thus, to design a subexponential time parameterized algorithm for MAX-SMTI on planar graphs, we need to design a faster algorithm on graphs of bounded treewidth. In particular, we need Theorem 6.

Proof of Theorem 6. The main idea of the proof in the context of MAX-SRTI (which encompasses MAX-SMTI) is as follows. We let (T, β) denote the input tree decomposition of G . First, in polynomial time, we transform (T, β) into a *nice* tree decomposition of the same width [18]; the number of nodes in this decomposition is bounded by $\mathcal{O}(n)$. Briefly, a nice tree decomposition is one where T is a binary tree, a node with two children has the same bag as its children, and any other node either has an empty bag or a bag that differs by exactly one vertex from the bag of its child. Moreover, the bag of the root, denoted by r , is empty. Let μ be a matching in graph G such that $\beta(v)$ is a bag in G .

For a function $f : \beta(v) \rightarrow V(G)$, we say μ *complies* with f if for every $x \in \beta(v)$, $f(x) = \mu(x)$.

Our algorithm is based on dynamic programming (DP). Our DP table, $\mathbf{N}$, contains an entry $\mathbf{N}[v, f]$ for every node v of T and injective function $f : \beta(v) \rightarrow V(G)$. Such an entry stores a value in $\{0, 1, \dots, \lfloor n/2 \rfloor\} \cup \{-\infty\}$. The meaning of the entry can be summarized as follows. $\mathbf{N}[v, f]$ should contain the maximum size of a stable matching

in $G[\gamma(v)]$, the subgraph of G induced by the vertex-set $\gamma(v)$, which complies with f . The value $-\infty$ indicates that no such stable matching exists.

Once the table is computed correctly, the solution is given by the value stored in $N[r, f]$ where f is the function whose domain is the empty set. Roughly speaking, the basis corresponds to leaves (whose bags are empty), and are initialized to store 0.

(Join Node) For a node v with two children v_1 and v_2 , we compute the sum of the entries associated with these children (where f is consistent), from which we decrease the number of edges matched by f as they are over counted (not counting vertices matched to themselves). Let $n_f = |\{x \in \beta(v) \mid f(x) \neq x\}|$, denoting the number of edges matched by f but excluding those that represent vertices matched to themselves. Then,

$$N[v, f] = N[v_1, f] + N[v_2, f] - n_f.$$

(Forget Node) Given a node v whose bag contains one fewer vertex than its child v' , we examine each option of matching this vertex, and take the one that maximizes the entry of the child. So, for any function f' such that when f' is restricted to $\beta(v)$ we get $f = f'|_{\beta(v)}$, then,

$$N[v, f] = \max_{f'} N[v', f']$$

(Introduce Node) Finally, given a node v whose bag contains one more vertex, x , than its child v' , we first check if the matching given by f is “locally” stable and “valid” in the following sense: if $x \in \beta(v)$, then either $x = f(x)$, or x and $f(x)$ are neighbors in G and $x = f(f(x))$ such that for every neighbor y of x that is present in $\beta(v)$, (x, y) does not form a blocking edge. The latter condition can be tested as the partners of both x and y are given by f . If the answer is negative, we assign $-\infty$. Otherwise, we consider the (unique) entry of the child that is consistent with the current entry, to which we add 1 if $x \neq f(x)$ and x ’s matched partner is not

already present in the current bag. Hence, we obtain the following,

$$\mathbf{N}[v, f] = \begin{cases} -\infty & \text{if } f \text{ is not valid} \\ \mathbf{N}[v', f|_{\beta(v')}] & \text{if } f(x) = x \text{ or } f(x) \in \beta(v) \\ \mathbf{N}[v', f|_{\beta(v')}] + 1 & \text{otherwise} \end{cases}$$

The proof of correctness of the formulas is given in the next lemma.

Lemma 3.4.5. *The recurrences in the proof of Theorem [6](#) are correct.*

Proof. We analyze the three nodes, Join, Forget and Introduce separately. We begin by recalling that $\mathbf{N}[v, f]$ contains the maximum size of a stable matching in $G[\gamma(v)]$, the subgraph of G induced by the vertex-set $\gamma(v)$, which *complies* with f .

Let v be a leaf node. Since, $\beta(v) = \emptyset$, we only have one value $\mathbf{N}[v, \cdot] = 0$. We will prove that the formulas hold in each of the following cases by showing inequality in both directions.

Join Node: Let v be a join node with children v_1 and v_2 . So, $\beta(v) = \beta(v_1) = \beta(v_2)$. Let μ be the maximum size stable matching that *complies* with f . Let μ_1 and μ_2 denote the matchings obtained by restricting μ to $G[\gamma(v_1)]$ and $G[\gamma(v_2)]$, respectively. Note that $\gamma(v_1) \cap \gamma(v_2) = \beta(v)$, and n_f is the contribution of $\beta(v)$ to $\mathbf{N}[v, f]$. Also, $\mathbf{N}[v_i, f] \geq |\mu_i|$, for each $i \in [2]$. So, $\mathbf{N}[v_1, f] + \mathbf{N}[v_2, f] \geq |\mu_1| + |\mu_2|$. However, in $|\mu_1| + |\mu_2|$ we add the contribution of $\beta(v)$ twice. Therefore we deduct n_f , thereby yielding $\mathbf{N}[v_1, f] + \mathbf{N}[v_2, f] - n_f \geq \mathbf{N}[v, f]$.

On the other hand, let μ_1 and μ_2 denote the matchings which attain the maximum for $\mathbf{N}[v_1, f]$ and $\mathbf{N}[v_2, f]$. Note that both μ_1 and μ_2 comply with f . Hence, $\mu_1 \cup \mu_2$ is a matching in $G[\gamma(v)]$, and $\mathbf{N}[v_1, f] + \mathbf{N}[v_2, f] - n_f \leq \mathbf{N}[v, f]$ follow as a consequence. Thus, we can conclude that $\mathbf{N}[v_1, f] + \mathbf{N}[v_2, f] - n_f = \mathbf{N}[v, f]$.

Forget Node: Let μ be the maximum size stable matching of $G[\gamma(v)]$ that *complies*

with f . Let us define f' as follows.

$$f'(y) = \begin{cases} f(y) & \text{if } y \in \beta(v) \\ \mu(y) & \text{if } y = x \end{cases}$$

Since μ is one of the matchings that is considered in the definition of $\mathbf{N}[v, f]$, we have, $\max_{f'} \mathbf{N}[v', f'] \geq \mathbf{N}[v, f]$. Therefore, for a function f' such that $f'|_v = f$, we have $\max_{f'} \mathbf{N}[v', f'] \geq \mathbf{N}[v, f]$.

On the other hand, observe that, for any function f' such that $f = f'|_v$, each matching that is considered in the definition of $\mathbf{N}[v', f']$ is also considered in the definition of $\mathbf{N}[v, f]$. This implies, that for any such f' , $\mathbf{N}[v, f] \geq \mathbf{N}[v', f']$; so $\mathbf{N}[v, f] \geq \max_{f'} \mathbf{N}[v', f']$.

Introduce Node: Let v be a node whose bag contains one more vertex, x , than its child v' . We first consider the case when f is locally stable. That is, for any neighbor y of x , (x, y) does not form a blocking edge. So, there is a stable matching that *complies* with f , call it μ' . The maximum size of μ' in $G[\gamma(v')]$ is $\mathbf{N}[v', f|_{\beta(v')}]$. Since $\gamma(v) = \gamma(v') \cup \{x\}$, the size of the maximum size stable matching in $G[\gamma(v)]$ is at most $\mathbf{N}[v', f|_{\beta(v')}] + 1$. Hence, $\mathbf{N}[v, f] \leq \mathbf{N}[v', f|_{\beta(v')}] + 1$. For the case where $f(x) = x$ or $f(x) \in \beta(v)$ we note that $\mathbf{N}[v, f] \leq \mathbf{N}[v', f|_{\beta(v')}]$.

On the other hand, let μ' be a matching for which maximum is attained in the definition of $\mathbf{N}[v', f|_{\beta(v')}]$, thus, $\mathbf{N}[v', f|_{\beta(v')}] = |\mu'|$. Since f is valid, $\mu = \mu' \cup \{(x, f(x))\}$ is a stable matching in $G[\gamma(v)]$. Also, note that μ *complies* with f . If x 's matched partner is not already present in $\beta(v)$ and $f(x) \neq x$, then $\mathbf{N}[v, f] \geq |\mu'| + 1 = \mathbf{N}[v', f|_{\beta(v')}] + 1$. Otherwise, $\mathbf{N}[v, f]$ is at least $|\mu'|$. Then, $\mathbf{N}[v, f] \geq \mathbf{N}[v', f|_{\beta(v')}]$.

In the case, where f is not locally stable there is no stable matching that *complies* with f . So, we correctly assign $-\infty$.

This concludes the description and the proof of correctness of the recursive

formulas for computing the values of $\mathbf{N}[\cdot, \cdot]$. $\diamond$

Since $\mathbf{N}$ contains $n^{\mathcal{O}(\mathbf{tw})}$ entries, each computable in time $n^{\mathcal{O}(\mathbf{tw})}$, this concludes the proof of the theorem. $\diamond$

We thus conclude the proofs of Theorems [4](#) and [5](#).

3.5 Conclusions

We designed polynomial kernels and parameterized algorithms for MAX-SMTI and MIN-SMTI parameterized by the solution size, and for MAX-SRTI parameterized by the size of a maximum matching. In addition, we studied these problems when the input is restricted to planar graphs. Our algorithms for general graphs run in time $2^{\mathcal{O}(k \log k)} + n^{\mathcal{O}(1)}$. Hence, it would be interesting to either design an algorithm with running time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$, or show an appropriate conditional lower bound. Moreover, it is an interesting open question to determine whether MAX-SMTI, MIN-SMTI and MAX-SRTI are FPT parameterized by the treewidth of the input graph. Finally, our study also gives rise to possible further investigations of above guarantee parameterizations of these problems.

Chapter 4

Balanced Stable Matching

In the previous chapter we saw that finding a maximum size stable matching is NP-hard in presence of ties. In this chapter, we study another NP-hard variant of the STABLE MARRIAGE problem. We study the classical model of STABLE MARRIAGE, that is, when the preference lists are strict linear ordering. The seminal paper [58] by Gale and Shapely on stable matchings shows that given an instance of STABLE MARRIAGE, a stable matching necessarily *exists* and can be found in time linear in the size of the input instance. This algorithm produces only “special kind” of stable matchings. However, there can be exponentially many stable matchings of the instance. Finding a stable matching that satisfy certain criteria can be hard, as we will see in this is chapter.

In the recent past, a collective effort to study matching under preferences through the lens of Parameterized Complexity was initiated [106, 23, 126, 69, 80, 77, 68, 124, 107, 116, 117, 17]. Along with [17, 23], our work has initiated the study of *kernelization* in the context of matchings under preferences. To the best of our knowledge, our work is the first to introduce “above guarantee parameterization” to this topic.

In Matching under Preferences, a matching is an allocation (or assignment) of

agents to resources that satisfies some predefined criterion of compatibility/acceptability. Here, (arguably) the best known model is the *two-sided model*, where the agents on one side are referred to as *men*, and the agents on the other side are referred to as *women*. A few illustrative examples of real life situations where this model is relevant include matching residents to hospitals, students to colleges, kidney patients to donors and users to servers in a distributed Internet service. At the heart of all of these applications lies the fundamental STABLE MARRIAGE problem. In particular, the Nobel Prize in Economics was awarded to Shapley and Roth in 2012 “for the theory of stable allocations and the practice of market design.” Moreover, several books have been dedicated to the study of STABLE MARRIAGE as well as optimization variants of this classical problem such as the EGALITARIAN STABLE MARRIAGE, MINIMUM REGRET STABLE MARRIAGE, SEX-EQUAL STABLE MARRIAGE and BALANCED STABLE MARRIAGE problems [71, 99, 114, 70].

The input of STABLE MARRIAGE consists of a set of men, M , and a set of women, W . Each agent a has a set of *acceptable partners*, $\mathcal{A}(a)$, who are subjectively ranked by a in a strict order. Consequently, each agent a has a so-called *preference list*, where $p_a(b)$ is the position of $b \in \mathcal{A}(a)$ in a ’s preference list. Lower values of positions are associated with more preferred agents. Without loss of generality, it is assumed that if an agent a ranks an agent b , then the agent b ranks the agent a as well. The sets of preference lists of the men and the women are denoted by $\mathcal{L}_M$ and $\mathcal{L}_W$, respectively. In this context, we say that a pair of a man and a woman, (m, w) , is an *acceptable pair* if both $m \in \mathcal{A}(w)$ and $w \in \mathcal{A}(m)$. Accordingly, the notion of a *matching* refers to a matching between men and women, where two people matched to one another form an acceptable pair. Roughly speaking, the goal of the STABLE MARRIAGE problem is to *find* a matching that is *stable* in the following sense: there should not exist two people who prefer being matched to one another over their current “status”. More precisely, a matching μ is said to be stable if it does not have a *blocking pair*, which is an acceptable pair (m, w) such that **(i)** either

m is unmatched by μ or $p_m(w) < p_m(\mu(m))$, and **(ii)** either w is unmatched by μ or $p_w(m) < p_w(\mu(w))$. Here, the notation $\mu(a)$ indicates the person to whom μ matches the person a . Note that a person always prefers being matched to an acceptable partner over being unmatched.

The seminal paper [58] by Gale and Shapely on stable matchings shows that given an instance of STABLE MARRIAGE, a stable matching necessarily *exists*, but it is not necessarily unique. In fact, for a given instance of STABLE MARRIAGE, there can be an *exponential* number of stable matchings, and they should be viewed as a *spectrum* where the two extremes are known as the *man-optimal stable matching* and the *woman-optimal stable matching*. Formally, the man-optimal stable matching, denoted by μ_M , is a stable matching such that every stable matching μ satisfies the following condition: every man m is either unmatched by both μ_M and μ or $p_m(\mu_M(m)) \leq p_m(\mu(m))$. The woman-optimal stable matching, denoted by μ_W , is defined analogously. In the μ_M (resp. μ_W) the men (resp. women) receive their best possible partners and women (resp. men) receive their worst possible partners among the set stable matchings. These two extremes, which give the best possible solution for one party at the expense of the other party, always exist and can be computed in polynomial time [58]. Naturally, it is desirable to analyze matchings that lie somewhere in the middle. Here, the quantity $p_a(\mu(a))$ is the “dissatisfaction” of a in a matching μ , where a smaller value signifies a lesser amount of dissatisfaction. The most well-known measures are as follows:

- μ is *globally desirable* if it minimizes $\sum_{(m,w) \in \mu} (p_m(\mu(m)) + p_w(\mu(w)))$, called the *egalitarian stable matching*;
- μ is *minimum regret* if it minimizes $\max_{(m,w) \in \mu} \{\max\{p_m(\mu(m)), p_w(\mu(w))\}\}$, called the *minimum regret stable matching*;
- μ is *fair towards both sides* if it minimizes $|\sum_{(m,w) \in \mu} p_m(\mu(m)) - \sum_{(m,w) \in \mu} p_w(\mu(w))|$, called the *sex-equal stable matching*;

- μ is *desirable by both sides* if it minimizes $\max\{\sum_{(m,w) \in \mu} p_m(w), \sum_{(m,w) \in \mu} p_w(m)\}$, called the *balanced stable matching*.

Each notion above leads to a natural, *different* well-studied optimization problem (see Related Work). We focus on the NP-hard BALANCED STABLE MARRIAGE (BSM) problem, where the objective is to find a stable matching μ that minimizes

$$\text{balance}(\mu) = \max\left\{\sum_{(m,w) \in \mu} p_m(w), \sum_{(m,w) \in \mu} p_w(m)\right\}.$$

This problem was introduced in the influential work of Feder [46] on stable matchings, and was shown to be NP-complete and admitting a 2-approximation algorithm.

Our Contribution. Above guarantee parameterization is a topic of extensive study in Parameterized Complexity [29]. We introduce two “above-guarantee parameterizations” of BSM. Consider the minimum value O_M (O_W) of the total dissatisfaction of men (women) realizable by a stable matching. Formally, $O_M = \sum_{(m,w) \in \mu_M} p_m(w)$, and $O_W = \sum_{(m,w) \in \mu_W} p_w(m)$, where μ_M (μ_W) is the man-optimal (woman-optimal) stable matching. Denote $\text{Bal} = \min_{\mu \in \text{SM}} \text{balance}(\mu)$, where SM is the set of all stable matchings. An input integer k would indicate that our objective is to decide whether $\text{Bal} \leq k$. In SM, all stable matchings match the same set of agents, A^* which can be computed in polynomial time. As $\text{balance}(\mu) \geq \frac{|A^*|}{2}$ for any stable matching μ , BSM is trivially fixed-parameter tractable (FPT) with respect to k . In our first parameterization, the parameter is $k - \min\{O_M, O_W\}$, and in the second one, it is $k - \max\{O_M, O_W\}$.

ABOVE-MIN BALANCED STABLE MARRIAGE (ABOVE-MIN BSM)

Input: An instance $(M, W, \mathcal{L}_M, \mathcal{L}_W)$ of BALANCED STABLE MARRIAGE, and a non-negative integer k .

Parameter: $t = k - \min\{O_M, O_W\}$

Task: Is $\text{Bal} \leq k$?

ABOVE-MAX BALANCED STABLE MARRIAGE (ABOVE-MAX BSM)

Input: An instance $(M, W, \mathcal{L}_M, \mathcal{L}_W)$ of BALANCED STABLE MARRIAGE, and a non-negative integer k .

Parameter: $t = k - \max\{O_M, O_W\}$

Task: Is $\text{Bal} \leq k$?

Choice of parameters: Note that the least dissatisfaction the party of men can hope for (call it *minimum dissatisfaction*) is O_M , and the least dissatisfaction the party of women can hope for (also call it *minimum dissatisfaction*) is O_W . First, consider the parameter $t = k - \min\{O_M, O_W\}$. Whenever we have a solution such that the amounts of dissatisfaction of *both* parties are *close enough* to the minimum, this parameter is small. (When the parameter is small, we cannot simply pick μ_M or μ_W since $\text{balance}(\mu_M)$ and $\text{balance}(\mu_W)$ can be *arbitrarily larger* than $\min\{O_M, O_W\}$.) Indeed, the closer the dissatisfaction of both parties to the minimum (which is exactly the case where both parties would find the solution desirable), the smaller the parameter is, and the smaller the parameter is, the faster a parameterized algorithm is. In this above guarantee parameterization, the guarantee value (i.e., $\min\{O_M, O_W\}$) is already quite high—for example, our parameter is significantly smaller than $k - n'$, where n' is the number of men (or women) matched by a stable matching, since $k - \min\{O_M, O_W\}$ is (i) never larger than $k - n'$, and (ii) can be

arbitrarily smaller than $k - n'$, e.g. $k - n'$ can be of the magnitude of $\mathcal{O}(n)$ while our parameter is 0.

Since we are taking the *minimum* of $\{O_M, O_W\}$, we need the dissatisfaction of *both* parties to be close to optimum in order to have a small parameter. As we are able to show that BSM is FPT with respect to this parameter, it is very natural to next examine the case where we take the *max* of $\{O_M, O_W\}$. In this case, the closer the dissatisfaction of *at least one* party to the optimum, the smaller the parameter is. In other words, now to have a small parameter, the demand from a solution is weaker than before. In the vocabulary of Parameterized Complexity, it is said that the parameterization by $t = k - \max\{O_M, O_W\}$ is “above a higher guarantee” than the parameterization by $t = k - \min\{O_M, O_W\}$, since it is *always* the case that $\max\{O_M, O_W\} \geq \min\{O_M, O_W\}$. Interestingly, as we show in this chapter, the parameterization by $k - \max\{O_M, O_W\}$ results in a problem that is W[1]-hard. Hence, the complexities of the two parameterizations behave very differently. We remark that in Parameterized Complexity, it is *not at all* the rule that when one takes an “above a higher guarantee” parameterization, the problem would become W[1]-hard, as can be evidenced by the VERTEX COVER problem, the classical above guarantee parameterizations in this field, for which three distinct above guarantee parameterizations yielded FPT algorithms [63, 112, 30, 134]. Overall, our results draw a nontrivial line between tractability and intractability of above guarantee parameterization of BSM.

Finally, the three main theorems that we establish in this study are as follows.

Theorem 7. ABOVE-MAX BSM is W[1]-hard.

Theorem 8. ABOVE-MIN BSM admits a kernel that has at most $3t$ men, at most $3t$ women, and such that each person has at most $2t + 1$ acceptable partners.

Theorem 9. ABOVE-MIN BSM can be solved in time $\mathcal{O}^*(8^t)$.

¹ $\mathcal{O}^*$ hides the factors polynomial in input size.

Our techniques: The overview of the reduction we develop to prove Theorem 7 is presented followed by the formal analysis in Section 5.2. The proof of Theorem 8 is based on the introduction and analysis of a variant of ABOVE-MIN BSM which we will call FUNCTIONAL BSM or ABOVE-MIN FBSM . Our kernelization algorithm consists of several phases, each simplifying a different aspect of ABOVE-MIN BSM, and shedding light on structural properties of the YES-instances of this problem. We stress that the design and order of our reduction rules are very carefully tailored to ensure correctness. For example, it is tempting to execute Step 2 before Step 1 (see the outline in Section 4.3.1), but this is simply incorrect as it alters the set of stable matchings. The reduction rules are easy to express for this functional variant of ABOVE-MIN BSM. Hence, we choose to define the reduction rules for this functional variant of ABOVE-MIN BSM instead of ABOVE-MIN BSM directly. Note that Theorem 8 already implies that ABOVE-MIN BSM is FPT. To obtain a parameterized algorithm whose running time is single exponential in the parameter (Theorem 9), we utilize the method of bounded search trees on top of our kernel in Section 4.4.

Related Work. The problem of finding an egalitarian stable matching (defined in page 41), called EGALITARIAN STABLE MARRIAGE (ESM), is known to be solvable in polynomial time due to Irving et al. [86]. If preference lists have so called *ties* or agents do not have genders (i.e., in *roommates* setting), then ESM is NP-hard [85]. ESM is one of the most well studied topics in Matching under Preferences (see [71, 99, 114]), and the survey of these results is outside the scope of our study. To the best of our knowledge, in the framework of Parameterized Complexity, the only known work is by Chen et al. [23] (which presents positive FPT results for the generalizations above). ESM does not distinguish between men and women, and therefore it does not fit scenarios where it is necessary to differentiate between the individual satisfaction of each party. In such scenarios, SEX-EQUAL STABLE

MARRIAGE (SESM) and BSM come into play.

In the SESM problem, the objective is to find a sex-equal stable matching, a stable matching that is fair towards both sides by minimizing the difference between their individual amounts of satisfaction. Unlike EGM, the SESM problem is known to be NP-hard [94]. At first sight, the balance measure might seem conceptually similar to the sex-equal measure, but in fact, the two measures are quite different. Indeed, BSM does not attempt to find a stable matching that is fair, but one that is desirable by both sides. In other words, BSM examines the amount of dissatisfaction of each party *individually*, and attempts to minimize the worse one among the two. BSM models the scenario where each party is selfish in the sense that it desires a matching where its own dissatisfaction is minimized, irrespective of the dissatisfaction of the other party, and our goal is to find a matching desirable by both parties by ensuring that each individual amount of dissatisfaction does not exceed some threshold. However, even if parties are not selfish, a balanced stable matching may be preferable over a sex-equal one—indeed, in some situations, a sex-equal stable matching may fit the old phrase “the sorrow of many is the comfort of fools”, as it can achieve equality not by making one party better off at the expense of the other, but simply by making both parties worse. Thus, although the amount of literature devoted to SESM is much greater than the one about BSM, we find BSM at least as important as SESM. The survey of results about SESM is outside the scope of our study, but let us remark that in some situations, the minimization of $\text{balance}(\mu)$ may indirectly also minimize $|\delta(\mu)|$ where $\delta(\mu) = \sum_{(m,w) \in \mu} p_m(\mu(m)) - \sum_{(m,w) \in \mu} p_w(\mu(w))$, but in other situations, this may not be the case. Indeed, McDermid [122] constructed a family of instances where there does *not* exist any matching that is both a sex-equal stable matching and a balanced stable matching (the construction is also available in the book [114]).

Manlove discusses that BSM also admits a $(2 - 1/\ell)$ -approximation algorithm

where ℓ is the maximum size of a set of acceptable partners [114]. O’Malley [129] phrased the BSM problem in terms of constraint programming. The parameterized complexity of BSM with respect to treewidth parameterizations (of graphs associated with the problem) was studied in [69]. Recently, McDermid and Irving [123] expressed explicit interest in the design of fast exact exponential-time algorithms for BSM. McDermid and Irving [123] showed that SESM is NP-hard even if it is only necessary to decide whether the target $\Delta = \min_{\mu \in \text{SM}} |\delta(\mu)|$ is 0 or not [123], where SM is the set of all stable matchings of the instance. In particular, this means that SESM is not only W[1]-hard with respect to Δ , but it is even paraNP-hard with respect to this parameter.² In the case of BSM, however, the problem is trivially FPT with respect to the target $\text{Bal} = \min_{\mu \in \text{SM}} \text{balance}(\mu)$.

In the framework of Parameterized Complexity, Chen et al. [23] showed that deciding whether there exists a matching with at most k blocking pairs in the roommates setting is W[1]-hard. In addition to BSM, Gupta et al. [69] also considered treewidth parameterizations of other optimization variants of STABLE MARRIAGE. Mních and Schlotter [126] inspected the parameterized complexity of an optimization variant of STABLE MARRIAGE where blocking pairs were allowed to exist in order to ensure that some agents will be matched by the output matching. Meeks and Rastegari [124] introduced a parameterization regarding “agent types” and studied several optimization variants of STABLE MARRIAGE with respect to it. Marx and Schlotter [116] studied size optimization variants of STABLE MARRIAGE in the presence of ties, where the parameters are the total length, maximum length, and number of ties. In addition, Marx and Schlotter [117] studied a generalization of STABLE MARRIAGE, called HOSPITALS/RESIDENTS WITH COUPLES, with the number of couples as a parameter. Other studies of parameterized complexity of problems (less closely) related to STABLE MARRIAGE are of the GROUP ACTIVITY

²If a parameterized problem is NP-hard even when the value of the parameter is a fixed constant (that is, independent of the input), then the problem is said to be paraNP-hard.

SELECTION problem [80, 77, 68] and of the STABLE INVITATIONS problem [106]. Some works that are not directly related to our work but study the parameterized complexity of problems that involve restricted preference lists are [82, 1, 15, 72, 87].

4.1 Preliminaries

Throughout the chapter, whenever the instance $\mathcal{I}$ of BSM under discussion is not clear from context or we would like to put emphasis on $\mathcal{I}$, we add “($\mathcal{I}$)” to the appropriate notation. For example, we use the notation $t(\mathcal{I})$ rather than t . When we would like to refer to the balance of a stable matching μ in a specific instance $\mathcal{I}$, we would use the notation $\text{balance}_{\mathcal{I}}(\mu)$. A matching is called a *perfect matching* if it matches every person (to some other person).

A Functional Variant of STABLE MARRIAGE. To obtain our kernelization algorithm for ABOVE-MIN BSM, it will be convenient to work with, as explained below, a “functional” definition of preferences, resulting in a “functional” variant of this problem which we call ABOVE-MIN FBSM. Here, instead of the sets of preferences lists $\mathcal{L}_M$ and $\mathcal{L}_W$, the input consists of sets of preference *functions* $\mathcal{F}_M$ and $\mathcal{F}_W$, where $\mathcal{F}_M$ replaces $\mathcal{L}_M$ and $\mathcal{F}_W$ replaces $\mathcal{L}_W$. Specifically, every person $a \in M \cup W$ has an injective (one-to-one) function $f_a : \mathcal{A}(a) \rightarrow \mathbb{N}$, called a *preference function*. Similar models are known in literature [132]. Intuitively, a lower function value corresponds to a higher preference. The preference functions are injective because each person’s preference list is a strict order of his/her acceptable partners. Consequently, every preference function defines a total ordering over a set of acceptable partners. Note that all of the definitions presented in the introduction extend to our functional variant in the natural way. For the sake of formality, we will specify the necessary adaptations. Our kernelization algorithm for ABOVE-MIN

BSM works as follows: Firstly, we create an instance of ABOVE-MIN FBSM, secondly, we run the kernelization algorithm for ABOVE-MIN FBSM to obtain a kernel for our instance; and finally, we transform that kernel into a kernel for the original instance of ABOVE-MIN BSM.

We adapt standard definitions presented for ABOVE-MIN BSM to the case where preferences are specified by functions rather than lists. The input of the FUNCTIONAL STABLE MARRIAGE problem consists of a set of men, M , and a set of women, W . Each person a has a set of *acceptable partners*, denoted by $\mathcal{A}(a)$, and an injective function $f_a : \mathcal{A}(a) \rightarrow \mathbb{N}$ called a *preference function*. Without loss of generality, it is assumed that if a person a belongs to the set of acceptable partners of a person b , then the person b belongs to the set of acceptable partners of the person a . The set of preference functions of the men and the women are denoted by $\mathcal{F}_M$ and $\mathcal{F}_W$, respectively. A pair of a man and a woman, (m, w) , is an *acceptable pair* if both $m \in \mathcal{A}(w)$ and $w \in \mathcal{A}(m)$. Accordingly, the notion of a *matching* refers to a matching between men and women, where two people that are matched to one another form an acceptable pair. A matching μ is *stable* if it does not have a *blocking pair*, which is an acceptable pair (m, w) such that **(i)** either m is unmatched by μ or $f_m(w) < f_m(\mu(m))$, and **(ii)** either w is unmatched by μ or $f_w(m) < f_w(\mu(w))$. The goal of the FUNCTIONAL STABLE MARRIAGE problem is to find a stable matching.

The man-optimal stable matching, denoted by μ_M , is a stable matching such that every stable matching μ satisfies the following condition: every man m is either unmatched by both μ_M and μ or $f_m(\mu_M(m)) \leq f_m(\mu(m))$. The woman-optimal stable matching, denoted by μ_W , is defined analogously. Given a stable matching μ , define $\text{balance}(\mu) = \max\{\sum_{(m,w) \in \mu} f_m(w), \sum_{(m,w) \in \mu} f_w(m)\}$. Moreover, $\text{Bal} = \min_{\mu \in \text{SM}} \text{balance}(\mu)$, where SM is the set of all stable matchings, $O_M = \sum_{(m,w) \in \mu_M} f_m(w)$, and $O_W = \sum_{(m,w) \in \mu_W} f_w(m)$. Finally, ABOVE-MIN FBSM is defined as follows.

ABOVE-MIN FUNCTIONAL BALANCED STABLE MARRIAGE (ABOVE-MIN FBSM)

Input: An instance $(M, W, \mathcal{F}_M, \mathcal{F}_W)$ of FUNCTIONAL BALANCED STABLE MARRIAGE, and a non-negative integer k .

Parameter: $t = k - \min\{O_M, O_W\}$

Task: Is $\text{Bal} \leq k$?

Clearly, it is straightforward to turn an instance of ABOVE-MIN BSM into an equivalent instance of ABOVE-MIN FBSM as stated in the following observation.

Observation 4.1.1. *Let $\mathcal{I} = (M, W, \mathcal{L}_M, \mathcal{L}_W, k)$ be an instance of ABOVE-MIN BSM. For each $a \in M \cup W$, define $f_a : \mathcal{A}(a) \rightarrow \mathbb{N}$ by setting $f_a(b) = p_a(b)$ for all $b \in \mathcal{A}(a)$. Then, $\mathcal{I}$ is a YES-instance of ABOVE-MIN BSM if and only if $(M, W, \mathcal{F}_M = \{f_m\}_{m \in M}, \mathcal{F}_W = \{f_w\}_{w \in W}, k)$ is a YES-instance of ABOVE-MIN FBSM.*

4.1.1 Known Results

Here, we state several classical results, which were originally presented in the context of STABLE MARRIAGE. By their original proofs, these results also hold in the context of FUNCTIONAL STABLE MARRIAGE. To be more precise, given an instance of FUNCTIONAL STABLE MARRIAGE, we can construct an equivalent instance of STABLE MARRIAGE, by ranking the acceptable partners in the order of their function values, where a smaller value implies a higher preference. The instances are equivalent in the sense that they give rise to the exact same set of stable matchings. Hence, all the structural results about stable matchings in the usual setting (modeled by strict preference lists) apply to the generalized setting, modeled by injective functions.

Proposition 4.1.1 ([58]). *For any instance of STABLE MARRIAGE (or FUNCTIONAL STABLE MARRIAGE), there exist a man-optimal stable matching, μ_M , and*

a woman-optimal stable matching, μ_W , and both μ_M and μ_W can be computed in polynomial time.

The following powerful proposition is known as the Rural-Hospital Theorem (and notably does not hold for instances that have ties).

Proposition 4.1.2 ([60] Theorem 1). *Given an instance of STABLE MARRIAGE (or ABOVE-MIN FBSM), the set of men and women that are matched is the same for all stable matchings.*

We further need a proposition regarding the man-optimal and woman-optimal stable matchings that implies Proposition 4.1.2 [60].

Proposition 4.1.3 ([71]). *For any instance of STABLE MARRIAGE (or FUNCTIONAL STABLE MARRIAGE), every stable matching μ satisfies the following conditions: every woman w is either unmatched by both μ_M and μ or $p_w(\mu_M(w)) \geq p_w(\mu(w))$, and every man m is either unmatched by both μ_W and μ or $p_m(\mu_W(m)) \geq p_m(\mu(m))$.*

While designing our kernelization algorithm, we might be able to determine whether the input instance is a YES-instance or a NO-instance. For the sake of clarity, in the first case, we simply return YES, and in second case, we simply return NO. To properly comply with the definition of a kernel, the return of YES and NO should be interpreted as the return of a trivial YES-instance and a trivial NO-instance, respectively. Here, a trivial YES-instance can be the one in which $M = W = \emptyset$ and $k = 0$, where the only stable matching is the one that is empty and whose balance is 0, and a trivial NO-instance can be the one where $M = \{m\}$, $W = \{w\}$, $\mathcal{A}(m) = \{w\}$, $\mathcal{A}(w) = \{m\}$ and $k = 0$.

4.2 Hardness

In this section, we prove Theorem 7. For this purpose, we give a reduction from a W[1]-hard problem, CLIQUE [36]. Thus, to prove Theorem 7, it is sufficient to prove the next lemma. First, we define CLIQUE formally.

CLIQUE

Input: A graph $G = (V, E)$, and a positive integer k .

Question: Does G contain a complete graph on k vertices?

Parameter: k .

Lemma 4.2.1. *Given an instance $\mathcal{I} = (G = (V, E), k)$ of CLIQUE, an equivalent instance $\widehat{\mathcal{I}} = (M, W, \mathcal{L}_M, \mathcal{L}_W, \widehat{k})$ of ABOVE-MAX BSM with parameter $t = 6k + 3k(k - 1)$ can be constructed in time $f(k) \cdot |\mathcal{I}|^{\mathcal{O}(1)}$ for some function f .*

The goal is to construct (in “FPT time”) an instance $\widehat{\mathcal{I}} = (M, W, \mathcal{L}_M, \mathcal{L}_W, \widehat{k})$ of ABOVE-MAX BSM. The following subsections contain an informal explanation of the intuition underlying the gadget construction followed by the formal description and the analysis.

Reduction. Let $\mathcal{I} = (G = (V, E), k)$ be some instance of CLIQUE. We select arbitrary orders on V and E , and accordingly we denote $V = \{v_1, v_2, \dots, v_{|V|}\}$ and $E = \{e_1, e_2, \dots, e_{|E|}\}$.

First, to construct the sets M and W , we define three pairwise-disjoint subsets of M , called M_V, M_E and $\widetilde{M}$, and three pairwise-disjoint subsets of W , called W_V, W_E and $\widetilde{W}$. Then, we set $M = M_V \uplus M_E \uplus \widetilde{M} \uplus \{m^*\}$ and $W = W_V \uplus W_E \uplus \widetilde{W} \uplus \{w^*\}$, where m^* and w^* denote a new man and a new woman, respectively.

- $M_V = \{m_v^i : v \in V, i \in \{1, 2\}\}; W_V = \{w_v^i : v \in V, i \in \{1, 2\}\}.$
- $M_E = \{m_e^i : e \in E, i \in \{1, 2\}\}; W_E = \{w_e^i : e \in E, i \in \{1, 2\}\}.$

- Let $\delta = 2(|V| + |E| + |V||E| + |V||E|^2) - k(4 + 4k + 2|E| + (k - 1)|V||E|)$.
Then, $\widetilde{M} = \{\widetilde{m}^i : i \in \{1, 2, \dots, \delta\}\}$ and $\widetilde{W} = \{\widetilde{w}^i : i \in \{1, 2, \dots, \delta\}\}$.

Note that $|M| = |W|$. We remark that in what follows, we assume w.l.o.g. that $\delta \geq 0$ and $|V| > k + k(k - 1)/2$, else the size of the input instance $\mathcal{I}$ of CLIQUE is bounded by a function of k and can therefore, by using brute-force, be solved in FPT time.

Roughly speaking, each pair of men, m_v^1 and m_v^2 , represents a vertex, and we aim to ensure that either both men will be matched to their best partners (in the man-optimal stable matching) or both men will be matched to other partners (where there would be only *one* choice that ensures stability). Accordingly, we will guarantee that the choice of matching these two men to their best partners translates to *not* choosing the vertex they represent into the clique, and the other choice translates to choosing this vertex into the clique.

Now, having just the set M_V , we can encode selection of vertices into the clique, but we cannot ensure that the vertices we select indeed form a clique. For this purpose, we also have the set M_E which, in a manner similar to M_V , encodes selection of edges into the clique. By designing the instance in a way that the situation of the men in the man-optimal stable matching is significantly worse than that of the women in the women-optimal stable matching, we are able to ensure that at most $2(k + k(k - 1)/2)$ men in $M_V \cup M_E$ will not be assigned their best partners (here, we exploit the condition that $\text{balance}(\mu) \leq \widehat{k}$ for a solution μ). Here the man m^* plays a crucial role—by using dummy men and women (in the sets $\widetilde{M}$ and $\widetilde{W}$) that prefer each other over all other people, we ensure that the situation of m^* is always “extremely bad” (from his viewpoint), while the situation of his partner, w^* , is always “excellent” (from her viewpoint).

At this point, we first need to ensure that the edges that we select indeed connect

the vertices that we select. For this purpose, we carefully design our reduction so that when a pair of men representing some edge e obtain partners worse than those they have in the man-optimal stable matching, it must be that the men representing the endpoints of e have also obtained partners worse than those they have in the man-optimal stable matching, else stability will not be preserved—the partners of the men representing the endpoints of e will form blocking pairs together with the men representing e .

Finally we observe that we still need to ensure that among our $2(k + k(k-1)/2)$ distinguished men in $M_V \cup M_E$, which are associated with $k + k(k-1)/2$ selected elements (vertices and edges), there will be exactly $2k$ distinguished men from M_V and exactly $k(k-1)$ distinguished men from M_E , which would mean we have chosen k vertices and $k(k-1)/2$ edges. For this purpose, we construct an instance where for the women, it is only somewhat “beneficial” that the men in M_V will not be matched to their best partners, but it is extremely beneficial that the men in M_E will not be matched to their best partners. This objective is achieved by carefully placing dummy men (from $\widetilde{M}$) in the preference lists of women in W_E . By again exploiting the condition that $\text{balance}(\mu) \leq \widehat{k}$ for a solution μ , we are able to ensure that there would be at least $k(k-1)$ distinguished men from M_E . We are now ready to present the formal details.

4.2.1 Formal Description of the Reduction

Next, we proceed with the formal presentation of our reduction by defining preference of every man $m \in M$, and thus constructing $\mathcal{L}_M$, given in Table [4.1](#). Accordingly, we define preference of every woman $w \in W$, and thus construct $\mathcal{L}_W$, given in Table [4.2](#).

Finally, we define $\widehat{k} = |M| + \delta + 6(k + \frac{k(k-1)}{2})$. It is clear that the entire construction (under the assumptions that $\delta \geq 0$ and $|V| > k + \frac{k(k-1)}{2}$) can be

M	Preference List	
m_v^1	$w_v^1, \tilde{w}^1, \tilde{w}^2, w_v^2$	for all $m_v^1 \in M_V$
m_v^2	$w_v^2, \tilde{w}^1, \tilde{w}^2, w_v^1$	for all $m_v^2 \in M_V$
$m_{\{u,v\}}^1$	$w_{\{u,v\}}^1, w_u^1, w_v^1, w_{\{u,v\}}^2$	for all $m_{\{u,v\}}^1 \in M_E$ where ³ $u < v$
$m_{\{u,v\}}^2$	$w_{\{u,v\}}^2, w_u^2, w_v^2, w_{\{u,v\}}^1$	for all $m_{\{u,v\}}^2 \in M_E$ where $u < v$
$\tilde{m}^i$	$\tilde{w}^i, w_{e_1}^1, w_{e_2}^1, \dots, w_{e_{ E }}^1, w_{e_1}^2, w_{e_2}^2, \dots, w_{e_{ E }}^2, [X]$ such that $i \leq V E $, $X = \{w_v^j \in W_V : \tilde{m}^i \in \mathcal{A}(w_v^j), j \in \{1, 2\}\}$	for all $\tilde{m}^i \in \tilde{M}$
$\tilde{m}^i$	$\tilde{w}^i$	for all $\tilde{m}^i \in \tilde{M}$ such that $i > V E $
m^*	$\tilde{w}^1, \tilde{w}^2, \dots, \tilde{w}^\delta, w^*$	

Table 4.1: Preference list for every man $m \in M$, and thus constructing $\mathcal{L}_M$. Here, $[X]$ denotes a fixed permutation of X .

performed in polynomial time.

4.2.2 The Parameter

Our current objective is to verify that t is indeed bounded by a function of k . For this purpose, we first observe that for all $i \in \{1, 2, \dots, \delta\}$, it holds that $p_{\tilde{m}^i}(\tilde{w}^i) = p_{\tilde{w}^i}(\tilde{m}^i) = 1$. Therefore, for all $\mu \in \text{SM}(\hat{\mathcal{I}})$ and $i \in \{1, 2, \dots, \delta\}$, we have that $\mu(\tilde{m}^i) = \tilde{w}^i$, else $(\tilde{m}^i, \tilde{w}^i)$ would have formed a blocking pair for μ in $\hat{\mathcal{I}}$.

Observation 4.2.1. *For all $\mu \in \text{SM}(\hat{\mathcal{I}})$ and $i \in \{1, 2, \dots, \delta\}$, it holds that $\mu(\tilde{m}^i) = \tilde{w}^i$.*

Now, note that $\mathcal{A}(m^*) = \tilde{W} \cup \{w^*\}$. Thus, by Observation 4.2.1, we have that for all $\mu \in \text{SM}(\hat{\mathcal{I}})$, either m^* is unmatched or $\mu(m^*) = w^*$. However, $\mathcal{A}(w^*) = \{m^*\}$, which implies that in the former case, (m^*, w^*) forms a blocking pair. Thus, we also have the following observation.

³Recall that we have defined an order on V .

W	Preference List
w_v^1	$m_v^2, [\{m_e^1 \in M_E : v \in e\}], [\{\tilde{m}^i \in \widetilde{M} : i \leq E - \deg_G(v)\}], m_v^1$ for all $w_v^1 \in W_V$
w_v^2	$m_v^1, [\{m_e^2 \in M_E : v \in e\}], [\{\tilde{m}^i \in \widetilde{M} : i \leq E - \deg_G(v)\}], m_v^2$ for all $w_v^2 \in W_V$
w_e^1	$m_e^2, \tilde{m}^1, \tilde{m}^2, \dots, \tilde{m}^{ V E }, m_e^1$ for all $w_e^1 \in W_E$
w_e^2	$m_e^1, \tilde{m}^1, \tilde{m}^2, \dots, \tilde{m}^{ V E }, m_e^2$ for all $w_e^2 \in W_E$
$\tilde{w}^i$	$\tilde{m}^i, m^*, m_{v_1}^1, m_{v_2}^1 \dots, m_{v_{ V }}^1, m_{v_1}^2, m_{v_2}^2 \dots, m_{v_{ V }}^2$ for $\tilde{w}^i \in \widetilde{W}$, $i \in \{1, 2\}$
$\tilde{w}^i$	$\tilde{m}^i, m^*$ for all $\tilde{w}^i \in \widetilde{W}$ such that $i > 2$
w^*	m^*

Table 4.2: Preference list for every woman $w \in W$, and thus constructing $\mathcal{L}_W$. For any set X , $[X]$ denotes a fixed permutation of X . Here, $\deg_G(v)$ denotes the degree of the vertex v in G .

Observation 4.2.2. *For all $\mu \in SM(\widehat{\mathcal{I}})$, it holds that $\mu(m^*) = w^*$.*

Let us proceed by identifying the man-optimal μ_M and the woman-optimal μ_W stable matchings. For this purpose, we first define a matching μ'_M as follows.

- For all $m_v^i \in M_V$: $\mu'_M(m_v^i) = w_v^i$.
- For all $m_e^i \in M_E$: $\mu'_M(m_e^i) = w_e^i$.
- For all $\tilde{m}^i \in \widetilde{M}$: $\mu'_M(\tilde{m}^i) = \tilde{w}^i$.
- $\mu'_M(m^*) = w^*$.

Lemma 4.2.2. *It holds that $\mu_M = \mu'_M$.*

Proof. Since for all $i \in \{1, 2, \dots, \delta\}$, it holds that $\mu'_M(\tilde{m}^i) = \tilde{w}^i$ and $p_{\tilde{m}^i}(\tilde{w}^i) = p_{\tilde{w}^i}(\tilde{m}^i) = 1$, we have that there cannot exist a blocking pair with at least one person from $\widetilde{M} \cup \widetilde{W}$. Now, notice that for every $m \in M$, including m^* , the woman most preferred by m who is outside $\widetilde{W}$ is also the one with whom it is matched. Therefore, there cannot exist any blocking pair for μ'_M , and by Observation 4.2.1, we further conclude that indeed $\mu_M = \mu'_M$. $\diamond$

Now, we define a matching μ'_W as follows.

- For all $w_v^1 \in W_V$: $\mu'_W(w_v^1) = m_v^2$, and for all $w_v^2 \in W_V$: $\mu'_W(w_v^2) = m_v^1$.
- For all $w_e^1 \in W_E$: $\mu'_W(w_e^1) = m_e^2$, and for all $w_e^2 \in W_E$: $\mu'_W(w_e^2) = m_e^1$.
- For all $\tilde{w}^i \in \widetilde{W}$: $\mu'_W(\tilde{w}^i) = \tilde{m}^i$.
- $\mu'_W(w^*) = m^*$.

Lemma 4.2.3. *It holds that $\mu_W = \mu'_W$.*

Proof. In the matching μ'_W , every woman is matched with the man she prefers the most. Thus, it is immediate that $\mu_W = \mu'_W$. $\diamond$

As a corollary to Lemmata [4.2.2](#) and [4.2.3](#), we obtain the following result.

Corollary 4.2.1. $O_M = |M| + \delta$ and $O_W = |W|$.

Proof. First, note that

$$\begin{aligned} O_M &= \sum_{(m,w) \in \mu_M} p_m(w) \\ &= p_{m^*}(\mu_M(m^*)) + \sum_{m \in M \setminus \{m^*\}} p_m(\mu_M(m)) = (\delta + 1) + (|M| - 1) = |M| + \delta. \end{aligned}$$

Second, note that

$$O_W = \sum_{(m,w) \in \mu_W} p_w(m) = \sum_{w \in W} p_w(\mu_W(w)) = |W|.$$

$\diamond$

We are now ready to bound t .

Lemma 4.2.4. *The parameter t associated with $\widehat{\mathcal{I}}$ is equal to $6(k + \frac{k(k-1)}{2})$.*

Proof. By the definition of t , we have that

$$\begin{aligned} t &= \widehat{k} - \max\{O_M, O_W\} \\ &= |M| + \delta + 6(k + \frac{k(k-1)}{2}) - \max\{|M| + \delta, |W|\} = 6(k + \frac{k(k-1)}{2}). \end{aligned}$$

◇

4.2.3 Correctness

First, from Lemma 4.2.2 and Proposition 4.1.2 we derive the following useful observation.

Observation 4.2.3. *Every $\mu \in SM(\widehat{\mathcal{I}})$ matches all people in $M \cup W$.*

Next, we proceed to state the first direction necessary to conclude that the input instance $\mathcal{I}$ of CLIQUE and our instance $\widehat{\mathcal{I}}$ of ABOVE-MAX BSM are equivalent.

Lemma 4.2.5. *If $\mathcal{I}$ is a YES-instance, then $\widehat{\mathcal{I}}$ is a YES-instance.*

Proof. Suppose that $\mathcal{I}$ is a YES-instance, and let U be the vertex set of a clique on k vertices in G . We denote $M_V^U = \{m_v^i \in M_V : v \in U\}$ and $M_E^U = \{m_{\{u,v\}}^i \in M_E : u, v \in U\}$. Then, we define a matching μ as follows.

- For all $m_v^i \in M_V$:
 - If $m_v^i \in M_V^U$: $\mu(m_v^i) = w_v^{3-i}$.
 - Else: $\mu(m_v^i) = w_v^i$.
- For all $m_e^i \in M_E$:
 - If $m_e^i \in M_E^U$: $\mu(m_e^i) = w_e^{3-i}$.
 - Else: $\mu(m_e^i) = w_e^i$.

- For all $\tilde{m}^i \in \widetilde{M}$: $\mu(\tilde{m}^i) = \tilde{w}^i$.
- $\mu(m^*) = w^*$.

We claim that $\mu \in SM(\widehat{\mathcal{I}})$ and $\text{balance}(\mu) \leq \widehat{k}$, which would imply that $\widehat{\mathcal{I}}$ is a YES-instance. To this end, we first show that $\mu \in SM(\widehat{\mathcal{I}})$. Since for all $i \in \{1, 2, \dots, \delta\}$, it holds that $\mu(\tilde{m}^i) = \tilde{w}^i$ and $p_{\tilde{m}^i}(\tilde{w}^i) = p_{\tilde{w}^i}(\tilde{m}^i) = 1$, we have that there cannot exist a blocking pair with at least one person from $\widetilde{M} \cup \widetilde{W}$. Thus, there can also not be a blocking pair with any person from $\{m^*, w^*\}$.

On the one hand, notice that for every $m \in (M_V \setminus M_V^U) \cup (M_E \setminus M_E^U) \cup \{m^*\}$, the woman most preferred by m who is outside $\widetilde{W}$ is also the one with whom it is matched. Thus, no man in $(M_V \setminus M_V^U) \cup (M_E \setminus M_E^U) \cup \{m^*\}$ can belong to a blocking pair. Moreover, the set of acceptable partners of any woman in W_E matched to a man in $M_E \setminus M_E^U$ is a subset of $\widetilde{M} \cup (M_E \setminus M_E^U)$, and therefore such a woman cannot belong to a blocking pair. On the other hand, let W' denote the set of every woman that is matched to a man $m \in M_V^U \cup M_E^U$. Then, for every $w \in W'$, the man most preferred by w is also the one with whom she is matched. Therefore, no woman in W' can belong to a blocking pair. Hence, we also conclude that no woman in W_E can belong to a blocking pair.

Thus, if there exists a blocking pair, it must consist of a man $m \in M_V^U \cup M_E^U$ and a woman $w \in W_V \setminus W'$. Suppose, by way of contradiction, that there exists such a blocking pair (m, w) . First, let us assume that $m = m_v^i \in M_V^U$. In this case, since apart from w_v^i , all women in $\mathcal{A}(m_v^i)$ belong to $\widetilde{W} \cup \{\mu(m_v^i)\}$, we deduce that $w = w_v^i$. However, w_v^i prefers $\mu(w_v^i)$ over m_v^i , and thus we reach a contradiction. Next, we assume that $m = m_{\{u,v\}}^i \in M_E^U$. In this case, it must hold that w is either w_v^i or w_u^i . Without loss of generality, we assume that $w = w_v^i$. However, since $m_{\{u,v\}}^i \in M_E^U$, we have that $v \in U$. Therefore, $\mu(w_v^i) = m_v^{3-i}$. Since w_v^i prefers m_v^{3-i} over $m_{\{u,v\}}^i$, we reach a contradiction.

It remains to prove that $\text{balance}(\mu) \leq \widehat{k}$. To this end, we need to show that

$$\max\left\{\sum_{(m,w) \in \mu} p_m(w), \sum_{(m,w) \in \mu} p_w(m)\right\} \leq |M| + \delta + 6\left(k + \frac{k(k-1)}{2}\right).$$

First, note that

$$\begin{aligned} \sum_{(m,w) \in \mu} p_m(w) &= \sum_{m \in M_V^U} p_m(\mu(m)) + \sum_{m \in M_V \setminus M_V^U} p_m(\mu(m)) + \sum_{m \in M_E^U} p_m(\mu(m)) \\ &\quad + \sum_{m \in M_E \setminus M_E^U} p_m(\mu(m)) + \sum_{m \in \widetilde{M}} p_m(\mu(m)) + p_{m^*}(\mu(m^*)) \\ &= 4|M_V^U| + |M_V \setminus M_V^U| + 4|M_E^U| + |M_E \setminus M_E^U| + |\widetilde{M}| + \delta + 1 \\ &= |M| + \delta + 3(|M_V^U| + |M_E^U|) = |M| + \delta + 6\left(k + \frac{k(k-1)}{2}\right). \end{aligned}$$

Second, note that

$$\begin{aligned} \sum_{(m,w) \in \mu} p_w(m) &= \sum_{m \in M_V^U} p_{\mu(m)}(m) + \sum_{m \in M_V \setminus M_V^U} p_{\mu(m)}(m) + \sum_{m \in M_E^U} p_{\mu(m)}(m) \\ &\quad + \sum_{m \in M_E \setminus M_E^U} p_{\mu(m)}(m) + \sum_{m \in \widetilde{M}} p_{\mu(m)}(m) + p_{\mu(m^*)}(m^*) \\ &= |M_V^U| + |M_V \setminus M_V^U|(|E| + 2) + |M_E^U| \\ &\quad + |M_E \setminus M_E^U|(|V||E| + 2) + |\widetilde{M}| + 1 \\ &= |M| + 2(|V| - k)(|E| + 1) + 2\left(|E| - \frac{k(k-1)}{2}\right)(|V||E| + 1) \\ &= |M| + \delta + 6\left(k + \frac{k(k-1)}{2}\right). \end{aligned}$$

This concludes the proof of the lemma. $\diamond$

We now turn to prove the second direction.

Lemma 4.2.6. *If $\widehat{\mathcal{I}}$ is a YES-instance, then $\mathcal{I}$ is a YES-instance.*

Proof. Suppose that $\widehat{\mathcal{I}}$ is a YES-instance, and let μ be a stable matching such that

$\text{balance}(\mu) \leq \widehat{k}$. By Observations [4.2.1](#) and [4.2.2](#), it holds that

- For all $i \in \{1, 2, \dots, \delta\}$: $\mu(\widetilde{m}^i) = \widetilde{w}^i$.
- $\mu(m^*) = w^*$.

Thus, since Observation [4.2.3](#) implies that all vertices in M_V should be matched by μ , we deduce that

- For all $v \in V$: Either both $\mu(m_v^1) = w_v^1$ and $\mu(m_v^2) = w_v^2$ or both $\mu(m_v^2) = w_v^1$ and $\mu(m_v^1) = w_v^2$.

Let U denote the set of every $v \in V$ such that $\mu(m_v^2) = w_v^1$ and $\mu(m_v^1) = w_v^2$. Moreover, denote $M_V^U = \{m_v^i \in M_V : v \in U\}$. By the item above, and since all vertices in M_E should also be matched by μ , we further deduce that

- For all $e \in E$: Either both $\mu(m_e^1) = w_e^1$ and $\mu(m_e^2) = w_e^2$ or both $\mu(m_e^2) = w_e^1$ and $\mu(m_e^1) = w_e^2$.

Let S denote the set of every $e \in E$ such that $\mu(m_e^2) = w_e^1$ and $\mu(m_e^1) = w_e^2$. Moreover, denote $M_E^S = \{m_e^i \in M_E : e \in S\}$. If there existed $\{u, v\} \in S$ such that $u \notin U$, then $(m_{\{u,v\}}^1, w_u^1)$ would have formed a blocking pair, which contradicts the fact that μ is a stable matching. Thus, we have that the set of endpoints of the edges in S is a subset of U .

We claim that $|U| = k$ and that U is the vertex set of a clique in G , which would imply that $\mathcal{I}$ is a YES-instance. Since we have argued that the set of endpoints of the edges in S is a subset of U , it is sufficient to show that $|U| \leq k$ and $|S| \geq \frac{k(k-1)}{2}$ (note that $|S| \geq \frac{k(k-1)}{2}$ implies that $|U| \geq k$), as this would imply that U is indeed the vertex set of a clique on k vertices in G . First, since $\text{balance}(\mu) \leq \widehat{k}$, we have

that $\sum_{(m,w) \in \mu} p_m(w) \leq |M| + \delta + 6(k + \frac{k(k-1)}{2})$. Now, note that

$$\begin{aligned}
\sum_{(m,w) \in \mu} p_m(w) &= \sum_{m \in M_V^U} p_m(\mu(m)) + \sum_{m \in M_V \setminus M_V^U} p_m(\mu(m)) + \sum_{m \in M_E^S} p_m(\mu(m)) \\
&\quad + \sum_{m \in M_E \setminus M_E^S} p_m(\mu(m)) + \sum_{m \in \widetilde{M}} p_m(\mu(m)) + p_{m^*}(\mu(m^*)) \\
&= 4|M_V^U| + |M_V \setminus M_V^U| + 4|M_E^S| + |M_E \setminus M_E^S| + |\widetilde{M}| + \delta + 1 \\
&= |M| + \delta + 6(|U| + |S|).
\end{aligned}$$

Thus, we deduce that $|U| + |S| \leq k + \frac{k(k-1)}{2}$. Now, observe that since $\text{balance}(\mu) \leq \widehat{k}$, we also have that $\sum_{(m,w) \in \mu} p_w(m) \leq |M| + \delta + 6(k + \frac{k(k-1)}{2})$. Here, on the one hand we note that

$$\begin{aligned}
\sum_{(m,w) \in \mu} p_w(m) &= \sum_{m \in M_V^U} p_{\mu(m)}(m) + \sum_{m \in M_V \setminus M_V^U} p_{\mu(m)}(m) + \sum_{m \in M_E^S} p_{\mu(m)}(m) \\
&\quad + \sum_{m \in M_E \setminus M_E^S} p_{\mu(m)}(m) + \sum_{m \in \widetilde{M}} p_{\mu(m)}(m) + p_{\mu(m^*)}(m^*) \\
&= |M_V^U| + |M_V \setminus M_V^U|(|E| + 2) + |M_E^S| \\
&\quad + |M_E \setminus M_E^S|(|V||E| + 2) + |\widetilde{M}| + 1 \\
&= |M| + 2(|V| - |U|)(|E| + 1) + 2(|E| - |S|)(|V||E| + 1) \\
&= |M| + 2(|V| + |E| + |V||E| + |V||E|^2) \\
&\quad - 2|U|(|E| + 1) - 2|S|(|V||E| + 1).
\end{aligned}$$

On the other hand, we note that

$$\begin{aligned}
\widehat{k} &= |M| + \delta + 6(k + \frac{k(k-1)}{2}) \\
&= |M| + 2(|V| + |E| + |V||E| + |V||E|^2) - k(4 + 4k + 2|E| + (k-1)|V||E|) \\
&\quad + 6(k + \frac{k(k-1)}{2}) \\
&= |M| + 2(|V| + |E| + |V||E| + |V||E|^2) - 2k(|E| + 1) - k(k-1)(|V||E| + 1).
\end{aligned}$$

Thus, we have that

$$|U|(|E| + 1) + |S|(|V||E| + 1) \geq k(|E| + 1) + \frac{k(k-1)}{2}(|V||E| + 1)$$

Recall that we have also shown that $|U| + |S| \leq k + \frac{k(k-1)}{2}$. Thus, since $|U| \leq k + \frac{k(k-1)}{2} - |S|$, to satisfy the above equation it must hold that $|S| \geq \frac{k(k-1)}{2}$. Since $|U| + |S| \leq k + \frac{k(k-1)}{2}$, we deduce that $|U| \leq k$. This, as we have argued earlier, finished the proof. $\diamond$

This concludes the proof of Theorem [7](#).

4.3 Kernel

In this section, we design a kernelization algorithm for ABOVE-MIN BSM. More precisely, we prove Theorem [8](#).

4.3.1 Functional Balanced Stable Marriage

To prove Theorem [8](#), we first prove the following result for the ABOVE-MIN FBSM problem.

Lemma 4.3.1. *ABOVE-MIN FBSM admits a kernel with at most $2t$ men, at most $2t$ women, and such that the image of the preference function of each person is a subset of $\{1, 2, \dots, t+1\}$.*

To obtain the desired kernelization algorithm, we execute the following plan if $t \geq 0$. (If $t < 0$, then $k < \min\{O_M, O_W\} \leq \max\{O_M, O_W\}$. We prove in Lemma [4.3.2](#) that it is a No-instance.)

1. **Cleaning Prefixes and Suffixes.** Simplify the preference functions by “cleaning” suffixes and thereby also “cleaning” prefixes.

In this step, we remove from a person a ’s set of acceptable partners, $\mathcal{A}(a)$, an individual who will never be matched to the former in *any* stable matching of the given instance. Intuitively speaking, we remove people who are either in the *prefix* or the *suffix* of from a ’s preference list. A prefix for a man (woman) consists of women (men) who are ranked better than his (her) man (woman) optimal stable partner. Analogously, a suffix for a man (woman) consists of women (men) who are ranked lower than his (her) woman (man) optimal stable partner. For a man m , we execute this step by removing from $\mathcal{A}(m)$ all those women whose f_m -value is either lower than $f_m(\mu_M(m))$, the function value of m ’s man-optimal stable matching partner, or higher than $f_m(\mu_W(m))$, the function value of m ’s woman optimal stable matching partner. Symmetrically, for woman w , we reduce $\mathcal{A}(w)$ by removing all those men whose f_w -value is lower than $f_w(\mu_W(w))$ or higher than $f_w(\mu_M(w))$. Since the domain of a preference function f_a is the set of acceptable partners $\mathcal{A}(a)$, this step restricts the preference function itself.

2. **Perfect Matching.** Zoom into the set of people matched by every stable matching.

When the preferences are complete, that is, each person is an acceptable partner for every individual on the other side, then the stable matchings are perfect. Otherwise, we know that by the Rural Hospital theorem [59], if a person (man/woman) is unmatched in the man optimal stable matching, then that person will be unmatched in every stable matching in the instance. Consequently, we can remove such people the instance and restrict the preference functions appropriately. This ensures that in the resulting instance, every stable matching is a perfect matching.

3. **Overcoming Sadness.** Bound the number of “sad” people. A person is sad

if s/he is not a “happy” person, defined to be one who has the same partner in every stable matching. Thus, an individual a is sad if a ’s best stable matching partner is also not a ’s worst stable matching partner.

In this step, we take an accounting of the progress we have made so far in reducing the size of the instance, namely by reducing the number of men and the number of women. We show that after when the first two steps are not applicable, then the number of sad people in the instance convey useful properties about the instance. Specifically, if the number of sad women/ men in the instance is more than $2t$, then it is a NO-instance. Conversely, if there are no sad men (or women) in the instance, then it is a YES-instance if and only if the **balance()**-value of the man optimal stable matching is at most k .

4. Marrying Happy People. Remove “happy” people from the instance.

Note that a happy man is always matched to the same happy woman in every stable matching; such a pair is better known as a *fixed pair* in the stable matching literature. By removing a fixed pair, say (m_h, w_h) , from the instance, we are creating a new instance in which the set of stable matchings are in bijective correspondence with the set of stable matchings in the original instance. The balance values of each matchings, however, will goes down by a fixed and known quantity, $f_{m_h}(w_h)$ or $f_{w_h}(m_h)$. So when we remove stable pair from our instance, we transfer its contribution to the **balance()**-value of the original instance to the new instance. This ensures that the balance values of each of the matchings in the new instance is same as the value of the corresponding matching in the original instance.

5. Removing High Value Partners. Obtain “compact” preference functions by truncating “high-values”.

For each person, the goal is to remove acceptable partners who if matched to (in a stable matching) will inexorably raise the balance value of the resulting

matching to higher than k , irrespective of the other matching pairs. Thus, it follows that such a matching cannot be a yes certificate (the reason it is a YES-instance) for our instance. Moreover, if it is a NO-instance, then the balance value of every stable matching must be higher than k . Hence, removing a very high value stable matching from the instance (as long as there are others in the instance⁴) will not destroy the equivalence property of the resulting instance.

Specifically, we know that if for a man m and woman w , who are each others acceptable partner, we either have $f_m(w) > f_m(\mu_M(m)) + (k - O_M)$ or $f_w(m) > f_w(\mu_W(w)) + (k - O_W)$, then any stable matching that contains the pair (m, w) will have **balance**()-value greater than k . Hence, it is safe to remove w from the set $\mathcal{A}(m)$ and m from $\mathcal{A}(w)$.

Since we cleaned the prefixes, it may be that for some man m and woman w , the pre-images $f_m^{-1}(1)$ and $f_w^{-1}(1)$ may not be m 's man optimal stable matching partner and w 's woman optimal stable partner, respectively. In this scenario we perform the following step.

6. Shrinking Gaps. Shrink some of the gaps created by previous steps.

We perform this step by decreasing $f_m(w')$ by 1 for every woman $w' \in \mathcal{A}(m)$. Similarly, for every $m' \in \mathcal{A}(w)$, we decrease $f_w(m')$ by 1. Since, we are reducing the preference function value of m and w 's best stable matching partner, and by Step **1** we know that both of them will be matched in every stable matching, it is necessary that we reduce the value of k (the target **balance**()-value) by 1.

This would complete the construction of the kernel for ABOVE-MIN FBSM.

One way to see the intuition behind the safeness of each of our reduction rules is to see their effect on the stable matching lattice of the original instance.

⁴Since our instance only contains sad men and women, it follows readily that there are more than one stable matchings

Effect on the stable matching lattice: Let $\mathcal{I}$ denote the initial instance. The applications of Steps 1-3 leaves the stable matching lattice of $\mathcal{I}$ intact. Step 4 creates an instance, denoted by $\mathcal{J}$, whose stable matching lattice consists of submatchings of the original instance such that the stable matchings in the two instances are in bijective correspondence. Specifically, each stable matching μ in $\mathcal{I}$ gives rises to a stable matching μ' in $\mathcal{J}$ such that $\mu \setminus \mu'$ is the set of fixed pairs in $\mathcal{I}$. Step 5 actually results in a lattice that is a subset of the lattice of the initial instance. Finally it is easy to see to that Step 6, which does not remove or alter partnerships, and thus does not have any affect on the stable matching lattice of the initial instance.

Obtaining a kernel for ABOVE-MIN BSM: Using the kernel for ABOVE-MIN FBSM, we obtain a kernel for ABOVE-MIN BSM by interpreting the preference functions as preference lists. The main roadblock towards this is that the preference functions may have “gaps” in the function values, i.e the function f_m (f_w) for some man m (woman w) may not be surjective in the range $[f_m(\mu_M(m)), f_m(\mu_W(m))]$ ($[f_w(\mu_W(w)), f_w(\mu_M(w))]$). Notably, while creating the preference list for m , we cannot have gaps in the preference list of m .

7. Maximum function value is bounded. Since Step 5 is not applicable, for any pair (m, w) who are each others acceptable partner, we have

$$f_m(w) \leq f_m(\mu_M(m)) + (k - O_M) \leq f_m(\mu_M(m)) + k - \min\{O_M, O_W\} = f_m(\mu_M(m)) + t$$

Symmetrically, we have $f_w(m) \leq f_w(\mu_M(m)) + t$.

Since Step 6 is not applicable, it follows that $f_m(\mu_M(m)) = 1$ and $f_w(\mu_W(w)) =$

1. Hence, we have ensured that $t + 1$ is the maximum value any preference function can take.

8. Filling the “gaps” in the preference function. For a man m , consider

its preference function values in increasing order: $f_m(x_1), f_m(x_2), \dots, f_m(x_{t'})$, where $t' \leq t + 1$, due to Step [7](#)

If any of these two adjacent values are not actually consecutive, then we have a “gap” in our preference function. For some $i \in [t']$ let $s_i = f_m(x_{i+1}) - f_m(x_i) > 1$, then we fill the “gap” between $f_m(x_i)$ and $f_m(x_{i+1})$ by s dummy women $y_1, \dots, y_{s_i}$, where $s_i \leq t' - 1 \leq t$. We do this for every gap that exists in the preference function of m . Since the maximum value of $f_m(\cdot)$ is at most $t + 1$, thus we require no more than t dummy women to fill the gaps in m ’s preference function. Note that the same set of t dummy women can be used to fill the gaps in each man’s preference function. Formally, this is done by defining another preference function g_m which extends the domain of f_m such that there are no gaps in g_m . The preference function g_m can be interpreted as a preference list in a straightforward fashion: $[x_1, \dots, x_i, y_1, \dots, y_s, x_{i+1}, \dots, x_{t'}]$. Doing this for every agent man/woman whose preference function has ”gaps” leads to a kernel for ABOVE-MIN BSM. Thus, overall the kernel for ABOVE-MIN BSM has t additional men and women each.

In our upcoming discussion, we elaborate on each of these reduction rules formally. In what follows, we let $\mathcal{I}$ denote our current instance of ABOVE-MIN FBSM. Initially, this instance is the input instance, but as the execution of our algorithm progresses, the instance is modified. The reduction rules that we present are applied *exhaustively* in the order of their presentation. In other words, at each point of time, the first rule whose condition is true is the one that we apply next. In particular, the execution terminates once the value of t drops below 0, as implied by the following rule.

It is worth mentioning that the phenomenon captured by some of our initial Reduction Rules such as [4.3.2](#) and [4.3.3](#) have previously been studied (see [71](#) Theorem 1.2.5 and 1.4.2), albeit in a different context or language. The safeness of those rules, therefore, may well be a foregone conclusion for some readers. However,

we present all the proofs here for the sake of completeness. From Proposition [4.1.1](#) we can infer that each of our reduction rules can indeed be implemented in time polynomial in number of men and women.

Reduction Rule 4.3.1. *If $k < \max\{O_M, O_W\}$, then return NO.*

Lemma 4.3.2. *Reduction Rule [4.3.1](#) is safe.*

Proof. For every $\mu \in \text{SM}$, it holds that $\text{balance}(\mu) \geq \max\{O_M, O_W\}$. Thus, if $k < \max\{O_M, O_W\}$, then every $\mu \in \text{SM}$ satisfies $\text{balance}(\mu) > k$. In this case, we conclude that $\text{Bal} > k$, and therefore $\mathcal{I}$ is a NO-instance. $\diamond$

Note that if $t < 0$, then $k < \min\{O_M, O_W\} \leq \max\{O_M, O_W\}$.

Cleaning Prefixes and Suffixes. We begin by modifying the images of the preference functions. We remark that it is *necessary* to perform this step first as otherwise the following steps would not be correct. To clean prefixes while ensuring *both* safeness and that the parameter t does not increase, we would actually need to clean *suffixes* first. Formally, we define suffixes as follows.

Definition 4.3.1. *Let (m, w) denote an acceptable pair. If m is matched by μ_W and $f_m(w) > f_m(\mu_W(m))$, then we say that w belongs to the suffix of m . Similarly, if w is matched by μ_M and $f_w(m) > f_w(\mu_M(w))$, then we say that m belongs to the suffix of w .*

By Proposition [4.1.3](#), we have the following observation.

Observation 4.3.1. *Let (m, w) denote an acceptable pair such that one of its members belongs to the suffix of the other member. Then, there is no $\mu \in \text{SM}(\mathcal{I})$ that matches m with w .*

For every person a , let $\text{worst}(a)$ be the person in $\mathcal{A}(a)$ to whom f_a assigns its worst preference value. More precisely, $\text{worst}(a) = \operatorname{argmax}_{b \in \mathcal{A}(a)} f_a(b)$. We will now clean suffixes.

Reduction Rule 4.3.2. *If there exists a person a such that $\text{worst}(a)$ belongs to the suffix of a , then define the preference functions as follows.*

- $f'_a = f_a|_{\mathcal{A}(a) \setminus \{\text{worst}(a)\}}$ and $f'_{\text{worst}(a)} = f_{\text{worst}(a)}|_{\mathcal{A}(\text{worst}(a)) \setminus \{a\}}$.
- For all $b \in M \cup W \setminus \{a, \text{worst}(a)\}$: $f'_b = f_b$

The new instance is $\mathcal{J} = (M, W, \{f'_{m'}\}_{m' \in M}, \{f'_{w'}\}_{w' \in W}, k)$.

Lemma 4.3.3. *Reduction Rule 4.3.2 is safe, and $t(\mathcal{I}) = t(\mathcal{J})$.*

Proof. By the definition of the new preference functions, we have that for every $\mu \in \text{SM}(\mathcal{I}) \cap \text{SM}(\mathcal{J})$, it holds that $\sum_{(m,w) \in \mu} f_m(w) = \sum_{(m,w) \in \mu} f'_m(w)$ and $\sum_{(m,w) \in \mu} f_w(m) = \sum_{(m,w) \in \mu} f'_w(m)$. In particular, this means that to conclude that $\text{Bal}(\mathcal{I}) = \text{Bal}(\mathcal{J})$ (which implies safeness) as well as that $O_M(\mathcal{I}) = O_M(\mathcal{J})$ and $O_W(\mathcal{I}) = O_W(\mathcal{J})$ (which implies that $t(\mathcal{I}) = t(\mathcal{J})$), it is sufficient to show that $\text{SM}(\mathcal{I}) = \text{SM}(\mathcal{J})$. For this purpose, first consider some $\mu \in \text{SM}(\mathcal{I})$. By Observation 4.3.1, it holds that $(a, \text{worst}(a)) \notin \mu$. Hence, μ is a matching in $\mathcal{J}$. Moreover, if μ has a blocking pair in $\mathcal{J}$, then by the definition of the new preference functions, it is also a blocking pair in $\mathcal{I}$. Since μ is stable in $\mathcal{I}$, we have that $\mu \in \text{SM}(\mathcal{J})$.

In the second direction, consider some $\mu \in \text{SM}(\mathcal{J})$. Then, it is clear that μ is a matching in $\mathcal{I}$. Moreover, if μ has a blocking pair (m, w) in $\mathcal{I}$ that is not $(a, \text{worst}(a))$, then (m, w) is an acceptable pair in $\mathcal{J}$, and therefore by the definition of the new preference functions, we have that (m, w) is also a blocking pair in $\mathcal{J}$. Hence, since μ is stable in $\mathcal{J}$, the only pair that can block μ in $\mathcal{I}$ is $(a, \text{worst}(a))$. Thus, to show that $\mu \in \text{SM}(\mathcal{I})$, it remains to prove that $(a, \text{worst}(a))$ cannot block μ in $\mathcal{I}$. Suppose,

by way of contradiction, that $(a, \text{worst}(a))$ blocks μ in $\mathcal{I}$. In particular, this means that $f_a(\text{worst}(a)) < f_a(\mu(a))$. However, this contradicts the definition of $\text{worst}(a)$. $\diamond$

By cleaning suffixes, we actually also accomplish the objective of cleaning prefixes, which are defined as follows.

Definition 4.3.2. *Let (m, w) denote an acceptable pair. If m is matched by μ_M and $f_m(w) < f_m(\mu_M(m))$, then we say that w belongs to the prefix of m . Similarly, if w is matched by μ_W and $f_w(m) < f_w(\mu_W(w))$, then we say that m belongs to the prefix of w .*

Lemma 4.3.4. *Let $\mathcal{I}$ be an instance of ABOVE-MIN FBSM on which Reduction Rules [4.3.1](#) to [4.3.2](#) have been exhaustively applied. Then, there does not exist an acceptable pair (m, w) such that one of its members belongs to the prefix of the other one.*

Proof. Suppose, by way of contradiction, that there exists an acceptable pair (m, w) such that one of its members belongs to the prefix of the other one. Without loss of generality, assume that w belongs to the prefix of m . Then, $f_m(w) < f_m(\mu_M(m))$. Since μ_M is a stable matching, it cannot be blocked by (m, w) , which means that w is matched by μ_M and $f_w(\mu_M(w)) < f_w(m)$. Thus, we have that m belongs to the suffix of w , which contradicts the assumption that Reduction Rule [4.3.2](#) was applied exhaustively. $\diamond$

Corollary 4.3.1. *Let $\mathcal{I}$ be an instance of ABOVE-MIN FBSM on which Reduction Rules [4.3.1](#) to [4.3.2](#) have been exhaustively applied. Then, for every acceptable pair (m, w) in $\mathcal{I}$ where m and w are matched (not necessarily to each other) by both μ_M and μ_W , it holds that $f_m(\mu_M(m)) \leq f_m(w) \leq f_m(\mu_W(m))$ and $f_w(\mu_W(w)) \leq f_w(m) \leq f_w(\mu_M(w))$.*

Perfect Matching. Having Corollary 4.3.1 at hand, we are able to provide a simple rule that allows us to assume that every solution matches all people.

Reduction Rule 4.3.3. *If there exists a person unmatched by μ_M , then let M' and W' denote the subsets of men and women, respectively, who are matched by μ_M . For each $a \in M' \cup W'$, denote $\mathcal{A}'(a) = \mathcal{A}(a) \cap (M' \cup W')$, and define $f'_a = f_a|_{\mathcal{A}'(a)}$. The new instance is $\mathcal{J} = (M', W', \{f'_m\}_{m \in M'}, \{f'_w\}_{w \in W'}, k)$.*

Note that if a person is unmatched in μ_M , then, due to Proposition 4.1.2, s/he is unmatched in every stable matching in $\mathcal{I}$. To prove the safeness of this rule, we first prove the following lemma.

Lemma 4.3.5. *Let $\mathcal{I}$ be an instance of ABOVE-MIN FBSM on which Reduction Rules 4.3.1 to 4.3.2 have been exhaustively applied. Then, for every person a not matched by μ_M , it holds that $\mathcal{A}(a) = \emptyset$.*

Proof. Let a be a person not matched by μ_M . Then, by Proposition 4.1.2, it holds that a is not matched by any stable matching. We assume w.l.o.g. that a is a man, say m . First, note that $\mathcal{A}(m)$ cannot contain any woman w that is not matched by some stable matching, else (m, w) would have formed a blocking pair for that stable matching. Second, we claim that $\mathcal{A}(m)$ cannot contain a woman w that is matched by some stable matching. Suppose, by way of contradiction, that this claim is false. Then, by Proposition 4.1.2, it holds that $\mathcal{A}(m)$ contains a woman w that is matched by μ_M . We have that w prefers $\mu_M(w)$ over m , else (m, w) would have formed a blocking pair for μ_M , which is impossible as μ_M is a stable matching. However, this implies that m belongs to the suffix of w , which contradicts the supposition that Reduction Rule 4.3.2 has been exhaustively applied. We thus conclude that $\mathcal{A}(a) = \emptyset$. $\diamond$

Lemma 4.3.6. *Reduction Rule 4.3.3 is safe, and $t(\mathcal{I}) = t(\mathcal{J})$.*

Proof. By the definition of the new preference functions, we have that for every $\mu \in \text{SM}(\mathcal{I}) \cap \text{SM}(\mathcal{J})$, it holds that $\sum_{(m,w) \in \mu} f_m(w) = \sum_{(m,w) \in \mu} f'_m(w)$ and $\sum_{(m,w) \in \mu} f_w(m) = \sum_{(m,w) \in \mu} f'_w(m)$. To conclude that the lemma is correct, it is thus sufficient to argue that $\text{SM}(\mathcal{I}) = \text{SM}(\mathcal{J})$. For this purpose, first consider some $\mu \in \text{SM}(\mathcal{I})$. By Proposition 4.1.2, we have that μ is also a matching in $\mathcal{J}$.

Moreover, if μ has a blocking pair in $\mathcal{J}$, then by the definition of the new preference functions, it is also a blocking pair in $\mathcal{I}$. Since μ is stable in $\mathcal{I}$, we have that $\mu \in \text{SM}(\mathcal{J})$.

In the second direction, consider some $\mu \in \text{SM}(\mathcal{J})$. Then, it is clear that μ is a matching in $\mathcal{I}$. By Lemma 4.3.5, if μ has a blocking pair (m, w) in $\mathcal{I}$, then both $m \in M'$ and $w \in W'$. However, for such a blocking pair (m, w) , we have that (m, w) is an acceptable pair in $\mathcal{J}$, and therefore by the definition of the new preference functions, we have that (m, w) is also a blocking pair in $\mathcal{J}$. Hence, since μ is stable in $\mathcal{J}$, we conclude that μ is also stable in $\mathcal{I}$. $\diamond$

By Proposition 4.1.2, from now onwards, we have that for the given instance, any stable matching is a perfect matching. Due to this observation, we can denote $n = |M| = |W|$, and for any stable matching μ , we have the following equalities.

$$(I) \quad \sum_{(m,w) \in \mu} f_m(w) = \sum_{m \in M} f_m(\mu(m)); \quad \sum_{(m,w) \in \mu} f_w(m) = \sum_{w \in W} f_w(\mu(w)).$$

Overcoming Sadness. As every stable matching is a perfect matching, every person is matched by every stable matching, including the man-optimal and woman-optimal stable matchings. Thus, it is well defined to classify the people who do not have the same partner in the man-optimal and woman-optimal stable matchings as “sad”. That is,

Definition 4.3.3. A person $a \in M \cup W$ is sad if $\mu_M(a) \neq \mu_W(a)$.

We let M_S and W_S denote the sets of sad men and sad women, respectively. People who are not sad are termed *happy*. Accordingly, we let M_H and W_H denote the sets of happy men and happy women, respectively. Note that $M_S = \emptyset$ if and only if $W_S = \emptyset$. Moreover, note that by the definition of μ_M and μ_W , for a happy person a it holds that a and $\mu_M(a) = \mu_W(a)$ are matched to one another by every stable matching. Let us now bound the number of sad people in a YES-instance.

Reduction Rule 4.3.4. *If $|M_S| > 2t$ or $|W_S| > 2t$, then return NO.*

Lemma 4.3.7. *Reduction Rule 4.3.4 is safe.*

Proof. We only prove that if $|M_S| > 2t$, then $\mathcal{I}$ is a NO-instance, as the proof of the other case is symmetric to this one. Let us assume that $|M_S| > 2t$. Suppose, by way of contradiction, that $\mathcal{I}$ is a YES-instance. Then, there exists a stable matching μ such that $\text{balance}(\mu) \leq k$. Partition $M_S = M'_S \uplus M''_S$ as follows. Set M'_S to be the set of all m in M_S such that m is not the partner of $\mu(m)$ in μ_W .

$$M'_S = \{m \in M_S \mid f_{\mu(m)}(m) > f_{\mu(m)}(\mu_W(\mu(m)))\}.$$

Accordingly, set $M''_S = M_S \setminus M'_S$. Since $|M_S| > 2t$, at least one among $|M'_S|$ and $|M''_S|$ is (strictly) larger than t . Let us first handle the case where $|M'_S| > t$. Then,

$$\begin{aligned} \sum_{w \in W} f_w(\mu(w)) &= \sum_{\{w \mid \mu(w) \in M'_S\}} f_w(\mu(w)) + \sum_{\{w \mid \mu(w) \notin M'_S\}} f_w(\mu(w)) \\ &\geq \sum_{\{w \mid \mu(w) \in M'_S\}} [f_w(\mu_W(w)) + 1] + \sum_{\{w \mid \mu(w) \notin M'_S\}} f_w(\mu_W(w)) \\ &= \sum_{w \in W} f_w(\mu_W(w)) + \sum_{\{w \mid \mu(w) \in M'_S\}} 1 = O_W + |M'_S| \\ &> O_W + t \geq k. \end{aligned}$$

Here, the first inequality followed directly from the definition of M'_S . As we have

reached a contradiction, it must hold that $|M_S''| > t$. However, we now have that

$$\begin{aligned}
\sum_{m \in M} f_m(\mu(m)) &= \sum_{m \in M_S''} f_m(\mu(m)) + \sum_{m \notin M_S''} f_m(\mu(m)) \\
&\geq \sum_{m \in M_S''} [f_m(\mu_M(m)) + 1] + \sum_{m \notin M_S''} f_m(\mu_M(m)) \\
&= \sum_{m \in M} f_m(\mu_M(m)) + \sum_{\{m \mid m \in M_S''\}} 1 = O_M + |M_S''| \\
&> O_M + t \geq k.
\end{aligned}$$

Here, the first inequality followed from the definition of M_S'' . Indeed, for all $m \in M_S''$, we have that $f_{\mu(m)}(m) \leq f_{\mu(m)}(\mu_W(\mu(m)))$, else m would have belonged to M_S' . However, in this case we deduce that $m = \mu_W(\mu(m))$, and since $m \in M_S$, we have that $\mu(m) \neq \mu_M(m)$, which implies that $f_m(\mu(m)) \geq f_m(\mu_M(m)) + 1$. As we have again reached a contradiction, we conclude the proof. $\diamond$

Marrying Happy People. Towards the removal of happy people, we first need to handle the special case where there are no sad people. In this case, there is exactly one stable matching, which is the man-optimal stable matching (that is equal, in this case, to the woman-optimal stable matching). This immediately implies the safeness of the following rule.

Reduction Rule 4.3.5. *If $M_S = W_S = \emptyset$, then return YES if $\text{balance}(\mu_M) \leq k$ and NO otherwise.*

Observation 4.3.2. *Reduction Rule 4.3.5 is safe.*

We now turn to discard happy people. When we perform this operation, we need to ensure that the balance of the instance is preserved. This is because we do not know which side (men or women) attains the $\text{Bal}(\mathcal{I})$ value, hence we cannot reduce the quantity k by the dissatisfaction of the happy people on that side. Consequently, we need to ensure that $\text{Bal}(\mathcal{I}) = \text{Bal}(\mathcal{J})$, where $\mathcal{J}$ denotes the new instance resulting

from the removal of some happy people. Towards this, we let (m_h, w_h) denote a *happy pair*, which is simply a pair of a happy man and happy woman who are matched to each other in every stable matching⁵. Then, we redefine the preference functions in a manner that allows us to transfer the “contributions” of m_h and w_h from $\text{Bal}(\mathcal{I})$ to $\text{Bal}(\mathcal{J})$ via some sad man and woman. We remark that these sad people exist because Reduction Rule 4.3.5 does not apply. The details are as follows.

Reduction Rule 4.3.6. *If there exists a happy pair (m_h, w_h) , then proceed as follows. Select an arbitrary sad man m_s and an arbitrary sad woman w_s . Denote $M' = M \setminus \{m_h\}$ and $W' = W \setminus \{w_h\}$. For each person $a \in M' \cup W'$, the new preference function $f'_a : \mathcal{A}(a) \setminus \{m_h, w_h\} \rightarrow \mathbb{N}$ is defined as follows.*

- For each $w \in \mathcal{A}(m_s) \setminus \{w_h\}$: $f'_{m_s}(w) = f_{m_s}(w) + f_{m_h}(w_h)$.
- For each $m \in \mathcal{A}(w_s) \setminus \{m_h\}$: $f'_{w_s}(m) = f_{w_s}(m) + f_{w_h}(m_h)$.
- For each $w \in W' \setminus \{w_s\}$: $f'_w = f_w|_{M'}$, and for each $m \in M' \setminus \{m_s\}$: $f'_m = f_m|_{W'}$.

The new instance is $\mathcal{J} = (M', W', \{f'_m\}_{m \in M'}, \{f'_w\}_{w \in W'}, k)$.

The following lemma proves the forward direction of the safeness of Reduction Rule 4.3.6.

Lemma 4.3.8. *Let $\mu \in \text{SM}(\mathcal{I})$ and $\mathcal{J}$ be the instance produced by Reduction Rule 4.3.6. Then, $\mu' = \mu \setminus \{(m_h, w_h)\}$ is a stable matching in $\mathcal{J}$ such that $\text{balance}_{\mathcal{J}}(\mu') = \text{balance}_{\mathcal{I}}(\mu)$.*

Proof. We first show that $\mu' \in \text{SM}(\mathcal{J})$, i.e., μ' is stable in $\mathcal{J}$. Then, we show that $\text{balance}_{\mathcal{J}}(\mu') = \text{balance}_{\mathcal{I}}(\mu)$. By Reduction Rule 4.3.3, it holds that μ is a perfect matching in $\mathcal{I}$. Since (m_h, w_h) is a happy pair, it is clear that $(m_h, w_h) \in \mu$, and therefore μ' is a perfect matching in $\mathcal{J}$. Let $(m, w) \notin \mu'$ be some acceptable pair in

⁵Such a pair is often referred to as fixed pair in literature.

$\mathcal{J}$. Since $\mu \in \text{SM}(\mathcal{I})$ and it is a perfect matching, it holds that $f_m(w) > f_m(\mu(m))$ or $f_w(m) > f_w(\mu(w))$. Let us consider these two possibilities separately.

- Suppose that $f_m(w) > f_m(\mu(m))$. If $m \neq m_s$, then $f'_m(w) = f_m(w)$ and $f'_m(\mu'(m)) = f_m(\mu(m))$, and therefore $f'_m(w) > f'_m(\mu'(m))$. Else, $f'_m(w) = f_m(w) + f_{m_h}(w_h)$ and $f'_m(\mu'(m)) = f_m(\mu(m)) + f_{m_h}(w_h)$, and therefore again $f'_m(w) > f'_m(\mu'(m))$.
- Suppose that $f_w(m) > f_w(\mu(w))$. Analogously to the previous case, we get that $f'_w(m) > f'_w(\mu'(w))$.

Since the choice of (m, w) was arbitrary, we conclude that μ' does not have a blocking pair in $\mathcal{J}$, and therefore $\mu' \in \text{SM}(\mathcal{J})$. To show that $\text{balance}_{\mathcal{J}}(\mu') = \text{balance}_{\mathcal{I}}(\mu)$, note that

$$\begin{aligned}
\text{balance}_{\mathcal{J}}(\mu') &= \max\left\{\sum_{m \in M \setminus \{m_h\}} f'_m(\mu'(m)), \sum_{w \in W \setminus \{w_h\}} f'_w(\mu'(w))\right\} \\
&= \max\left\{f'_{m_s}(\mu'(m_s)) + \sum_{m \in M \setminus \{m_h, m_s\}} f'_m(\mu'(m)), \right. \\
&\quad \left. f'_{w_s}(\mu'(w_s)) + \sum_{w \in W \setminus \{w_h, w_s\}} f'_w(\mu'(w))\right\} \\
&= \max\left\{f_{m_s}(\mu(m_s)) + f_{m_h}(w_h) + \sum_{m \in M \setminus \{m_h, m_s\}} f_m(\mu(m)), \right. \\
&\quad \left. f_{w_s}(\mu(w_s)) + f_{w_h}(m_h) + \sum_{w \in W \setminus \{w_h, w_s\}} f_w(\mu(w))\right\} \\
&= \max\left\{\sum_{m \in M} f_m(\mu(m)), \sum_{w \in W} f_w(\mu(w))\right\} = \text{balance}_{\mathcal{I}}(\mu).
\end{aligned}$$

This concludes the proof. $\diamond$

The following observation helps to prove the reverse direction of the safeness of Reduction Rule [4.3.6](#).

Observation 4.3.3. *Let $\mathcal{I}$ be an instance of ABOVE-MIN FBSM on which Reduction Rules [4.3.1](#) to [4.3.2](#) have been exhaustively applied. Then, for every happy pair (m_h, w_h) , it holds that $\mathcal{A}(m_h) = \{w_h\}$ and $\mathcal{A}(w_h) = \{m_h\}$.*

Lemma 4.3.9. *Let $\mu' \in \text{SM}(\mathcal{J})$. Then, $\mu = \mu' \cup \{(m_h, w_h)\}$ is a stable matching in $\mathcal{I}$ such that $\text{balance}_{\mathcal{I}}(\mu) = \text{balance}_{\mathcal{J}}(\mu')$.*

Proof. We first show that $\mu \in \text{SM}(\mathcal{I})$. By Reduction Rule [4.3.3](#), it holds that μ' is a perfect matching in $\mathcal{J}$. Since (m_h, w_h) is a happy pair and by Observation [4.3.3](#), we have that μ is a perfect matching in $\mathcal{I}$ such that neither m_h nor w_h participate in any pair that blocks μ (if such a pair exists). Let $(m, w) \notin \mu'$ be some acceptable pair in $\mathcal{I}$ such that $m \neq m_h$ and $w \neq w_h$. Since $\mu' \in \text{SM}(\mathcal{J})$ and it is a perfect matching, it holds that $f'_m(w) > f'_m(\mu'(m))$ or $f'_w(m) > f'_w(\mu'(w))$. Let us consider these two possibilities separately.

- Suppose that $f'_m(w) > f'_m(\mu(m))$. If $m \neq m_s$, then $f_m(w) = f'_m(w)$ and $f_m(\mu(m)) = f'_m(\mu'(m))$, and therefore $f_m(w) > f_m(\mu(m))$. Else, $f_m(w) = f'_m(w) - f_{m_h}(w_h)$ and $f_m(\mu(m)) = f'_m(\mu'(m)) - f_{m_h}(w_h)$, and therefore again $f_m(w) > f_m(\mu(m))$.
- Suppose that $f'_w(m) > f'_w(\mu'(w))$. Analogously to the previous case, we get that $f_w(m) > f_w(\mu(w))$.

Since the choice of (m, w) was arbitrary, we conclude that μ does not have a blocking pair in $\mathcal{I}$, and therefore $\mu \in \text{SM}(\mathcal{I})$. To show that $\text{balance}_{\mathcal{I}}(\mu) = \text{balance}_{\mathcal{J}}(\mu')$, we follow the exact same argument as the one present in the proof of Lemma [4.3.8](#) $\diamond$

We now turn to justify the use of Reduction Rule [4.3.6](#). By Lemmata [4.3.8](#) and [4.3.9](#), it holds that $\text{Bal}(\mathcal{I}) = \text{Bal}(\mathcal{J})$, and that both $\text{balance}_{\mathcal{I}}(\mu_M(\mathcal{I})) =$

$\text{balance}_{\mathcal{J}}(\mu_M(\mathcal{J}))$ and $\text{balance}_{\mathcal{I}}(\mu_W(\mathcal{I})) = \text{balance}_{\mathcal{J}}(\mu_W(\mathcal{J}))$. As the argument k remained untouched, we have the following lemma.

Lemma 4.3.10. *Reduction Rule [4.3.6](#) is safe, and $t(\mathcal{I}) = t(\mathcal{J})$.*

Before we examine the preference functions more closely, we prove the following result.

Lemma 4.3.11. *Given an instance $\mathcal{I}$ of ABOVE-MIN FBSM, one can exhaustively apply Reduction Rules [4.3.1](#) to [4.3.6](#) in polynomial time to obtain an instance $\mathcal{J}$ such that $t(\mathcal{J}) \leq t(\mathcal{I})$. All people in $\mathcal{J}$ are sad and matched by every stable matching, and there exist at most $2t$ men and at most $2t$ women.*

Proof. Notice that each one of Reduction Rules [4.3.1](#) to [4.3.6](#) can be applied in polynomial time. By applying them sequentially and exhaustively, the process either terminates by outputting a trivial YES (NO)-instance or it shrinks the size of the instance in each step. Hence, it is clear that the instance $\mathcal{J}$ is obtained in polynomial time, and the claim that $t(\mathcal{J}) \leq t(\mathcal{I})$ follows from Lemmata [4.3.6](#) and [4.3.10](#). Due to Reduction Rules [4.3.3](#) and [4.3.6](#), we know that each person in $\mathcal{J}$ is sad and matched by every stable matching. Thus, due to Reduction Rule [4.3.4](#), we also know that there exist at most $2t$ men and at most $2t$ women. $\diamond$

Truncating High-Values. So far we have bounded the number of people. However, the images of the preference functions can contain integers that are not bounded by a function polynomial in the parameter. Thus, even though the number of people is upper bounded by $4t$, the total size of the instance can be huge. Hence, we need to process the images of the preference functions.

The intuition behind the Reduction Rule [4.3.7](#) (and thus the proof of Lemma [4.3.12](#)) can be described as follows. Recall that we have already modified preference

functions in Reduction Rule [4.3.2](#) so that they would not contain irrelevant information in the prefixes and suffixes. Our current goal is to truncate “high-values” of preference functions. Suppose that there exists a stable matching μ and a man m such that $f_m(\mu(m)) > t + f_m(\mu_M(m))$. That is, μ matches m to a woman whose functional value in f_m is larger than the functional value of the woman with whom m is matched by μ_M by at least t units. Then, $\text{balance}(\mu) \geq \sum_{m' \in M} f_{m'}(\mu(m')) > O_M + t \geq k$. Hence, irrespective of whether or not the current instance is a YES-instance, we know that μ is not a yes-certificate for our problem. We thus observe that we should delete all those acceptable pairs whose presence in any stable matching prevents its balance from being upper bounded by k . Formally, we have the following rule.

Reduction Rule 4.3.7. *If there exists an acceptable pair (m, w) such that $f_m(w) > (k - O_M) + f_m(\mu_M(m))$ or $f_w(m) > (k - O_W) + f_w(\mu_W(w))$, then define the preference functions as follows:*

- $f'_m = f_m|_{A(m) \setminus \{w\}}$ and $f'_w = f_w|_{A(w) \setminus \{m\}}$.
- For all $a \in M \cup W \setminus \{m, w\}$: $f'_a = f_a$.

The new instance is $\mathcal{J} = (M, W, \{f'_{m'}\}_{m' \in M}, \{f'_{w'}\}_{w' \in W}, k)$.

Lemma 4.3.12. *Reduction Rule [4.3.7](#) is safe, and $t(\mathcal{I}) \geq t(\mathcal{J})$.*

Proof. Without loss of generality, suppose that $f_m(w) > (k - O_M(\mathcal{I})) + f_m(\mu_M(m))$. Due to Reduction Rule [4.3.1](#), we have that $k - O_M(\mathcal{I}) \geq 0$, and therefore $f_m(w) > f_m(\mu_M(m))$, which implies that $w \neq \mu_M(m)$. That is, $(m, w) \notin \mu_M$. Since f_m and f'_m differ only at w and f_w and f'_w differ only at m , we have that $\mu_M(\mathcal{I})$ is also a matching in $\mathcal{J}$, and due to Corollary [4.3.1](#), we deduce that $\mu_M(\mathcal{I}) = \mu_M(\mathcal{J})$. First, we would like to show that $t(\mathcal{I}) \geq t(\mathcal{J})$. For this purpose, it is sufficient to show that $O_M(\mathcal{I}) \leq O_M(\mathcal{J})$ and $O_W(\mathcal{I}) \leq O_W(\mathcal{J})$. Since $\mu_M(\mathcal{I}) = \mu_M(\mathcal{J})$, it is clear that $O_M(\mathcal{I}) = O_M(\mathcal{J})$. By Reduction Rule [4.3.3](#), $\mu_M(\mathcal{I})$ matches all people in $\mathcal{I}$.

Thus, by Proposition [4.1.2](#) and since $\mu_M(\mathcal{I}) = \mu_M(\mathcal{J})$, we have that $\mu_W(\mathcal{J})$ matches all people in $\mathcal{J}$, and hence all people in $\mathcal{I}$. By Corollary [4.3.1](#), for any woman w' and the man m' most preferred by w' in $\mathcal{J}$, it holds that w' does not prefer m' over $\mu_W(w')$ in $\mathcal{I}$. Thus, by the definition of the preference functions, we have that $O_W(\mathcal{I}) \leq O_W(\mathcal{J})$.

To show that the rule is safe, we need to show that $\text{Bal}(\mathcal{I}) \leq k$ if and only if $\text{Bal}(\mathcal{J}) \leq k$. For this purpose, let us first suppose that $\text{Bal}(\mathcal{I}) \leq k$. Then, there exists $\mu \in \text{SM}(\mathcal{I})$ such that $\text{balance}_{\mathcal{I}}(\mu) \leq k$. Notice that if $\mu(m) = w$, then since $f_m(w) > (k - O_M) + f_m(\mu_M(m))$ and by the equation [\(II\)](#) in “Perfect Matching”, we have that

$$\begin{aligned} k &\geq \text{balance}_{\mathcal{I}}(\mu) \geq \sum_{m' \in M} f_{m'}(\mu(m')) \\ &> (k - O_M(\mathcal{I})) + \sum_{m' \in M} f_{m'}(\mu_M(m')) = (k - O_M(\mathcal{I})) + O_M(\mathcal{I}), \end{aligned}$$

which is a contradiction. Hence, $(m, w) \notin \mu$. Therefore, μ is a matching in $\mathcal{J}$. By the definition of the new preference functions, if μ has a blocking pair in $\mathcal{J}$, then this pair also blocks μ in $\mathcal{I}$. Since μ is stable in $\mathcal{I}$, we have that μ is also stable in $\mathcal{J}$. Now, by our definition of the new preference functions, we have that $\text{balance}_{\mathcal{I}}(\mu) = \text{balance}_{\mathcal{J}}(\mu)$. Since $\text{balance}_{\mathcal{I}}(\mu) \leq k$, we thus conclude that $\text{Bal}(\mathcal{J}) \leq k$.

In the second direction, suppose that $\text{Bal}(\mathcal{J}) \leq k$. Then, there exists $\mu \in \text{SM}(\mathcal{J})$ such that $\text{balance}_{\mathcal{J}}(\mu) \leq k$. Clearly, μ is also a matching in $\mathcal{I}$. Moreover, by the definition of the preference functions, every acceptable pair in $\mathcal{I}$ that is also present in $\mathcal{J}$ cannot block μ in $\mathcal{I}$, else it would have also blocked μ in $\mathcal{J}$. Thus, if μ has a blocking pair in $\mathcal{I}$, then this pair must be (m, w) . We claim that (m, w) cannot block μ in $\mathcal{I}$, which would imply that $\mu \in \text{SM}(\mathcal{I})$. Suppose, by way of contradiction, that this claim is not true. Recall that we have already proved that $\mu_M(\mathcal{I}) = \mu_M(\mathcal{J})$. Let us denote $\mu_M = \mu_M(\mathcal{I})$. We have that μ matches m , which implies that

$f_m(\mu(m)) > f_m(w)$. Since $f_m(w) > (k - O_M(\mathcal{I})) + f_m(\mu_M(m))$, we deduce that $f_m(\mu(m)) > (k - O_M(\mathcal{I})) + f_m(\mu_M(m))$. Furthermore, since $f'_m(\mu(m)) = f_m(\mu(m))$, $f'_m(\mu_M(m)) = f_m(\mu_M(m))$ and $O_M(\mathcal{I}) = O_M(\mathcal{J})$, we get that $f'_m(\mu(m)) > (k - O_M(\mathcal{J})) + f'_m(\mu_M(m))$. However, we then have that

$$\begin{aligned} k &\geq \text{balance}_{\mathcal{J}}(\mu) \geq \sum_{m' \in M} f'_{m'}(\mu(m')) \\ &> (k - O_M(\mathcal{J})) + \sum_{m' \in M} f'_{m'}(\mu_M(m')) = (k - O_M(\mathcal{J})) + O_M(\mathcal{J}), \end{aligned}$$

which is a contradiction. Therefore, $\mu \in \text{SM}(\mathcal{I})$. The definition of the new preference functions imply that $\text{balance}_{\mathcal{I}}(\mu) = \text{balance}_{\mathcal{J}}(\mu)$. Since $\text{balance}_{\mathcal{J}}(\mu) \leq k$, we may conclude that $\text{Bal}(\mathcal{I}) \leq k$. $\diamond$

Shrinking Gaps. Currently, there might still exist a man $\bar{m}$ or a woman $\bar{w}$ such that $f_{\bar{m}}(\mu_M(\bar{m})) > 1$ or $f_{\bar{w}}(\mu_W(\bar{w})) > 1$, respectively. In the following rule, we would like to decrease some values assigned by the preference functions of such men and women in a manner that preserves equivalence.

Reduction Rule 4.3.8. *If there exist $\bar{m} \in M$ and $\bar{w} \in W$ such that $f_{\bar{m}}(\mu_M(\bar{m})) > 1$ and $f_{\bar{w}}(\mu_W(\bar{w})) > 1$, then define the preference functions as follows.*

- For all $w \in \mathcal{A}(\bar{m})$: $f'_{\bar{m}}(w) = f_{\bar{m}}(w) - \alpha$ and for all $m \in \mathcal{A}(\bar{w})$: $f'_{\bar{w}}(m) = f_{\bar{w}}(m) - \alpha$, where $\alpha = \min\{f_{\bar{m}}(\mu_M(\bar{m})), f_{\bar{w}}(\mu_W(\bar{w}))\}$.
- For all $a \in M \cup W \setminus \{\bar{m}, \bar{w}\}$: $f'_a = f_a$.

The new instance is $\mathcal{J} = (M, W, \{f'_{m'}\}_{m' \in M}, \{f'_{w'}\}_{w' \in W}, k - \alpha)$.

Lemma 4.3.13. *Reduction Rule 4.3.8 is safe, and $t(\mathcal{I}) = t(\mathcal{J})$.*

Proof. Let us first observe that the set of acceptable pairs in $\mathcal{I}$ is equal to the set of acceptable pairs of $\mathcal{J}$. Furthermore, for every person a , including the cases where

this person is either $\bar{m}$ or $\bar{w}$, any two acceptable partners b and b' of a that satisfy $f_a(b) < f_a(b')$ also satisfy $f'_a(b) < f'_a(b')$, and vice versa. Indeed, this observation follows directly from our definition of the new preference functions. In other words, if a person prefers some person over another in $\mathcal{I}$, then this person also has the same preference order in $\mathcal{J}$, and vice versa. We thus deduce that $\text{SM}(\mathcal{I}) = \text{SM}(\mathcal{J})$.

We proceed by claiming that for all $\mu \in \text{SM}(\mathcal{I})$, we have that $\text{balance}_{\mathcal{I}}(\mu) = \text{balance}_{\mathcal{J}}(\mu) + \alpha$. Indeed, by the definition of the new preference functions and the equation (I) in “Perfect Matching”, we have that $\text{balance}_{\mathcal{I}}(\mu)$

$$\begin{aligned}
&= \max\left\{\sum_{m \in M} f_m(\mu(m)), \sum_{w \in W} f_w(\mu(w))\right\} \\
&= \max\left\{f_{\bar{m}}(\mu(\bar{m})) + \sum_{m \in M \setminus \{\bar{m}\}} f_m(\mu(m)), f_{\bar{w}}(\mu(\bar{w})) + \sum_{w \in W \setminus \{\bar{w}\}} f_w(\mu(w))\right\} \\
&= \max\left\{f'_{\bar{m}}(\mu(\bar{m})) + \alpha + \sum_{m \in M \setminus \{\bar{m}\}} f'_m(\mu(m)), f'_{\bar{w}}(\mu(\bar{w})) + \alpha + \sum_{w \in W \setminus \{\bar{w}\}} f'_w(\mu(w))\right\} \\
&= \max\left\{\sum_{m \in M} f'_m(\mu(m)), \sum_{w \in W} f'_w(\mu(w))\right\} + \alpha = \text{balance}_{\mathcal{J}}(\mu) + \alpha.
\end{aligned}$$

Furthermore, the arguments above also show that $O_M(\mathcal{I}) = O_M(\mathcal{J}) + \alpha$ and $O_W(\mathcal{I}) = O_W(\mathcal{J}) + \alpha$. Hence, we have that $\text{Bal}(\mathcal{I}) = \text{Bal}(\mathcal{J}) + \alpha$. Since k was decreased by α , we may conclude that the rule is safe and that $t(\mathcal{I}) = t(\mathcal{J})$. $\diamond$

After the exhaustive application of Reduction Rule 4.3.8, no more than $\mathcal{O}(t^2)$ number of times, at least one of the two parties does not have any member without a person assigned 1 by his/her preference function. Thus,

Observation 4.3.4. *Let $\mathcal{I}$ be an instance of ABOVE-MIN FBSM that is reduced with respect to Reduction Rules 4.3.1 to 4.3.8. Then, either (i) for every $m \in M$, we have that $f_m(\mu_M(m)) = 1$, or (ii) for every $w \in W$, we have that $f_w(\mu_W(w)) = 1$. In particular, either (i) $O_M = |M|$ or (ii) $O_W = |W|$.*

This concludes the description of our reduction rules. We are now ready to prove Lemma 4.3.1.

Proof of Lemma 4.3.1. Given an instance $\mathcal{I}$ of ABOVE-MIN FBSM, our kernelization algorithm exhaustively applies Reduction Rules 4.3.1 to 4.3.8, after which it outputs the resulting instance, $\mathcal{J}$, as the kernel. Notice that each rule among Reduction Rules 4.3.1 to 4.3.8 can be applied in polynomial time, and it either terminates the execution of the algorithm or shrinks the size of the instance. Hence, it is clear that the instance $\mathcal{J}$ is obtained in polynomial time. The claims that $\mathcal{J}$ and $\mathcal{I}$ are equivalent and that $t(\mathcal{J}) \leq t(\mathcal{I})$ follow directly from the lemmata that prove the safeness of each rule as well as argue with respect to the parameter. By Lemma 4.3.11, we also have that the instance contains at most $2t$ (sad) men and at most $2t$ (sad) women.

It remains to show that the image of the preference function of each person is a subset of $\{1, 2, \dots, t+1\}$. Since the domain of the preference function of each person is the set of acceptable partners for that person, it is sufficient to show that every acceptable pair (m, w) satisfies $f_m(w) \leq t+1$ and $f_w(m) \leq t+1$. By Reduction Rule 4.3.7, every acceptable pair (m, w) satisfies $f_m(w) \leq (k - O_M) + f_m(\mu_M(m))$ and $f_w(m) \leq (k - O_W) + f_w(\mu_W(w))$. Moreover, for any man m and woman w , it holds that $f_m(\mu_M(m)) \leq O_M - (|M| - 1)$ and $f_w(\mu_W(w)) \leq O_W - (|W| - 1)$. Thus, every acceptable pair (m, w) satisfies $f_m(w) \leq k - (|M| - 1)$ and $f_w(m) \leq k - (|W| - 1)$. By Reduction Rule 4.3.3 and Observation 4.3.4, we have that $t = k - \min\{O_M, O_W\} = k - |M| = k - |W|$. Hence, we further conclude that every acceptable pair (m, w) satisfies $f_m(w) \leq t+1$ and $f_w(m) \leq t+1$. Thus, the proof is complete. $\diamond$

4.3.2 Balanced Stable Marriage

Having proved Lemma 4.3.1, we have a kernel for ABOVE-MIN FBSM. We would like to employ this kernelization algorithm to design one for ABOVE-MIN BSM. For this purpose, we need to remove gaps from preference functions. Once we do this, we can view preference functions as preference lists and obtain the desired kernel. Hence, the following lemma concludes the proof of Theorem 8.

Lemma 4.3.14. *ABOVE-MIN BSM admits a kernel that has at most $3t$ men among whom at most $2t$ are sad and at most t are happy, at most $3t$ women among whom at most $2t$ are sad and at most t are happy. Additionally, each happy person has at most $2t + 1$ acceptable partners and each sad person has at most $t + 1$ acceptable partners. Moreover, every stable matching in the kernel is a perfect matching.*

In what follows, we describe our kernelization algorithm for ABOVE-MIN BSM. Let $\mathcal{K} = (M', W', \mathcal{L}_{M'}, \mathcal{L}_{W'}, k')$ be the input instance, which is an instance of ABOVE-MIN BSM. Our algorithm begins by applying the reduction given by Observation 4.1.1 to translate $\mathcal{K}$ into an instance $\mathcal{I}' = (M', W', \mathcal{F}_{M'}, \mathcal{F}_{W'}, k')$ of ABOVE-MIN FBSM. Then, our algorithm applies the kernelization algorithm given by Lemma 4.3.1 to $\mathcal{I}'$, obtaining a reduced instance $\mathcal{I} = (M, W, \mathcal{F}_M, \mathcal{F}_W, k)$ of ABOVE-MIN FBSM. By Lemma 4.3.1, this instance has at most $2t$ men, at most $2t$ women, and the image of the preference function of each person is a subset of $\{1, 2, \dots, t + 1\}$. To eliminate “gaps” in the preference functions, the algorithm proceeds as described below. Note that we no longer apply any reduction rule from Section 4.3.1 (even if its condition is satisfied), as we currently give a *new* kernelization procedure rather than an extension of the previous one. Let us first formally define the notion of a gap.

Definition 4.3.4. *Let $a \in M \cup W$, and i be a positive integer outside the image of f . If there exists an integer $j > i$ that belongs to the image of f , then f_a is said to have a gap at i .*

Inserting Dummies. We have ensured that the largest number in the image of any preference function is at most $t + 1$. As every person is sad, every person must have at least two acceptable partners. Hence, it follows that there are at most $t - 1 \leq t$ gaps. To handle the gaps of *all* people, we create a set of t dummy men and t dummy women. Our objective is to introduce these dummy people as acceptable partners for people who have gaps in their preference functions, such that the function values of the dummy people would fill the gaps. Moreover, currently there are no happy people in the kernel, but after insertion the dummy people will be the happy people of the instance and create at most t happy pairs; and so the following rule would be applied only once.

Reduction Rule 4.3.9. *If there do not exist happy people, then let $X = \{x_1, x_2, \dots, x_t\}$ denote a set of new (dummy) men, and $Y = \{y_1, y_2, \dots, y_t\}$ denote a set of new (dummy) women. For each $i \in \{1, 2, \dots, t\}$, initialize $\mathcal{A}(x_i) = \{y_i\}$, $\mathcal{A}(y_i) = \{x_i\}$ and $f_{x_i}(y_i) = f_{y_i}(x_i) = 1$. The new instance is $\mathcal{J} = (M \cup X, W \cup Y, \{f_m\}_{m \in M \cup X}, \{f_w\}_{w \in W \cup Y}, k + t)$.*

We note that for all $i \in \{1, 2, \dots, t\}$, it holds that (x_i, y_i) is a happy pair.

Lemma 4.3.15. *Reduction Rule 4.3.9 is safe, and $t(\mathcal{I}) = t(\mathcal{J})$.*

Proof. For all $i \in \{1, 2, \dots, t\}$, it holds that (x_i, y_i) is a happy pair, and therefore it is present in every stable matching in $\mathcal{J}$. By our definition of the new preference functions, it is clear that if μ is a stable matching in $\mathcal{I}$, then $\mu' = \mu \cup \{(x_1, y_1), \dots, (x_t, y_t)\}$ is a stable matching in $\mathcal{J}$. Moreover, if μ' is a stable matching in $\mathcal{J}$, then $\mu = \mu' \setminus \{(x_1, y_1), \dots, (x_t, y_t)\}$ is a stable matching in $\mathcal{I}$. Hence, since for all $i \in \{1, 2, \dots, t\}$, it holds that $f_{x_i}(y_i) = f_{y_i}(x_i) = 1$, our definition of the new preference functions directly implies that $\text{Bal}(\mathcal{I}) + t = \text{Bal}(\mathcal{J})$, $O_M(\mathcal{I}) + t = O_M(\mathcal{J})$ and $O_W(\mathcal{I}) + t = O_W(\mathcal{J})$. Hence, $t(\mathcal{I}) = t(\mathcal{J})$, which concludes the proof. $\diamond$

Reduction Rule 4.3.10. [Male version] *If there exists $m \in M$ such that f_m has a gap at some j , then select some $y_i \in Y \setminus \mathcal{A}(m)$, and set $\mathcal{A}'(m) = \mathcal{A}(m) \cup \{y_i\}$ and $\mathcal{A}'(y_i) = \mathcal{A}(y_i) \cup \{m\}$. The preference functions are defined as follows.*

- **The preference function of m :** $f'_m(y_i) = j$, and for all $a \in \mathcal{A}(m)$, $f'_m(a) = f_m(a)$.
- **The preference function of y_i :** $f'_{y_i}(m) = \max_{m' \in \mathcal{A}(y_i)} (f_{y_i}(m') + 1)$, and for all $a \in \mathcal{A}(y_i)$, $f'_{y_i}(a) = f_{y_i}(a)$.
- For all $a \in (M \cup W) \setminus \{m, y_i\}$: $f'_a = f_a$.

The new instance is $\mathcal{J} = (M, W, \{f'_{m'}\}_{m' \in M}, \{f'_{w'}\}_{w' \in W}, k)$.

Lemma 4.3.16. *Reduction Rule 4.3.10 is safe, and $t(\mathcal{I}) = t(\mathcal{J})$.*

Proof. The only modifications that are performed are the insertion of m into the set of acceptable partners of y_i as the least preferred person, and the insertion of y_i into the set of acceptable partners of m in a location that previously contained a gap. Let us first observe that since $f_{x_i}(y_i) = f_{y_i}(x_i) = 1$ and $f'_{x_i}(y_i) = f'_{y_i}(x_i) = 1$, it holds that (x_i, y_i) is a happy pair in both $\mathcal{I}$ and $\mathcal{J}$. Hence, it is clear that $\text{SM}(\mathcal{I}) = \text{SM}(\mathcal{J})$, $O_M(\mathcal{I}) = O_M(\mathcal{J})$, $O_W(\mathcal{I}) = O_W(\mathcal{J})$, and that the balance of any stable matching in $\mathcal{I}$ is equal to its balance in $\mathcal{J}$. We thus conclude that the rule is safe and that $t(\mathcal{I}) = t(\mathcal{J})$. $\diamond$

Analogously, we have a female version of Reduction Rule 4.3.10 where we fill a gap in the preference function of some woman $w \in W$. We do not repeat our arguments again, and straightaway state the following result, which follows directly from the safeness of Reduction Rule 4.3.9 and the male and female version of Reduction Rule 4.3.10.

Lemma 4.3.17. ABOVE-MIN FBSM admits a kernel that has at most $3t$ men among whom at most $2t$ are sad, at most $3t$ women among whom at most $2t$ are sad, and such that each happy person has at most $2t + 1$ acceptable partners and each sad person has at most $t + 1$ acceptable partners. Moreover, the kernel contains at most t happy pairs and the none of the preference functions contain any gaps.

Finally, we translate the kernel for ABOVE-MIN FBSM to an instance of ABOVE-MIN BSM as follows. For all $a \in M \cup W$ and $b \in \mathcal{A}(a)$, we set $p_a(b) = f_a(b)$. The new instance is $\mathcal{J} = (M, W, \{p_m\}_{m \in M}, \{p_w\}_{w \in W}, k)$. Clearly, we thus obtain an equivalent instance, which leads us to say that Lemma 4.3.14 is proved.

4.4 Parameterized Algorithm

In this section, we design a parameterized algorithm for ABOVE-MIN BSM, and prove Theorem 9. As our algorithm is based on the method of bounded search trees, first we will give a brief description of this technique and then describe the procedure.

4.4.1 Bounded Search Tree: An Overview

The running time of an algorithm that uses bounded search trees can be analyzed as follows (see, e.g., [29]). Suppose that the algorithm executes a branching rule which has ℓ branching options (each leading to a recursive call with the corresponding parameter value), such that, in the i^{th} branch option, the current value of the parameter decreases by b_i . Then, $(b_1, b_2, \dots, b_\ell)$ is called the *branching vector* of this rule. For this branching vector the upper bound $T(k)$ on the number of leaves in the search tree is given by the following linear recurrence: $T(k) = T(k - b_1) + T(k - b_2) + \dots + T(k - b_\ell)$, where k is the parameter.

We say that α is the *root* of $(b_1, b_2, \dots, b_\ell)$ if it is the (unique) positive real root

of $x^{b^*} = x^{b^*-b_1} + x^{b^*-b_2} + \dots + x^{b^*-b_\ell}$, where $b^* = \max\{b_1, b_2, \dots, b_\ell\}$. If $r > 0$ is the initial value of the parameter, and the algorithm (a) returns a result when (or before) the parameter is negative, and (b) only executes branching rules whose roots are bounded by a constant $c > 0$, then its running time is upper bounded by $\mathcal{O}^*(c^r)$. In particular, this yields a $\mathcal{O}^*(\alpha^r)$ -time algorithm, where α is the root of the branching vector of our algorithm.

4.4.2 Description of the Algorithm

Given an instance $\widehat{\mathcal{I}} = (\widehat{M}, \widehat{W}, \widehat{\mathcal{L}}_M, \widehat{\mathcal{L}}_W, \widehat{k})$ of ABOVE-MIN BSM, we begin by using the procedure given by Lemma 4.3.14 to obtain (in polynomial time) a kernel $\mathcal{I} = (M, W, \mathcal{L}_M, \mathcal{L}_W, k)$ of ABOVE-MIN BSM such that $\mathcal{I}$ has at most $3t$ men among whom at most $2t$ are sad, at most $3t$ women among whom at most $2t$ are sad. We denote the happy pairs in $\mathcal{I}$ by $(x_1, y_1), \dots, (x_h, y_h)$ for some $h \leq t$, where $X = \{x_1, \dots, x_t\}$ and $Y = \{y_1, \dots, y_t\}$. We denote the set of sad men by M_S , and for that we have $|M_S| \leq 2t$.

We proceed by executing a loop where each iteration corresponds to a different subset $M' \subseteq M_S$. For a specific iteration, our goal is to determine whether there exists a stable matching μ such that the following conditions are satisfied: $\text{balance}(\mu) \leq k$; for all sad men $m \in M'$, $\mu(m) \neq \mu_M(m)$; and for all sad men $m \in M_S \setminus M'$, $\mu(m) = \mu_M(m)$. (Also, we recall that for any happy man x_i , we have $\mu(x_i) = \mu_M(x_i) = y_i$.) A stable matching satisfying these conditions (in the context of the current iteration) is said to be **valid**. We denote $r = k - O_M$, and observe that $r \leq t$ because $t = k - \min\{O_M, O_W\}$.

Let us now consider some specific iteration. To determine whether there exists a valid stable matching, our plan is to execute a branching procedure, called **Branch** (depicted in Algorithm 4.4.2), which outputs every set F of pairs of a man and a

woman, where the man is in M' and the following conditions are satisfied.

1. Every man m in M' participates in exactly one pair (m, w) of F , and for that unique pair, it holds that $w \in \mathcal{A}(m)$ and $p_m(w) > p_m(\mu_M(m))$.
2.
$$\sum_{\substack{m \in M' \\ (m, w) \in F}} (p_m(w) - p_m(\mu_M(m))) \leq r.$$

Algorithm 4.4.1: FPT Algorithm

Data: $\mathcal{I}, k$

Result: A matching μ such that $\text{balance}(\mu) \leq k$

```

1 for  $M' \subseteq M_S$  do
2   Set  $r = k - O_M$ .
3   Set  $\mathcal{F} = \emptyset$ .
4    $\mathcal{F} = \text{Branch}(0, \emptyset)$ .
5   for  $F \in \mathcal{F}$  do
6     Let
7        $\mu = F \cup \{(x_1, y_1), (x_2, y_2), \dots, (x_h, y_h)\} \cup \{(m, \mu_M(m)) : m \in M_S \setminus M'\}$ 
8       if  $\mu$  is a stable matching in  $\mathcal{I}$  then
9         if  $\text{balance}(\mu) \leq k$  then
10           return  $\mu$ 
11         end
12       end
13   end
14 return  $\emptyset$ .
```

4.4.3 The Procedure **Branch**

We now present the description of the procedure **Branch** in the context of some set $M' \subseteq M_S$. Let us denote $M' = \{m_1, m_2, \dots, m_p\}$ for an appropriate choice of p .

Each call to our procedure is of the form **Branch**($i, \mathcal{P}$) where $i \in \{0, 1, \dots, p+1\}$ and $\mathcal{P}$ is a set of pairs of a man in $\{m_1, m_2, \dots, m_i\}$ and a (sad) woman, such that the following conditions are satisfied.

(I) Each man m in $\{m_1, m_2, \dots, m_i\}$ participates in exactly one pair (m, w) of $\mathcal{P}$, and for that unique pair, it holds that $w \in \mathcal{A}(m)$ and $p_m(w) > p_m(\mu_M(m))$. We define $\mu_{\mathcal{P}}$ as the *function* whose domain is $M_i = \{m_1, m_2, \dots, m_i\}$ and which assigns to each man m in its domain the unique woman w such that $(m, w) \in \mathcal{P}$.

(II) Define $\text{balance}(\mathcal{P}) = \sum_{m \in \{m_1, \dots, m_i\}} (p_m(\mu_{\mathcal{P}}(m)) - p_m(\mu_M(m)))$. Then, $\text{balance}(\mathcal{P}) \leq r$.

Note that for the case $i = 0$, we have $\mathcal{P} = \emptyset$, and the algorithm calls the procedure **Branch**($0, \emptyset$).

The objective of a call to **Branch**($i, \mathcal{P}$) is to return a family of sets, $\mathcal{F}$, where each set $F \in \mathcal{F}$ is a set of pairs of a man in $\{m_{i+1}, m_{i+2}, \dots, m_p\}$ and a woman such that the following conditions are satisfied.

(III) Every man m in $\{m_{i+1}, m_{i+2}, \dots, m_p\}$ participates in exactly one pair (m, w) of F , and for that unique pair, it holds that $w \in \mathcal{A}(m)$ and $p_m(w) > p_m(\mu_M(m))$. We define μ_F as the *function* whose domain is $\{m_{i+1}, m_{i+2}, \dots, m_p\}$ and which assigns to each man m in its domain the unique woman w such that $(m, w) \in F$.

(IV) $\text{balance}(\mathcal{P}) + \sum_{m \in \{m_{i+1}, m_{i+2}, \dots, m_p\}} (p_m(\mu_F(m)) - p_m(\mu_M(m))) \leq r$.

Thus, each member of the family $\mathcal{F}$ returned by **Branch**($i, \mathcal{P}$) extends the matching $\mu_{\mathcal{P}}$ such that the resulting matching satisfies our stated goal, condition [2](#) (page [90](#)).

Measure: We use $(r - \text{balance}(\mathcal{P}))$ as the measure to analyze the **Branch** procedure. The measure is initially equal to r because initially $\mathcal{P} = \emptyset$. Therefore, according to the method of bounded search trees (see Section [4.4.1](#)), in order to derive the $\mathcal{O}^*(2^r)$

Algorithm 4.4.2: Branch($i, \mathcal{P}$)

Data: A pair $(i, \mathcal{P})$ satisfying conditions (I) and (II)

Result: A family of sets of man-woman pairs that satisfy conditions (III) and (IV)

```
1 if  $r < 0$  then
2   | return  $\emptyset$ .
3 else if  $i = p$  then
4   | return  $\{\emptyset\}$ .
5 else
6   | Let  $\widetilde{W} = \{w \in \mathcal{A}(m_{i+1}) \setminus Y : p_{m_{i+1}}(w) > p_{m_{i+1}}(\mu_M(m_{i+1}))\}$ 
7   |  $\widetilde{W} = \widetilde{W} \setminus (\{w : (m, w) \in \mathcal{P}\} \cup \{\mu_M(m) : m \in M_S \setminus M'\})$ 
8   | if  $|\widetilde{W}| < r$  then
9   |   |  $W^* = \widetilde{W}$ .
10  | else
11  |   | Let  $W^*$  be the set of  $r$  women in  $\widetilde{W}$  who are most preferred by  $m_{i+1}$ .
12  |   end
13  | Let  $W^* = \{w_1, w_2, \dots, w_q\}$ .
13  | // Note that  $q \leq r$ 
14  | for  $j \in \{1, 2, \dots, q\}$  do
15  |   |  $r = r - (p_{m_{i+1}}(w_j) - p_{m_{i+1}}(\mu_M(m_{i+1})))$ .
15  |   | // since it is the increase in balance( $\mathcal{P}$ ) if  $(m_{i+1}, w_j)$  is
15  |   | added to  $\mathcal{P}$ 
16  |   | Let  $\mathcal{F}_j = \text{Branch}(i+1, \mathcal{P} \cup \{(m_{i+1}, w_j)\})$ .
16  |   | //  $\mathcal{F}_j$  is the family of sets of pairs that is returned by
16  |   | the recursive call
17  |   |  $\mathcal{F} = \mathcal{F} \cup \{F \cup \{(m_{i+1}, w_j)\} : F \in \mathcal{F}_j\}$ .
18  |   end
19 end
20 return  $\mathcal{F}$ .
```

running time, it is sufficient to ensure that **Branch (a)** returns a result when (or before) the measure r is negative, and **(b)** only executes branching rules whose roots are bounded by 2.

When the measure r is negative, we simply return $\mathcal{F} = \emptyset$, as there does not exist a set F satisfying the conditions above. Otherwise, when $i = p$, we return $\mathcal{F} = \{\emptyset\}$. The time complexity analysis is presented in the proof of Claim 4.4.2.

We describe the **Branch** procedure in words for the sake of exposition.

Overview of Algorithm 4.4.2: Consider a call Branch($i, \mathcal{P}$) where $r \geq 0$ and

$i < p$. We define $\widetilde{W}$ to be the subset of sad women who are neither part of any pair in $\mathcal{P}$ nor are they matched to any man in $M_S \setminus M'$ under the man-optimal stable matching. That is, $\widetilde{W} = \{w \in \mathcal{A}(m_{i+1}) \setminus Y : p_{m_{i+1}}(w) > p_{m_{i+1}}(\mu_M(m_{i+1}))\} \setminus (\{w : (m, w) \in \mathcal{P}\} \cup \{\mu_M(m) : m \in M_S \setminus M'\})$.

We further refine $\widetilde{W}$ by letting W^* denote the set of r women in $\widetilde{W}$ who are most preferred by m_{i+1} . In case $|\widetilde{W}| < r$, we simply denote $W^* = \widetilde{W}$. Let us also denote $W^* = \{w_1, w_2, \dots, w_q\}$ for the appropriate value of $q \leq r$. Then, our procedure executes q branches. At the j^{th} branch, **Branch** calls itself recursively with $(i+1, \mathcal{P} \cup \{(m_{i+1}, w_j)\})$. Eventually, **Branch** returns $\bigcup_{j=1}^q \{ \{(m_{i+1}, w_j)\} \cup F : F \in \mathcal{F}_j \}$ where for each $j \in \{1, 2, \dots, q\}$, we set $\mathcal{F}_j$ to be the family of sets of pairs that was returned by the recursive call of the j^{th} branch.

Correctness of the Branch procedure. The correctness follows from the observation that we are conducting an exhaustive search. More precisely, if there exists a set F satisfying Conditions (III) and (IV), then it must include exactly one of the pairs in $\{(m_{i+1}, w_j) : j \in \{1, 2, \dots, q\}\}$, hence the manner in which the family $\mathcal{F}$ and the measure r are updated follow straightforwardly. We prove it formally in the following claim.

Claim 4.4.1. *Let $\mathcal{F}$ denote the output of **Branch**($i, \mathcal{P}$). Suppose that the matching $\mu_{\mathcal{P}}$ satisfies Conditions (I) and (II). Then each set $F \in \mathcal{F}$ outputted by **Branch**($i, \mathcal{P}$) yields a matching μ_F that satisfies Conditions (III) to (IV).*

Proof. Note that due to the condition in Line 4 of the algorithm, the output of the call **Branch**($p, \cdot$) is $\emptyset$. Hence, a pair (m_{i+1}, w_j) is added to $\mathcal{F}$ if $p \geq i+1$ and $r \geq 0$. Since $M' = \{m_1, m_2, \dots, m_p\}$, we have that $\mathcal{F}$ satisfies Condition (I).

In each iteration we add one (man, woman) pair to $\mathcal{P}$ and reduce r by the increase in $\text{balance}(\mathcal{P})$. Due to Line 2 of the algorithm, the process stops if r becomes negative. Hence, $\text{balance}(\mathcal{P}) \leq r$, that is Condition (II) is satisfied.

For each man $m \in \{m_1, \dots, m_p\}$, a pair involving m is added to F once at Line 17. Also, if (m, w) is added to F , then $w \in W^*$ which ensures Condition (III) holds. In a branch, if we add (m_{i+1}, w_j) to F , we have already reduced r by $p_{m_{i+1}}(w_j) - p_{m_{i+1}}(\mu_M(m_{i+1}))$. Hence, Condition (IV) is maintained as an invariant. $\diamond$

Claim 4.4.2. *Procedure **Branch** has a running time of $\mathcal{O}^*(2^r)$.*

Proof. As noted before, the measure $(r - \text{balance}(\mathcal{P}))$ is initially $r > 0$ because $\mathcal{P} = \emptyset$. We observe that at the j^{th} branch, the measure changes from $r - \text{balance}(\mathcal{P})$ to $r - \text{balance}(\mathcal{P} \cup \{(m_{i+1}, w_j)\})$. By our definition of w_j , we have $p_{m_{i+1}}(w_j) - p_{m_{i+1}}(\mu_M(m_{i+1})) \geq j$. Hence, at the worst case, the branching vector is $(1, 2, \dots, r)$.

Since the polynomial $x^r - \sum_{i=1}^{r-1} x^i - 1 = 0$ attains the global minima at $x = \frac{2r}{r+1}$ which approaches 2 as $r \rightarrow \infty$, we can conclude that 2 is the best upper bound for the branching vector. $\diamond$

Thus, the correctness and the time complexity of the procedure **Branch** is complete. $\diamond$

4.4.4 Algorithm

Here, we describe the last step of the algorithm and we will argue that by having the branching procedure **Branch**, we can conclude the proof of the correctness of the algorithm.

We examine each set F in the outputted family of sets. Then, we check whether the pairs in F , together with $(x_1, y_1), (x_2, y_2), \dots, (x_h, y_h)$ and every pair in $\{(m, \mu_M(m)) : m \in M_S \setminus M'\}$ form a stable matching whose balance is at most k . If the answer is positive, then we terminate the execution and accept. At the end, if we did not accept in any iteration, we reject.

Correctness. To see why the decision made in the above step is correct, suppose that there exists a stable matching μ whose balance is at most k . In this case, due to our exhaustive search, there exists some iteration in which μ is also valid. In that iteration is associated with some $M' \subseteq M_S$. Observe that the set of pairs $\{(m, \mu(m)) : m \in M'\}$ is one of the sets in the outputted family $\mathcal{F}$. Indeed, the satisfaction of Condition 1 (Section 4.4) follows from the fact that μ is a stable matching satisfying Conditions (I) and (III) of validity. Condition 2 (Section 4.4) is satisfied because of the fact that μ satisfies the Condition (II) and (IV) of validity.

Claim 4.4.3. *The time complexity of our algorithm to solve ABOVE-MIN BSM is $\mathcal{O}^*(8^t)$.*

Proof. Let us denote by T the running time of the procedure **Branch**. Then, the total running time of our algorithm is bounded by $\mathcal{O}^*(2^{|M_S|} \cdot T) = \mathcal{O}^*(4^t \cdot T)$.

Recall that $r = k - O_M$, and $t = k - \min O_M, O_W$, hence $r \leq t$. Hence, to derive the running time in Theorem 9, it is sufficient to ensure that $T = \mathcal{O}^*(2^r)$, as proved in Claim 4.4.2. Thus, the time complexity of our algorithm is $\mathcal{O}^*(4^t \cdot 2^r) = \mathcal{O}^*(8^t)$ since $r \leq t$. $\diamond$

Thus, Theorem 9 is proved. $\diamond$

4.5 Conclusion

In this chapter we studied an optimization variant of the famous stable matching problem in the realm of parameterized complexity. **BALANCED STABLE MATCHING** is a constrained stable matching that lies in between the two extremes i.e., the men optimal and the women optimal stable matchings, being globally desirable and fair to both sides. Mainly we studied the problem with respect to two above-guarantee parameters. We showed dichotomous results with respect to these two parameters.

To show the problem is **FPT** with respect to the first parameter, we designed a kernelization algorithm for the problem instead of directly designing a **FPT** algorithm. We believe this idea can be used to show fixed parameter tractability for other **NP**-hard problems in this area.

Chapter 5

Group Activity Seclection

Problem on Graphs(gGASP)

We have studied the complexity of two variants of the STABLE MATCHING problem in the previous chapters. We shift our focus to a different allocation problem where only one side (agents) has preference over the elements from the other side (activities). However, a agent's preference for an activity also depends on how many other agents are allocated to that activity. We ask if an activity should be assigned to a group of agents such that no agent wants leave his/her assigned group. In fact, this setting is common in many practical applications ranging from carpooling to workload delegation. Our previous notion of stability does not apply here directly. We study a different notion of stability called Nash stability. It formally enforces the condition that no agent wants to leave his/her assigned group.

Division of labor is required in varied real-world situations. For a task to be accomplished, be it the construction of a building or the development of a product, it is necessary to *assign* agents (such as people or companies) to appropriate activities, and those agents must be willing to contribute towards the common goal. Though workload delegation is perhaps the first example that comes to mind, management of

cooperation—or more precisely, formation of groups by agents participating in specific activities—is ubiquitous in almost all aspects of life. Indeed, other examples range from carpooling and seating arrangements to hobbies such as tennis or basketball. All such situations are neatly captured by the **Group Activity Selection Problem (GASP)** introduced in [32].

An instance of **GASP** is given by a finite set of agents N , where $|N| = n$, a finite set of activities $A = A^* \cup \{a_\emptyset\}$, where $A^* = \{a_1, \dots, a_p\}$ and $a_\emptyset$ is a *void activity*, and a profile $(\succeq_v)_{v \in N}$ of complete and transitive preference relations over the set of alternatives $X = (A^* \times \{1, 2, \dots, n\}) \cup \{(a_\emptyset, 1)\}$.^[1] Each alternative is a two sized tuple indicating an activity and a size of the group performing the activity. The void activity $a_\emptyset$ is introduced to allow agents to avoid undertaking activities, which also enables agents to be independent. For example, in the case of the development of a product, the set N may consist of employees of some company, each activity in the set A may correspond to the design of a certain component of the product, and the profile $(\succeq_v)_{v \in N}$ may be constructed according to the skills/personal preferences of the employees and their abilities/willingness to function in groups of varied sizes. The void activity would allow to exclude employees from the current project in case no suitable activities can be assigned to them.

The outcome of **GASP** is defined as an *assignment*, which is simply a function $\pi : N \rightarrow A$. Clearly, an arbitrary assignment is extremely undesirable unless the profile $(\succeq_v)_{v \in N}$ is completely meaningless. To take the profile into account, it is first natural to request that π would at least be *individually rational (IR)*, which means that for every agent $v \in N$ with $\pi(v) = a$ ($\neq a_\emptyset$), we have $(a, |\pi^a|) \succeq_v (a_\emptyset, 1)$, where $\pi^a = \pi^{-1}(a) = \{v \in N : \pi(v) = a\}$.^[2] That is, no agent v would rather “be alone” than being part of a group of size $|\pi_v|$ that performs activity $a = \pi(v)$, where $\pi_v = \pi^a$ (that is, π_v is the set of all those agents that have been assigned the same

¹For the sake of consistency, we follow the notations and definitions of [80].

²As we would always work with IR assignments, when specifying preference profiles, we would only explicitly state the alternatives that are preferred more than $a_\emptyset$.

activity as v). In addition, to enforce the execution of the activities in practice, no individual should desire to act on its own by deserting its group in favor of joining another group. In other words, we would like the assignment to be Nash stable. Formally, an agent $v \in N$ is said to have an *NS-deviation* to an activity $a \in A^*$ if $a \neq \pi(v)$ and $(a, |\pi^a| + 1) \succ_v (\pi(v), |\pi_v|)$, that is, v prefers to join the activity a , given everyone else plays the same activity as before. Accordingly, π is said to be *Nash stable* if it is individually rational and no agent has an NS-deviation. Darmann et al [32] showed that if an assignment is individually rational, the only agents who can profitably deviate are the ones assigned to the void activity. The requirement of Nash stability is much stronger than that of individual rationality, and there are cases where a Nash stable assignment does not exist.

Let us take a step back and observe that if an assignment π is Nash stable, then the only implication is that no agent has an alternative more preferred than the situation assigned to it by π . However, we do not ensure by any means that the agent would actually be able/willing to cooperate with other members in its group, so that the assignment can actually be executed in a satisfactory manner. Notably, relevant relations among agents, such as acquaintanceship, compatibility or geographical distance, can often be modeled naturally using graphs. To exploit this modeling ability, Igarashi et al. [80] introduced **gGASP**. Specifically, it is required that each group would correspond to a connected set of the underlying graph. For a deeper understanding of the rationale underlying this requirement, let us consider the case where the graph is a social network. Then, by ensuring that each group is a connected set, we ensure that each individual in the group would be acquainted with at least one other person in the group. The desirability of such property is clear when discussing activities such as carpooling, seating arrangements or sports as it is conceivable that people would prefer to share a taxi/sit next to/play with at least one other person whom they know. In the context of workload delegation, apart from a social aspect, it is likely that agents who are familiar with each other would

also be able to work more efficiently with each other, with each agent having at least one other agent as a comfortable communication link or a source to “count on”.

Formally, an instance of **gGASP** [80] consists of an instance $(N, (\succeq_v)_{v \in N}, A)$ of **GASP** and a set of communication links between agents, $L \subseteq \{\{u, v\} \mid u, v \in N, u \neq v\}$. Thus, we assume that we are also given a graph G with vertex set $V(G) = N$ and $E(G) = L$. Here, G is called underlying network (or graph) of **gGASP**. A set $S \subseteq N$ of agents is said to be a *coalition*, and it is a *feasible coalition* if $G[S]$ is connected where $G[S]$ is the graph induced on the vertices in S . Now, an NS-deviation by an agent v to an activity $a \in A^*$ is called a *feasible* NS-deviation if $\pi^a \cup \{v\}$ is a feasible coalition. Thus, in the context of **gGASP**, an assignment π is said to be *Nash stable* if it is individually rational and no agent has a feasible NS-deviation. We would be interested in the following question.

Nash Stable gGASP (gNSGA)

Input: An instance $\mathcal{I} = (N, (\succeq_v)_{v \in N}, A, G)$ of **gGASP**.

Task: Does $\mathcal{I}$ have a feasible Nash stable assignment (**fNsa**) ?

Igarashi et al. [80] showed that **gNSGA** is NP-complete even when the underlying graph is a path, a star, or if the size of each connected component is bounded by a constant. In addition, they exhibit FPT algorithms (for the same graph classes) when parameterized by the number of activities. In a more recent work by Igarashi et al. [77], the authors show that when parameterized by the number of players, **gNSGA** is W[1]-hard on cliques (the classical setting of **GASP**), but admits an XP-algorithm for the same graph classes. Furthermore, when the underlying graph is a clique, **gNSGA** is W[1]-hard when parameterized by the number of activities. They also give an FPT algorithm for acyclic graphs, parameterized by the number of activities. Specifically, Igarashi et al. [80] posed the following open question.

For general graphs, the exact parameterized complexity of determining the

existence of stable outcomes is unknown... for other networks, including trees, it is not even clear whether our problem is in XP with respect to the number of activities.

Our Contribution. Given that gNSGA is NP-hard even on paths and stars [80], and as this problem inherently encompasses parameters that can be often expected to be small in practice, it is indeed very natural to examine it from the viewpoint of parameterized complexity.³ In this context, we take the line of investigation initiated by the studies [80, 77] several steps forward, significantly advancing the state-of-the-art. In fact, as we explain below, we push some boundaries to their limits, and along the way, we give answer to questions posed by Igarashi et al. [80] that are even stronger than requested.

Hardness: Firstly, we consider $p = |A^*|$, the number of activities, as the parameter. Here, we show that gNSGA is NP-hard *even when merely one activity is present*, that is, $p = 1$. More precisely, we prove the following theorem. Here, Δ is the maximum degree of the graph G .

Theorem 10. *The gNSGA problem is NP-hard even when $p = 1$ and $\Delta = 5$.*

Recall that Igarashi et al. [80] contemplated whether gNSGA is fixed-parameter tractable (FPT) with respect to p . We show that even if $p = 1$, the problem is already NP-hard, and in fact it remains NP-hard even on graphs where the maximum degree Δ is as small as 5. In particular, we derive that gNSGA is para-NP-hard when the parameter is $p + \Delta$. That is, Theorem [10] implies that we do not expect to have an FPT algorithm, or even merely an XP algorithm, on general graphs with respect to both p and Δ *together*. Indeed, the existence of an XP algorithm, that is, an algorithm with running time $n^{f(p, \Delta)}$, where $n = |N|$ and f is any function depending only on p and Δ , would contradict Theorem [10].

³For standard definitions concerning parameterized complexity, see [29] or Section [2].

FPT Algorithm on General Graphs: In light of Theorem [10], we consider an additional parameter t —the maximum size of any group that can form a feasible coalition. Having this parameter at hand, we are able to design an FPT algorithm that handles *general graphs*. Before we state our theorem formally, note that t is a natural choice for a parameter. Indeed, sport teams/matches usually involve only few players, a taxi or a table have only limited space, and certain tasks are clearly suitable, or best performed, when only few people undertake them. We remark that Δ can also often be expected to be small. For example, when most people in an event (say, a donation evening) do not know each other well, this would indeed be the case when planning a seating arrangement. In addition, when new participants sign-up to organized sport activities, they might only know those friends that are also interested in those exact activities. Moreover, when a company operating across different countries would like to undertake some task, while employees generally know only those other employees with whom they share the same floor, we again arrive at a situation where Δ is small.

Theorem 11. *gNSGA on general graphs is solvable in time $\mathcal{O}((\Delta p)^{\mathcal{O}(tp)} \cdot n \log n)$.*

The proof of Theorem [11] uses the idea of an n - p^* - q^* -lopsided-universal family, introduced in [51], to “separate” agents that are assigned non-void activities from their neighbors. Once this is done, a non-trivial dynamic programming algorithm is developed to test whether there exists an fNsa.

FPT Algorithm for Graphs of Bounded Treewidth: Igarashi [80] designed FPT algorithms for gNSGA on paths, stars and graphs whose connected components are restricted to have constant size. In a more recent article, Igarashi et al. [77] designed an FPT algorithm with running time $\mathcal{O}(p^p \cdot (n + p)^{\mathcal{O}(1)})$ for gNSGA on acyclic graphs (i.e. forests). We generalize this result to a substantially wider class of graphs that includes all the above classes of graphs, namely, graphs of bounded treewidth. This class includes graphs that have an unbounded number of cycles, and

in fact it even generalizes the class of all graphs whose feedback vertex set number is small. Formally, we derive the following theorem.

Theorem 12. *The gNSGA problem on graphs of treewidth $\mathbf{tw}$ is solvable in time $\mathcal{O}(4^p \cdot (n + p)^{\mathcal{O}(\mathbf{tw})})$.*

Notably, our algorithm solves gNSGA on trees (where $\mathbf{tw} = 1$) in time $\mathcal{O}(4^p \cdot (n + p)^{\mathcal{O}(1)})$, that is, significantly faster than the specialized algorithm by Igarashi et al. [77]. In fact, its running time also matches the running time of the even more specialized algorithm by Igarashi et al. [77] for paths.

Related works. Papers [80, 77] that specifically solve gNSGA on restricted classes of graphs were discussed in some details earlier. Here, we give a brief (non-comprehensive) survey of a few results related to problems of flavor similar to that of gGASP, so as to understand the well established roots of gNSGA.

The literature on graph based cooperative games, to which gGASP is a new addition, can be traced back to Myerson’s seminal paper [128] which introduced the graph theoretic model of cooperation, where vertices represent agents participating in the game and edges between pairs of vertices represent cooperative relationship between agents corresponding to the vertices. In cooperative games, there are two basic notions of stability, one based on the individual and the other on the group. The latter notion corresponds to what is known as *core stability*. Hedonic games [10, 56, 57] form a domain similar to that of GASP where agents have preferences over other agents, but also groups of agents that include themselves. The primary challenges of designing efficient algorithms in hedonic games is that the space requirement for just storing/representing the preference profile is (in general) exponential in the number of agents in the game. Consequently, people have studied the problem in sparse graphs such as trees, or those with a small number of connected components, Igarashi and Elkind’s [78] is a recent work in this direction. Papers such as [22, 40, 78] explore stability in different kinds of cooperative games; but the central findings

of these papers are in stark contrast with that of [80] showing that restricted graph classes, such as paths, trees, stars etc, are amenable for algorithms that efficiently compute stable solutions. Building on the work of [80], Igarashi *et al.* [77] studied the parameterized complexity of Nash stability, core stability, as well as individual stability in gGASP with respect to parameters such as the number of activities and the number of players. Finally, we conclude our discussion by pointing the reader to a vast array of literature on *coalition formation games*. We refer the reader to [114, pg 222-223, 330] for an extensive discussion on the topic.

5.1 Preliminaries

For standard graph theoretic notation we refer to [35]. For a detailed definition of (*nice*) *tree decomposition*, see Section 2 or [29, pg 161]. For easy reference, we provide the relevant definitions.

We will give a polynomial time many-to-one reduction from the STEINER TREE^{*} problem (defined below) on graphs of maximum degree at most 4 to prove Theorem 10. Towards this we prove that the STEINER TREE^{*} problem is NP-complete.

STEINER TREE^{*}

Input: An undirected graph G^* on n vertices, $K \subseteq V(G^*)$ (called terminals) and a positive integer ℓ .

Task: Does there exist $H \subseteq V(G^*)$ such that $G^*[H]$ is a tree, $K \subseteq H$ and $|H| = \ell$?

STEINER TREE^{*} differs from the usual STEINER TREE problem as follows: here we demand the size of $|H|$ to be *exactly equal* to ℓ rather than at most ℓ . For the sake of completeness, first, we give a polynomial time many-to-one reduction to

show that the STEINER TREE* problem on graphs of maximum degree at most 4 is NP-complete.

5.1.1 NP-completeness of STEINER TREE* on graphs of maximum degree 4.

One of the known NP-completeness reductions [61] for STEINER TREE is given by a polynomial time many one transformation from EXACT COVER BY 3-SETS (X3C). In this problem we are given a set $\mathcal{X} = \{x_1, \dots, x_{3q}\}$ of elements and a collection $\mathcal{S} = \{S_1, \dots, S_n\}$ of 3-element subsets of X and the objective is to check whether there exists a subcollection $\mathcal{C} \subseteq \mathcal{S}$ such that every element of $\mathcal{X}$ is included in exactly one subset $S \in \mathcal{C}$. The problem is hard even if each element of $\mathcal{X}$ appears in at most three sets of $\mathcal{S}$ [61]. To show that STEINER TREE is NP-complete on graphs of maximum degree 4 we restrict ourselves to the instances of X3C where each element of $\mathcal{X}$ appears in at most three sets of $\mathcal{S}$. Next we give the known reduction from X3C to STEINER TREE. Given an instance $(\mathcal{X}, \mathcal{S})$ we build an instance (G, K, ℓ) as follows. The vertex set $V(G) = \mathcal{X} \cup \mathcal{S} \cup v_{\text{sp}}$. That is, we have a vertex for each element in $\mathcal{X}$ and each set in $\mathcal{S}$. Finally, we have a special vertex v_{sp} . The edge set $E(G)$ consists of the following:

$$E(G) = \{v_{\text{sp}}S_1, \dots, v_{\text{sp}}S_n\} \cup \bigcup_{j \in \{1, \dots, n\}} \{xS_j \mid x \in S_j\}.$$

That is, there is an edge from v_{sp} to each vertex corresponding to sets in $\mathcal{S}$ and an edge xS_j if x appears in the set S_j . The set of terminals K is given by $\{v_{\text{sp}}\} \cup \mathcal{X}$. From here, one can show that $(\mathcal{X}, \mathcal{S})$ is a YES-instance of X3C if and only if $(G, K, 4q + 1)$ is a YES-instance of STEINER TREE. Observe that since we started with an instance of X3C where each element of $\mathcal{X}$ appears in at most three sets of $\mathcal{S}$ we have that every vertex except v_{sp} has degree at most 4. Thus, to make the

instance $(G, K, 4q + 1)$ of maximum degree at most 4, we replace v_{sp} with a binary tree $\mathcal{T}$ that has n leaves. We uniquely assign each leaf w of $\mathcal{T}$ to a vertex $S \in \mathcal{S}$ and give an edge between w and S . Finally, make all the nodes in the binary tree as well as those present in $\mathcal{X}$ as the terminal set. Also, set $\ell = |V(\mathcal{T})| + 4q$. This completes the reduction.

A simple reduction from STEINER TREE (on graphs of maximum degree 4) that maps an instance to itself shows that STEINER TREE^{*} (on graphs of maximum degree 4) is NP-complete as well.

5.2 Hardness

In this section, we show that gNSGA is NP-complete *even when there is only one activity* and the maximum degree of the underlying graph is at most 4. Towards this, we give a polynomial time many-to-one reduction from STEINER TREE^{*} on graphs of maximum degree at most 4 to gNSGA.

5.2.1 NP-completeness of gNSGA

We give our construction that given an instance (G^*, K, ℓ) of STEINER TREE^{*}, produces an instance of gNSGA in polynomial time.

Construction. We first show how to construct the underlying graph G from G^* . To this end, we take a copy of G^* and make the following additions to construct G . For every $w \in K$, we construct a path P_w on $n + 1$ *dummy vertices* and add an edge between an end-point of P_w and the terminal vertex w . Now, for each vertex $u \in V(G^*) \setminus K$, we add a new vertex u' and connect u' to u . The vertex u' will act as a *stalker* for the vertex u . A vertex $x \in V(G)$ is called **(i)** a *terminal vertex* if $x \in K$, **(ii)** a *non-terminal vertex* if $x \in V^* = V(G^*) \setminus K$, **(iii)** a *dummy vertex*

if $x \in \text{Dummy} = \cup_{w \in K} P_w$, and **(iv)** a *stalker vertex* if $x \in \text{Stalker} = \{u' \mid u \in V^*\}$. This completes the construction of G .

Intuitively, a vertex in **Stalker** is connected to only one vertex in V^* and will prefer to form a coalition of size two than staying alone. Whereas, the neighboring vertex in V^* prefers to either join a large coalition or stay alone. So no stable assignment can assign a vertex from **Stalker** in a coalition of size two. The behaviour of the vertices in **Stalker** is similar to the vertices in Stalker Game [79]. The **Dummy** vertices and vertices in K prefers to join a large coalition as well. We will show that $K \cup \text{Dummy}$ along with $\ell - k$ vertices from V^* forms a stable coalition. This coalition is connected and hence gives a steiner tree.

Having constructed G , the instance $\mathcal{I}$ of **gNSGA** is defined as follows:

- The set of agents is $N = V(G)$ and the set of activities is $A = \{a\}$. That is, we only have one activity (in addition to the void activity $a_\emptyset$).
- Now we define the preference profiles $(\succeq_v)_{v \in N}$ of the agents. Let $|K| = t$ and $\gamma = (n+2)t + (\ell - t)$. Since $(a_\emptyset, 1)$ is last assignable activity, we do not need to describe the activities that are less preferred than $(a_\emptyset, 1)$ in $\succ_v$. For an agent $v \in N$, the preference profile is

$$\succeq_v := \begin{cases} \langle (a, \gamma), (a_\emptyset, 1) \rangle & \text{if } v \in K \\ \langle (a, \gamma), (a, 1), (a_\emptyset, 1) \rangle & \text{if } v \in V^* \\ \langle (a, \gamma), (a, \gamma + 1), (a_\emptyset, 1) \rangle & \text{if } v \in \text{Dummy} \\ \langle (a, 2), (a_\emptyset, 1) \rangle & \text{if } v \in \text{Stalker} \end{cases}$$

This completes the description of the instance of **gNSGA**.

Correctness. For correctness we show the following equivalence.

Lemma 5.2.1. (G^*, K, ℓ) is a YES-instance of **STEINER TREE*** if and only if $\mathcal{I}$ is

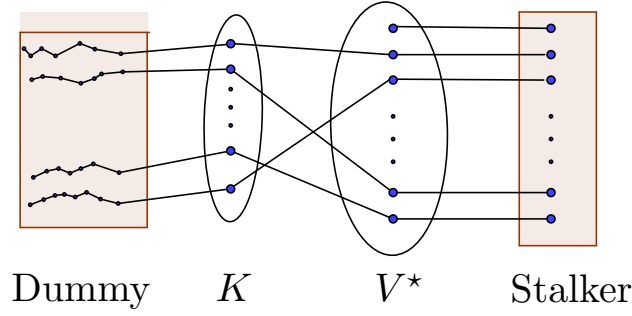


Figure 5.1: The construction of the underlying graph G .

a YES-instance of gNSGA.

Proof. For the forward direction, assume that there exists $H \subseteq V(G^*)$ such that $G^*[H]$ is connected, $K \subseteq H$ and $|H| = \ell$. Using the solution graph H , we define an assignment $\pi : N \rightarrow A$ as follows. For an agent $v \in N$,

$$\pi(v) := \begin{cases} a & \text{if } v \in (H \cup \text{Dummy}) \\ a_\emptyset & \text{otherwise} \end{cases}$$

Next, we prove that π is an fNsa; with that the forward direction is proved.

Claim 5.2.1. π is an fNsa.

Proof. Notice that the following γ agents are assigned to the activity a : the vertices in $H \cup \text{Dummy}$. Indeed, $|H \cup \text{Dummy}| = |H| + |\text{Dummy}| = (\ell - t) + t + (n + 1)t = \gamma$. Let $Z = H \cup \text{Dummy}$. Any agent $u \in Z$ does not want to deviate since (a, γ) is their most preferred option. Now, for any vertex $u \in V(G) \setminus V(K) \cup Z$ (non-terminal vertices not appearing in H and the stalker vertices), $(a_\emptyset, 1) \succ_u (a, \gamma + 1)$. Therefore, no vertex of $V(G')$ wants to deviate to another activity to which a neighbor is assigned. Hence, π is Nash stable. Now we show that it is fNsa. Now the vertices in π^a are the vertices of H and Dummy. Since G^* is an induced subgraph of G , we have that $G[H]$ is connected and it contains all the terminal vertices (K). The dummy vertices are connected to the terminals via paths and thus $G[\pi^a]$ is connected. All

other agents are assigned to $a_\emptyset$, and thus by definition they are connected. This implies that for every agent $v \in N$, π_v is a feasible coalition. Hence, π is an **fNsa**. $\diamond$

Now we will show the other direction. Let $\pi : N \rightarrow A$ be an **fNsa**. We first derive properties of π .

Property 1: *For every vertex $v' \in \text{Stalker}$, $\pi(v') = a_\emptyset$.* For a contradiction assume that $\pi(v') = a$. Since v' would not join the activity a alone, the vertex v in $V^\star$ joins the activity a . That is, $\pi(v) = a$. But $(a_\emptyset, 1) \succ_v (a, 2)$. Hence, v wants to deviate to the void activity $a_\emptyset$, a contradiction to the stability of π .

Property 2: $\pi^a \neq \emptyset$. For a contradiction assume that no vertex is assigned to activity a . Let $v \in V^\star$, then $(a, 1) \succ_v (a_\emptyset, 1)$. That is, v has an NS-deviation to the activity a , again contradicting the stability of π .

Property 3: *If there exists a vertex $v \in V^\star$ such that $v \in \pi^a$, then $|\pi^a| > 1$.* For a contradiction assume that $|\pi^a| = 1$ (by Property 2 we know that $\pi^a \neq \emptyset$). Then, the stalker vertex v' has an NS-deviation to the activity a as $(a, 2) \succ_{w'} (a_\emptyset, 1)$.

Property 4: $|\pi^a| \in \{\gamma, \gamma + 1\}$. By Properties 1 and 2, we know that there is a vertex $v \in V^\star \cup K \cup \text{Dummy}$ such that $v \in \pi^a$. Thus, if there exists a vertex $v \in (K \cup \text{Dummy}) \cap \pi^a$, then since $(a, \gamma) \succ_v (a_\emptyset, 1)$ or $(a, \gamma + 1) \succ_v (a_\emptyset, 1)$ (if $v \in \text{Dummy}$) we have that $|\pi^a| \in \{\gamma, \gamma + 1\}$. Furthermore, by Property 3, we know that if $v \in V^\star \cap \pi^a$ then $|\pi^a| > 1$. However, $(a, \gamma) \succ_v (a_\emptyset, 1)$, thus $|\pi^a| \in \{\gamma, \gamma + 1\}$.

Property 5: *For every $v \in K$, $\pi(v) = a$ and $|\pi^a| = \gamma$.* For a contradiction assume that there exists a vertex $v \in K$ such that $\pi(v) \neq a$. Then, since $G[\pi^a]$ is connected, we have that all the dummy vertices in the path P_v is not in π^a . This implies that $|\pi^a| \leq (|V^\star| + |K| - 1 + (|K| - 1)(n + 1)) < \gamma$. This contradicts Property 4. Finally, since a vertex $v \in K \cap \pi^a$ (in fact every vertex of K is in π^a) and $(a, \gamma) \succ_v (a_\emptyset, 1)$, we have that $|\pi^a| = \gamma$.

Property 6: *For every $v \in \text{Dummy}$, $\pi(v) = a$. Indeed, otherwise consider a vertex $v \in \text{Dummy}$ such that it has a neighbor in π^a (the fact that $K \subseteq \pi^a$ ensures an existence of such a vertex). Then the fact that $(a, \gamma + 1) \succ_v (a_\emptyset, 1)$ together with Property 5 imply that π is not stable, a contradiction.*

Now we are ready to show the reverse direction of the proof. Let $W = \pi^a \cap (V^* \cup K)$. Since $G[\pi^a]$ is connected, we have that $G[W]$ is connected. The last assertion follows from the fact that if we take a spanning tree L of $G[\pi^a]$, then the paths hanging from the vertices of K can be thought of as “long leaves” and by removing leaves we do not disconnect a tree. Furthermore, since $G[V^* \cup K]$ is same as G^* , we have that $G^*[W]$ is connected and contains all the terminals. Finally, for our proof the only thing that remains to show is that $|W| = \ell$. This follows from the fact that $|\pi^a| = \gamma$ and while constructing W we have removed exactly $(n + 1)t$ dummy vertices. This concludes the proof.

◇

Notice that G has maximum degree bounded by 5. We started with a graph G^* of maximum degree 4 and we have not added more than one new neighbor to any vertex. So the degree of any vertex in G is at most 5. Thus, our construction and Lemma 5.2.1 imply the proof of Theorem 10.

Our proof of Theorem 10 is robust in the sense that one can start with a family of graphs on which the STEINER TREE* problem is NP-complete and then do the reduction in a way that we remain inside the family of graphs we started with. For example, it is known that STEINER TREE* remains NP-complete on planar graphs of maximum degree 4 [62], and thus our reduction imply that gNSGA is NP-complete even when there is only one activity and the underlying network is a planar graph of max degree 5.

5.3 An FPT Algorithm for General Graphs

In the last section we established that **gNSGA** remains **NP**-complete even when the number of activities is one and the maximum degree of the underlying network is at most 4. As discussed in the introduction this immediately implies that we can not even have an **XP** algorithm parameterized by p and the maximum degree Δ of the underlying network. However, with an additional parameter t – the maximum size of any group that can form in an **fNsa**—we are able to design an **FPT** algorithm. We use this notation, let $f : A \rightarrow B$ be some function. Given $A' \subseteq A$, the notation $f(A') = b$ indicates that for all $a \in A'$, it holds that $f(a) = b$.

Let $\mathcal{I} = (N, (\succeq_v)_{v \in N}, A, G)$ be an instance of **gNSGA** where the maximum degree of G is at most Δ . Our algorithm has two phases: (a) Separation Phase and (b) Validation Phase. We first outline the phases. We start with the Separation Phase.

Let $\pi : N \rightarrow A$ be a *hypothetical fNsa* for the given input $\mathcal{I}$. For our algorithm we would like to have a function f from $N = V(G)$ to $\{1, 2\}$ with the following property:

- (P1) Let $N' \subseteq N$ be the set of agents who are assigned a non-null activity (that is an activity in A^*) by π . Then, f labels 1 to every agent in N' and labels 2 to all the agents in $N_G(N')$ (neighbors of N' in G that are not in N').

A function f that satisfies the property (P1) with respect to a **fNsa** π is called *nice with respect to π* . Furthermore, a function f from $V(G)$ to $\{1, 2\}$ is called *nice* if f satisfies the property (P1) for *some fNsa* π' for $\mathcal{I}$.

In the validation phase, given a nice function f , we construct a **fNsa** (if it exists), $\pi : N \rightarrow A$, such that all the agents that have been labeled 2 are assigned null activity $a_\emptyset$. In other words, the only agents that get assigned an activity from A^* are those which are labeled 1 by f . It is possible that π assigns $a_\emptyset$ to some agent that have been labeled 1 by f . To construct an assignment π , if it exists, we employ

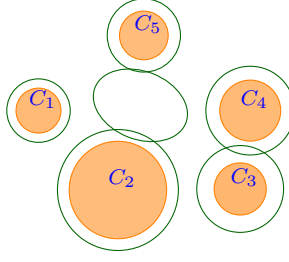


Figure 5.2: Depiction of how a nice function f assigns $\{1, 2\}$ to the vertices of G . Orange colored parts are assigned 1 by f and the white enclosed parts are assigned 2 by f . $C_1, C_2, \dots, C_5$ are the components of $G[f^{-1}(1)]$. For $i \in [5]$, the concentric circle outside C_i is $N(C_i)$. It is guaranteed that f assigns 2 to those vertices.

a dynamic programming procedure. Now we describe both the phases in details.

Seperation Phase. We first show the existence of a small sized family of functions such that given an instance of **gNSGA** on n agents such that it has a **fNsa**, then there exists a function that is nice with respect to this. Towards this we first introduce the notion of n - p^* - q^* -lopsided-universal family. Given a universe U and an integer ℓ by $\binom{U}{\ell}$ we denote all the ℓ -sized subsets of U . We say that a family $\mathcal{F}$ of sets over a universe U of size n is an n - p^* - q^* -lopsided-universal family if for every $A \in \binom{U}{p^*}$ and $B \in \binom{U \setminus A}{q^*}$ there is an $F \in \mathcal{F}$ such that $A \subseteq F$ and $B \cap F = \emptyset$. An alternative definition that is easily seen to be equivalent is that $\mathcal{F}$ is n - p^* - q^* -lopsided-universal family if for every subset $A \in \binom{U}{p^*+q^*}$ and every subset $A' \in \binom{A}{p^*}$, there is an $F \in \mathcal{F}$ such that $F \cap A = A'$.

Lemma 5.3.1. ([51]) *There is an algorithm that given n , p^* and q^* constructs an n - p^* - q^* -lopsided-universal family $\mathcal{F}$ of cardinality $\binom{p^*+q^*}{p^*} \cdot 2^{o(p^*+q^*)} \cdot \log n$ in time $\mathcal{O}(\binom{p^*+q^*}{p^*} \cdot 2^{o(p^*+q^*)} \cdot n \log n)$.*

We now show that there exists a family of functions, $\mathcal{H}(\mathcal{I})$, from N to $\{1, 2\}$ such that if $\mathcal{I}$ has a **fNsa**, π , then there exists a function $f \in \mathcal{H}$ such that f is nice with respect to π . We call the family of functions $\mathcal{H}(\mathcal{I})$, a *nice family* with respect to $\mathcal{I}$. Let the vertex set $N = V(G)$ of the graph G be denoted by $\{v_1, \dots, v_n\}$. We identify the vertex v_i with an integer i and thus we can view the vertex set as $[n]$. To construct the function f , we use Lemma 5.3.1. We apply Lemma 5.3.1 with universe $U = \{1, 2, \dots, n\}$, $p^* = tp$ and $q^* = \Delta tp = \Delta p^*$ and obtain a n - p^* - q^* -lopsided-

universal family $\mathcal{F}$ of size $\binom{p^*+q^*}{p^*} \cdot 2^{o(p^*+q^*)} \cdot \log n$ in time $\mathcal{O}(\binom{p^*+q^*}{p^*} \cdot 2^{o(p^*+q^*)} \cdot n \log n)$.

Given $\mathcal{F}$, we define $\mathcal{H}(\mathcal{I})$ as follows. For every set $X \in \mathcal{F}$, f_X is defined as follows:

$$f_X(x) = \begin{cases} 1 & \text{if } x \in X \\ 2 & \text{otherwise.} \end{cases}$$

Thus, $\mathcal{H}(\mathcal{I}) := \{f_X \mid X \in \mathcal{F}\}$. Now we show that $\mathcal{H}(\mathcal{I})$ is a nice family. Suppose $\mathcal{I}$ has a fNsa π . Let N' be the set of agents that are assigned non-null activity by π and $W = N_G(N')$. Since the size of any group is upper bounded by t we have that $|N'| \leq tp$ and since the maximum degree of G is upper bounded by Δ we have that $|W| \leq \Delta tp$. Now by the property of n - p^* - q^* -lopsided-universal family $\mathcal{F}$, we know that there exists a set $X \in \mathcal{F}$ such that $N' \subseteq X$ and $W \cap X = \emptyset$. By construction the function f_X is nice with respect to π . This brings us to the following lemma.

Lemma 5.3.2. *Let $\mathcal{I}$ be an instance of gNSGA. Then, in time $\mathcal{O}(\binom{p^*+q^*}{p^*} \cdot 2^{o(p^*+q^*)} \cdot n \log n)$ we can construct a nice family $\mathcal{H}(\mathcal{I})$ of size $\binom{p^*+q^*}{p^*} \cdot 2^{o(p^*+q^*)} \cdot \log n$. Here, $p^* = tp$ and $q^* = \Delta p^*$.*

Validation Phase Now, we give an algorithm that given a function $f : N \rightarrow \{1, 2\}$ tests whether f is a nice function. In other words either it *correctly* concludes that f is *not* a nice function or outputs a fNsa, $\pi : N \rightarrow A$, such that f is nice with respect to π .

Lemma 5.3.3. *Let $\mathcal{I}$ be an instance of gNSGA and $f : N \rightarrow \{1, 2\}$ be a function. Then in time, $\mathcal{O}(n4^p(p+1)^{tp})$, we can test whether or not f is nice. Moreover, if f is nice, then in the same time we can output an fNsa, witness to the property that f is nice.*

Proof. Let $\mathcal{I} = (N, (\succeq_v)_{v \in N}, A, G)$ denote an instance of gNSGA. We start by guessing, $A_X \subseteq A^*$, the set of activities that *will not be assigned* to any agent. Next, we describe an algorithm that tests whether f is nice with respect to the guess. That is, does there exists a fNsa, $\pi : N \rightarrow A$, such that π assigns

- (a) every activity in $A^\omega = A^* \setminus A_X$ to at least one agent;
- (b) none of the activities in A_X to any agent;
- (c) $a_\emptyset$ to every vertex labeled 2 by f ;
- (d) $a_\emptyset$ to every vertex in every connected component C of $G[f^{-1}(1)]$ such that $|C| > tp$.
- (e) $a_\emptyset$ to either every vertex in a connected component C of $G[f^{-1}(1)]$ or no vertex in the the connected component C .

Such a **fNsa** is called an A^ω -compatible **fNsa**. A search for A^ω -compatible **fNsa** leads us to the following simple test:

For every $v \in N$ and $a_x \in A_X$, if $f(v) = 2$ or $f(v) = 1$ and it is part of a component C of $G[f^{-1}(1)]$ such that $|C| > tp$ then check whether, $(a_\emptyset, 1) \succeq_v (a_x, 1)$. That is, if f is nice then we know that all the agents that have been labeled 2 by f or are part of large components in $G[f^{-1}(1)]$ must be assigned $a_\emptyset$ by π .

If this test fails then we return that f is not nice with respect to A^ω -compatible **fNsa**. From now onwards we assume that the test succeeded. Also *we modify* f so that every vertex in the connected component C of $G[f^{-1}(1)]$ such that $|C| > tp$ is assigned 2. This just helps simplifying the notations later. Thus, from now onwards we assume that the size of every component of $G[f^{-1}(1)]$ is at most tp .

Let $G_i = G[f^{-1}(i)]$, $i \in [2]$ and let $C_1, \dots, C_r$ be the components of G_1 . For $i \in [r]$, let H_i denote the vertex set $V(G_2) \cup_{j=1}^i C_j$ and Q_i denote the vertex set $V(G_2) \cup C_i$. Furthermore, given a function $h : C_i \rightarrow A^\dagger \cup A_X \cup \{a_\emptyset\}$, where $A^\dagger \subseteq A^\omega$, by $\hat{h}$ we denote the function from Q_i to $A^\dagger \cup A_X \cup \{a_\emptyset\}$ such that $\hat{h}|_{C_i} = h$ and for all vertices in $V(G_2)$ it assigns $a_\emptyset$. Now we describe a dynamic programming (DP)

algorithm that tests whether there exists an A^ω -compatible **fNsa**. Towards this we have a two dimensional table $M[\star, \star]$ with each entry being indexed with $A' \subseteq A^\omega$ and $i \in \{0, \dots, r\}$ and M takes values from $\{0, 1\}$ (i.e. true or false). In particular, we have the following definition.

Definition 5.3.1. $M[A', i]$ is set to 1, if there exists an assignment $\pi_i : H_i \rightarrow A' \cup A_X \cup \{a_\emptyset\}$ such that π_i is an A' -compatible **fNsa** for $(H_i, (\succeq_v)_{v \in H_i}, A' \cup A_X \cup \{a_\emptyset\}, G[H_i])$.

If $M[A^\omega, r] = 1$ then we know that there exists an A^ω -compatible **fNsa** for N , and thus we can conclude that $\mathcal{I}$ is a YES-instance. To compute the DP table we use the following recurrence. We first give the recurrence for the base case.

$$M[A', 0] = \begin{cases} 1 & \text{if } A' = \emptyset \\ 0 & \text{if } A' \neq \emptyset \end{cases}$$

Recursive formula. Now assume that $i \geq 1$ and $A' \subseteq A^\omega$, then we set $M[A', i] = 1$, if there exists a subset $A'' \subseteq A'$ and a function $h : C_i \rightarrow (A' \setminus A'') \cup \{a_\emptyset\}$ such that $M[A'', i-1] = 1$ and $\hat{h}$ is $(A' \setminus A'')$ -compatible **fNsa** for $(Q_i, (\succeq_v)_{v \in Q_i}, (A' \setminus A'') \cup A_X \cup \{a_\emptyset\}, G[Q_i])$. Otherwise, we set $M[A', i] = 0$.

In Lemma [5.3.4](#) we prove the correctness of the recurrence formally. Intuitively, the idea is the following. Every component of G_1 for which we have not fixed an assignment yet has size at most tp . So the number of distinct functions from C_i to a subset of $A^\omega \cup \{a_\emptyset\}$ is upper bounded by $(p+1)^{|C_i|}$. And across all the components it is upper bounded by $\sum_{j=1}^r (p+1)^{|C_j|} \leq n \times (p+1)^{tp}$. By DP we are trying to glue these partial functions (which also includes **fNsa** for H_i) to obtain the desired A^ω -compatible **fNsa**. For the correctness we use the fact that the connected components have a “buffer” around themselves – the neighbors that have been assigned 2. Thus, this allows us to “disjointly” glue partial functions across different components. The

running time follows from the fact that the size of $M[\star, \star]$ is at most $r \cdot 2^{p+1}$ and each entry can be filled in time $\mathcal{O}(2^{p+1}(p+1)^{tp})$. The algorithm can be made to output a witnessing **fNsa**, π , using the usual backtracking technique. This concludes the proof.

◇

Next, we present the proof of the following lemma.

Lemma 5.3.4. *The recurrence correctly computes the table entries $M[A', i]$ for all $A' \subseteq A^\omega$ and H_i , for $i \in \{0, \dots, r\}$.*

Proof. First we prove the base cases, that is, the values for $M[A', j]$ for $A' \subseteq A^\omega$ and $j = 0$ are correct. Observe that, if $j = 0$, then a **fNsa** $\pi : V(G_2) \rightarrow a_\emptyset$. That is a **fNsa** is an assignment that assigns $\{a_\emptyset\}$ to all vertices of H_0 . Hence, if A' is $\emptyset$, we have an A' -compatible **fNsa** for $(H_0, (\succeq_v)_{v \in H_0}, A' \cup A_X \cup \{a_\emptyset\}, G[H_0])$. If $A' \neq \emptyset$, then there is no **fNsa** for $(H_0, (\succeq_v)_{v \in H_0}, A' \cup A_X \cup \{a_\emptyset\}, G[H_0])$. We prove that the value of M defined in Definition 5.3.1 and value given by the recursive formula are equal by proving inequalities in both directions.

For one direction, suppose, there is an assignment $\pi_i : H_i \rightarrow A' \cup A_X \cup \{a_\emptyset\}$ such that π_i is an A' -compatible **fNsa** for $(H_i, (\succeq_v)_{v \in H_i}, A' \cup A_X \cup \{a_\emptyset\}, G[H_i])$. Let $\pi_i|_{H_{i-1}}$ is a solution for $M[A'', i-1]$ for some $A'' \subseteq A'$. Since the given f is *nice*, π_i assigns $a_\emptyset$ to the vertices in G_2 . Therefore, for any $v \in Ng(C_i)$, $\pi_i(v) = a_\emptyset$. Now, π_i either assigns $a_\emptyset$ to all the vertices of Q_i or it assigns some activities from A' to the vertices of C_i . In the former case, π_i is a solution for $M[A', i-1]$. So, $M[A', i-1] = 1$. In the later case, since π_i is stable, $\hat{h}$ is $(A' \setminus A'')$ -compatible **fNsa** for $(Q_i, (\succeq_v)_{v \in Q_i}, (A' \setminus A'') \cup A_X \cup \{a_\emptyset\}, G[Q_i])$. Hence, $M[A', i] = 1$.

For the other direction, suppose $M[A', i]$ evaluates true using the recurrence. We divide it into two cases.

Case 1: If $M[A', i-1]$ is true, then there is a feasible assignment π_{i-1} for $G[H_{i-1}]$.

We claim, π_i defined as follows is a solution for components in $G[H_i]$.

$$\pi_i(v) = \begin{cases} \pi_{i-1}(v) & \text{if } v \in H_{i-1} \\ a_\emptyset & \text{if } v \in Q_i \setminus H_{i-1} \end{cases}$$

Claim 5.3.1. π_i is a fNsa for $(H_i, (\succeq_v)_{v \in H_i}, A' \cup A_X \cup \{a_\emptyset\}, G[H_i])$

Proof. Since π_{i-1} is stable, for any $v \in H_{i-1}$ we have $(\pi_{i-1}(v), |\pi_{i-1_v}|) \succeq_v (a_\emptyset, 1)$ which directly implies, $(\pi_i(v), |\pi_{i_v}|) \succeq_v (a_\emptyset, 1)$. Therefore, we need to show that no vertex in $H_{i-1} \setminus H_i$ has a feasible NS-deviation. Since f is nice, from property (P1), for $v \in Ng(C_i)$, v should be assigned to $a_\emptyset$ by any solution. For any $v \in C_i$, since all the neighbors of v are assigned to $a_\emptyset$ there is no feasible NS-deviation by v . Hence π_i is stable. $\diamond$

Case 2: For some $A'' \subset A'$ the entry $M[A'', i-1]$ is true and there exists an assignment for the component C_i i.e., $h : C_i \rightarrow (A' \setminus A'') \cup \{a_\emptyset\}$ such that $\hat{h}$ is stable in $(Q_i, (\succeq_v)_{v \in Q_i}, (A' \setminus A'') \cup A_X \cup \{a_\emptyset\}, G[Q_i])$. We claim that π_i defined as follows is a solution for $M[A', i]$. As before, let π_{i-1} be the solution for $M[A'', i-1]$. Then,

$$\pi_i(v) = \begin{cases} \pi_{i-1}(v) & \text{if } v \in H_{i-1} \\ \hat{h}(v) & \text{if } v \in Q_i \setminus H_{i-1}. \end{cases}$$

Claim 5.3.2. π_i is a fNsa for $\mathcal{J} = (H_i, (\succeq_v)_{v \in H_i}, A' \cup A_X \cup \{a_\emptyset\}, G[H_i])$.

Proof. Note, both π_{i-1} and $\hat{h}$ are stable. So there can not be a Nash deviation for π_{i-1} or $\hat{h}$. Let $u, v \in E(G[H_i])$ and $u \in H_{i-1}$, $v \in Q_i \setminus H_{i-1}$. Since u and v are not in the same component of G_1 , either $f(u) = 2$ or $f(v) = 2$ or both. Since f is nice, either, $f(u) = a_\emptyset$ or $f(v) = a_\emptyset$ or both. Therefore, u (or v) has no deviation to $\pi_i(v)$ (or $\pi_i(u)$). Hence, π_i is stable, and is a solution for $M[A', i]$. $\diamond$

Hence, we have proved both the directions; and the lemma is proved. $\diamond$

Proof of Theorem 11. Our algorithm works as follows. Given an instance $\mathcal{I}$, it first applies Lemma 5.3.2 and in time $\mathcal{O}\left(\binom{p^*+q^*}{p^*} \cdot 2^{o(p^*+q^*)} \cdot n \log n\right)$ constructs a nice family $\mathcal{H}(\mathcal{I})$ of size $\binom{p^*+q^*}{p^*} \cdot 2^{o(p^*+q^*)} \cdot \log n$. Here, $p^* = tp$ and $q^* = \Delta p^*$. Now for every function $f \in \mathcal{H}(\mathcal{I})$, it applies Lemma 5.3.3 and in time, $\mathcal{O}(n4^p(p+1)^{tp})$, tests whether f is nice. Moreover, if f is nice then in the same time it outputs a **fNsa** witnessing that f is nice. This concludes the description of the algorithm. The correctness of the algorithm follows from Lemmas 5.3.2 and 5.3.3. The running time of the algorithm is upper bounded by the size of $\mathcal{H}(\mathcal{I})$ and the time taken to test niceness. Thus, it is upper bounded by $\mathcal{O}(n4^p(p+1)^{tp} \cdot \binom{p^*+q^*}{p^*} \cdot 2^{o(p^*+q^*)} \cdot n \log n)$. A simplification of this gives the desired running time. $\diamond$

Using Lemma 5.3.3 we can complete the proof of Theorem 11.

5.4 FPT Algorithm for Networks of Bounded Treewidth

In this section, we prove Theorem 12. First, recall in time $2^{\text{tw}} \cdot n$ we can obtain a nice tree decomposition of the network G whose width is $\eta = \mathcal{O}(\text{tw})$. In what follows, we let (T, β) denote such a decomposition; and then apply the method of dynamic programming as described in the following subsections.

The DP Table. Let us begin by describing our DP table, which we denote by **M**. Each entry of this table is of the form $[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ for any choice of arguments as follows.

- A node $x \in V(T)$, which indicates that the partial solutions corresponding to the current entry would only assign activities only to those vertices that belong to $\gamma(x)$.

- A subset of activities $B \subseteq A$ such that $a_\emptyset \in B$, which indicates that only activities from B can be assigned to those vertices that belong to $\gamma(x)$. Let us denote $B^\star = B \setminus \{a_\emptyset\}$.
- A subset of activities $\widehat{B} \subseteq B$ such that $a_\emptyset \in \widehat{B}$, which indicates that each activity in $\widehat{B}^\star = \widehat{B} \setminus \{a_\emptyset\}$ has been assigned to at least one vertex in $\gamma(x)$.
- A function $f_{act} : \beta(x) \rightarrow \widehat{B}$, which specifies exactly how to assign activities to those vertices that belong to $\beta(x)$.
- A function $f_{tot} : \beta(x) \rightarrow [n]$, which specifies exactly, for every vertex v in $\beta(x)$, how many vertices in total (i.e. in $V(G)$) should participate in the same activity as v . Here, the void activity is an exception as for any vertex v assigned to the void activity, we would demand that $f_{tot}(v)$ is set to 1, irrespective of how many vertices participate in this activity.
- A function $f_{cur} : \beta(x) \rightarrow [n]$, which specifies exactly, for every vertex v in $\beta(x)$, how many vertices have so far (i.e. in $\gamma(x)$) have been determined to participate in the same activity as v . Here, the void activity is an exception in the same sense as the one specified for f_{tot} .
- A partition P of $\beta(x)$, which is interpreted as follows. For the sake of clarity, we let $f_P : \beta(x) \rightarrow 2^{\beta(x)}$ denote the function that, for every vertex v in $\beta(x)$, assigns the set in P to which v belongs. Then, the information captured by P is the specification that for every pair of (distinct) vertices in $\beta(x)$, u and v , satisfying $f_P(u) = f_P(v)$, u and v participate in the same non-void activity, and there exists a path in $G[\gamma(x)]$ between u and v such that all of the vertices of this path participate in the same activity as u and v .

We would say that an entry $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ is *legal* if the following conditions are satisfied.

1. For all $u, v \in \beta(x)$ such that $f_{act}(u) = f_{act}(v)$, it holds that $f_{tot}(u) = f_{tot}(v)$ and $f_{cur}(u) = f_{cur}(v)$.
2. For all $v \in \beta(x)$, it holds that $f_{cur}(v) \leq f_{tot}(v)$.
3. For all (distinct) $u, v \in \beta(x)$ such that $f_P(u) = f_P(v)$, it holds that $f_{act}(u) = f_{act}(v) \neq a_\emptyset$.
4. For all $u, v \in \beta(x)$ such that $f_{cur}(u) = f_{tot}(u)$ and $f_{act}(u) = f_{act}(v)$, it holds that $f_P(u) = f_P(v)$.
5. For all $v \in \beta(x)$ such that $f_{act}(v) = a_\emptyset$, it holds that $f_{tot}(v) = 1$.
6. For all $v \in \beta(x)$ such that $f_{act}(v) \neq a_\emptyset$, it holds that $(f_{act}(v), f_{tot}(v)) \succ_v (a_\emptyset, 1)$.
7. For all $v \in \beta(x)$, it holds that $(f_{act}(v), f_{tot}(v)) \succ_v (a, 1)$ for every $a \in A \setminus B$.
8. For all $v \in \beta(x)$, there does not exist $u \in N_{G[\beta(x)]}(v)$ such that $f_{act}(v) \neq f_{act}(u)$, $f_{act}(u) \neq a_\emptyset$ and $(f_{act}(u), f_{tot}^\pi(u) + 1) \succ_v (\pi(v), f_{tot}^\pi(v))$.

Formally, we would say that the table $\mathbf{M}$ has been *computed correctly* if each of its entries $[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ stores either 0 or 1, and it stores 1 if and only if this entry is legal and there exists an assignment $\pi : \gamma(x) \rightarrow B$ that satisfies the following conditions. Here, for all $v \in \gamma(x)$, we would define $f_{tot}^\pi(v) = |\pi_v|$ if $\pi_v \cap \beta(x) = \emptyset$, and $f_{tot}^\pi(v) = f_{tot}(u)$ for any $u \in \pi_v \cap \beta(x)$ otherwise. Since the entry is legal and in particular satisfies Condition 1, this notation is well defined.

- I. For all $a \in B^*$, there exists $v \in \gamma(x)$ such that $\pi(v) = a$.
- II. For all $v \in \beta(x)$, it holds that $\pi(v) = f_{act}(v)$.
- III. For all $v \in \beta(x)$, it holds that $|\pi_v| = f_{cur}(v)$.
- IV. For all $v \in \gamma(x)$ such that $\pi(v) \neq a_\emptyset$, it holds that $(\pi(v), f_{tot}^\pi(v)) \succ_v (a_\emptyset, 1)$.
- V. For all $v \in \gamma(x)$, it holds that $(\pi(v), f_{tot}^\pi(v)) \succ_v (a, 1)$ for every $a \in A \setminus B$.

- VI. For all $v \in \gamma(x)$, there does not exist $u \in N_{G[\gamma(x)]}(v)$ such that $f_{act}(v) \neq f_{act}(u)$, $f_{act}(u) \neq a_\emptyset$ and $(f_{act}(u), f_{tot}^\pi(u) + 1) \succ_v (\pi(v), f_{tot}^\pi(v))$.
- VII. For all $u, v \in \beta(x)$ such that $f_P(u) = f_P(v)$, it holds that there exists a path in $G[\pi_v]$ between v and u .
- VIII. For all $v \in \gamma(x)$ such that $\pi_v \cap \beta(x) = \emptyset$, it holds that π_v is a feasible coalition.

We call an assignment as specified above a *witness* for $[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$.

5.4.1 Computation

The order of the computation of the entries corresponds to a postorder traversal with respect to the first argument, where the order between entries having the same first argument is arbitrary. For every entry that is not legal, we simply assign 0. In what follows, we present the computation for legal entries $[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$, considering different cases according to the type of the node x .

Leaf Node. Notice that as in this case $\beta(x) = \emptyset$, the domains of f_{act} , f_{tot} and f_{cur} are all empty, and $P = ()$. We set $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ to 1 if $\widehat{B}^\star = \emptyset$, and to 0 otherwise.

Forget Node. Let y denote the child of x in T . Recall that $|\beta(y) \setminus \beta(x)| = 1$. Moreover, let w denote the unique vertex in $\beta(y) \setminus \beta(x)$. Then, let $\mathcal{F}_{act}$ denote the set of all functions $f : \beta(y) \rightarrow \widehat{B}$ such that $f|_{\beta(x)} = f_{act}$, let $\mathcal{F}_{tot}$ denote the set of all functions $f : \beta(y) \rightarrow \widehat{B}$ such that $f|_{\beta(x)} = f_{tot}$, and let $\mathcal{F}_{cur}$ denote the set of all functions $f : \beta(y) \rightarrow \widehat{B}$ such that $f|_{\beta(x)} = f_{cur}$. Moreover, let $\mathcal{P}$ denote the set of every partition Q of $\beta(y)$ such that for all $u, v \in \beta(x)$, it holds that $f_P(u) = f_P(v)$ if and only if $f_Q(u) = f_Q(v)$. Now, we compute the entry $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ as follows.

1. If there exist $f'_{act} \in \mathcal{F}_{act}$, $f'_{tot} \in \mathcal{F}_{tot}$, $f'_{cur} \in \mathcal{F}_{cur}$ and $Q \in \mathcal{Q}$ such that the three following conditions hold, then we set $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ to 1.

(a) $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q] = 1$.

(b) If $f'_{act}(w) \neq a_\emptyset$ and there does not exist a vertex $v \in \beta(x)$ such that $f_{act}(v) = f'_{act}(w)$, then $f'_{tot}(w) = f'_{cur}(w)$.

(c) If $f'_{act}(w) \neq a_\emptyset$ and there does not exist a vertex $v \in \beta(x)$ such that $f_P(v) = f'_P(w)$, then there does not exist a vertex $v \in \beta(x)$ such that $f_{act}(v) = f'_{act}(w)$.

2. Otherwise, we set $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ to 0.

Introduce Node. Let y denote the child of x in T . Recall that $|\beta(x) \setminus \beta(y)| = 1$. Moreover, let w denote the unique vertex in $\beta(x) \setminus \beta(y)$. First, if there exists $v \in \beta(y)$ satisfying $f_P(v) = f_P(w)$ such that there does not exist $u \in N_{G[\beta(x)]}(w)$ for which $f_P(u) = f_P(v)$, then we set $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ to 0. Next, suppose that the above condition is false.

Denote $f'_{act} = f_{act}|_{\beta(y)}$, $f'_{tot} = f_{tot}|_{\beta(y)}$, $f'_{cur} = f_{cur}|_{\beta(y)}$. Moreover, let $\mathcal{P}$ denote the set of partitions Q of $\beta(y)$ that satisfy the following two conditions.

1. For all $u, v \in \beta(y)$ such that $f_P(u) = f_P(v) \neq f_P(w)$, it holds that $f_P(u) = f_P(v)$ if and only if $f_Q(u) = f_Q(v)$.
2. For all $v \in \beta(y)$ such that $f_P(v) = f_P(w)$ and $v \notin N_{G[\beta(x)]}(w)$, there exists $u \in N_{G[\beta(x)]}(w)$ such that $f_Q(v) = f_Q(u)$.

Now, we compute $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ as follows.

1. If $f_{act}(w) = a_\emptyset$ or there exists $v \in \beta(y)$ such that $f_{act}(v) = f_{act}(w)$, then $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ is set to 1 if there exists $Q \in \mathcal{P}$ such that $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q] = 1$, and to 0 otherwise.

2. Otherwise (if $f_{act}(w) \neq a_\emptyset$ and there does not exist $v \in \beta(y)$ such that $f_{act}(v) = f_{act}(w)$), then $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ is set to 1 if there exists $Q \in \mathcal{P}$ such that $\mathbf{M}[y, B, \widehat{B} \setminus \{f_{act}(w)\}, f'_{act}, f'_{tot}, f'_{cur}, Q] = 1$, and to 0 otherwise.

Join Node. Let y and z denote the children of x in T . Recall that $\beta(x) = \beta(y) = \beta(z)$. Let $\widehat{\mathcal{B}} = \{B' \in 2^{\widehat{B}} : a_\emptyset \in B'\}$. We say that two sets $B', B'' \in \widehat{\mathcal{B}}$ are *compatible* if $B' = (\widehat{B} \setminus B'') \cup \{f_{act}(v) : v \in \beta(x)\}$. Let $\mathcal{F}_{cur}$ be the set of all functions $f : \beta(x) \rightarrow [n]$ such that for all $v \in \beta(x)$, it holds that $f(v) \leq f_{cur}(v)$. We say that two function $f, f' \in \mathcal{F}_{cur}$ are *compatible* if for all $v \in \beta(x)$, it holds that $f(v) + f'(v) = f_{cur}(v) + |\{u \in \beta(x) : f_{cur}(u) = f_{cur}(v)\}|$. Let $\mathcal{P}$ denote the set of all partitions Q of $\beta(x)$ such that for all $u, v \in \beta(x)$, if $f_Q(u) = f_Q(v)$, then $f_P(u) = f_P(v)$. We say that two partitions $Q, Q' \in \mathcal{P}$ are *compatible* if for all $u, v \in \beta(x)$ such that $f_P(u) = f_P(v)$, at least one of the two following conditions holds: **(i)** $f_Q(u) = f_Q(v)$; **(ii)** $f_{Q'}(u) = f_{Q'}(v)$.

Now, we compute $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ as follows. If there exist compatible $B', B'' \in \widehat{\mathcal{B}}$, compatible $f, f' \in \mathcal{F}_{cur}$ and compatible $Q, Q' \in \mathcal{P}$ such that $\mathbf{M}[x, B', \widehat{B}, f_{act}, f_{tot}, f, Q] = \mathbf{M}[x, B'', \widehat{B}, f_{act}, f_{tot}, f', Q'] = 1$, then set $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ to 1. Otherwise, set this entry to 0.

5.4.2 Correctness

First, we note that the following observation clearly holds.

Observation 5.4.1. *The computation of every entry of $\mathbf{M}$ that is not legal is correct and can be performed in polynomial time.*

Now, by Condition in the definition of a correct computation, we also have the following observation.

Observation 5.4.2. *The computation of every legal entry of $\mathbf{M}$ whose first argument is a leaf of T is correct and can be performed in polynomial time.*

Let us proceed to consider the cases of a forget node, an introduce node and a join node. Here, when we consider some specific entry, we would assume that all entries computed before it have been computed correctly (that will be our inductive hypothesis).

Lemma 5.4.1. (Forget node) *The computation of every legal entry $[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ of $\mathbf{M}$ whose first argument x is a forget node of T is correct, and can be performed in polynomial time.*

Proof. Let us first observe that $|\mathcal{F}_{act}| = p+1$, $|\mathcal{F}_{tot}| = n$, $|\mathcal{F}_{cur}| = n$ and $|\mathcal{P}| \leq \mathbf{tw} + 1$, and therefore it is clear that the computation can be performed in polynomial time. Next, we consider the correctness of the computation.

In the first direction, suppose that a correct computation of the entry $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ should set it to 1. Then, there exists an assignment $\pi : \gamma(x) \rightarrow B$ that is a witness for $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$. To show that we set $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ to 1, we need to show that there exist $f'_{act} \in \mathcal{F}_{act}$, $f'_{tot} \in \mathcal{F}_{tot}$, $f'_{cur} \in \mathcal{F}_{cur}$ and $Q \in \mathcal{Q}$ such that the three following conditions hold.

1. $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q] = 1$.
2. If $f'_{act}(w) \neq a_\emptyset$ and there does not exist a vertex $v \in \beta(x)$ such that $f_{act}(v) = f'_{act}(w)$, then $f'_{tot}(w) = f'_{cur}(w)$.
3. If $f'_{act}(w) \neq a_\emptyset$ and there does not exist a vertex $v \in \beta(x)$ such that $f_P(v) = f'_{act}(w)$, then there does not exist a vertex $v \in \beta(x)$ such that $f_{act}(v) = f'_{act}(w)$.

First, define f'_{act} as the (unique) function in $\mathcal{F}_{act}$ that satisfies $f'_{act}(w) = \pi(w)$. Second, define f'_{tot} as the (unique) function in $\mathcal{F}_{tot}$ that satisfies $f'_{tot}(w) = f'_{tot}(v)$

for any $v \in \beta(x)$ satisfying $f_{act}(v) = \pi(w)$ if such a vertex v exists (since $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ is legal, and in particular satisfies Condition [1](#) (pg [120](#)) of that definition, this is well defined), and otherwise it satisfies $f'_{tot}(w) = |\pi_w|$. Third, define f'_{cur} as the (unique) function in $\mathcal{F}_{act}$ that satisfies $f'_{cur}(w) = |\pi_w|$. Finally, define Q as a partition in $\mathcal{P}$ such that if there exists $v \in \pi_w$ such that there exists a path in $G[\pi_w]$ between v and w , then arbitrarily choose such a vertex v and define $f_Q(w) = f_Q(v)$, and otherwise let w form a singleton set in Q . To conclude the proof of the first direction, we next show that the three conditions above are satisfied with respect to $f'_{act} \in \mathcal{F}_{act}, f'_{tot} \in \mathcal{F}_{tot}, f'_{cur} \in \mathcal{F}_{cur}$ and $Q \in \mathcal{Q}$ which we have just defined.

1. Since $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ is legal and π is a witness for $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$, our definition of $f'_{act}, f'_{tot}, f'_{cur}$ and Q directly implies that $M[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$ is legal. Thus, by the inductive hypothesis, it is sufficient to show that there exists π' that satisfies Conditions [I](#) to [VIII](#) required to be a witness for $M[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$. We claim that such π' is simply π . Indeed, the satisfaction of each individual condition by π' with respect to $M[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$ directly follows from the fact that π satisfies that condition with respect to $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ and from our definition of $f'_{act}, f'_{tot}, f'_{cur}$ and Q .
2. The satisfaction of this condition directly follows from our definition of f'_{cur} and f'_{tot} .
3. The satisfaction of this condition directly follows from our definition of f'_{act} and Q , and since π satisfies Condition [VIII](#) required for it to be a witness for $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$.

In the second direction, suppose that we set the entry $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ to 1. Then, there exist $f'_{act} \in \mathcal{F}_{act}, f'_{tot} \in \mathcal{F}_{tot}, f'_{cur} \in \mathcal{F}_{cur}$ and $Q \in \mathcal{Q}$ such that the three following conditions hold.

1. $M[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q] = 1$.
2. If $f'_{act}(w) \neq a_\emptyset$ and there does not exist a vertex $v \in \beta(x)$ such that $f_{act}(v) = f'_{act}(w)$, then $f'_{tot}(w) = f'_{cur}(w)$.
3. If $f'_{act}(w) \neq a_\emptyset$ and there does not exist a vertex $v \in \beta(x)$ such that $f_P(v) = f'_{act}(w)$, then there does not exist a vertex $v \in \beta(x)$ such that $f_{act}(v) = f'_{act}(w)$.

From the first condition above, by the inductive hypothesis, we have that $M[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$ is legal and there exists π' that is a witness for this entry. Define $\pi = \pi'|_{\beta(x)}$. We claim that π is a witness for $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$. By the second condition above and since π' is a witness for $M[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$, we have that $(\pi(w), f_{tot}^\pi(w)) \succ_w (a, 1)$ for every $a \in A \setminus B$, that if $\pi(w) \neq a_\emptyset$, it holds that $(\pi(v), f_{tot}^\pi(v)) \succ_v (a_\emptyset, 1)$, and that there does not exist $u \in N_{G[\gamma(x)]}(w)$ such that $f_{act}(w) \neq f_{act}(u)$, $f_{act}(u) \neq a_\emptyset$ and $(f_{act}(u), f_{tot}^\pi(u) + 1) \succ_w (\pi(w), f_{tot}^\pi(w))$. Moreover, by the third condition above and since π' is a witness for $M[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$, we have that if $\pi_w \cap \beta(x) = \emptyset$, then π_w is a feasible coalition. In light of these arguments, the satisfaction of each of the required conditions by π to be a witness for $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ directly follows from the fact that π' satisfies that condition with respect to $M[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$. This concludes the proof. $\diamond$

Lemma 5.4.2. (Introduce Node) *The computation of every legal entry $[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ of M whose first argument x is an introduce node of T is correct; and can be performed in time $\mathcal{O}(\mathbf{tw}^{\mathcal{O}(\mathbf{tw})} \cdot (n + p)^{\mathcal{O}(1)})$.*

Proof. Let us first observe that $|\mathcal{P}| = \mathbf{tw}^{\mathcal{O}(\mathbf{tw})}$, and therefore it is clear that the computation can be performed in time $\mathcal{O}(\mathbf{tw}^{\mathcal{O}(\mathbf{tw})} \cdot (n + p)^{\mathcal{O}(1)})$. Next, we consider the correctness of the computation.

In the first direction, suppose that a correct computation of the entry $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ should set it to 1. Then, there exists an assignment π :

$\gamma(x) \rightarrow B$ that is a witness for $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$. To show that we set $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ to 1, we first need to show that for all $v \in \beta(y)$ satisfying $f_P(v) = f_P(w)$, there exists $u \in N_{G[\beta(x)]}(w)$ for which $f_P(u) = f_P(v)$. However, this follows directly from the fact that π satisfy Condition VII. of being a witness for $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ and from the definition of a tree decomposition. Next, it remains to show that the appropriate condition among the following two conditions is satisfied.

1. If $f_{act}(w) = a_\emptyset$ or there exists $v \in \beta(y)$ such that $f_{act}(v) = f_{act}(w)$, then there exists $Q \in \mathcal{P}$ such that $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q] = 1$.
2. Otherwise (if $f_{act}(w) \neq a_\emptyset$ and there does not exist $v \in \beta(y)$ such that $f_{act}(v) = f_{act}(w)$), then there exists $Q \in \mathcal{P}$ such that $\mathbf{M}[y, B, \widehat{B} \setminus \{f_{act}(w)\}, f'_{act}, f'_{tot}, f'_{cur}, Q] = 1$.

Define Q as the partition of $\beta(y)$ that satisfies the following two conditions.

1. For all $u, v \in \beta(y)$ such that $f_P(u) = f_P(v) \neq f_P(w)$, it holds that $f_P(u) = f_P(v)$ if and only if $f_Q(u) = f_Q(v)$.
2. For all $u, v \in \beta(y)$ such that $f_P(u) = f_P(v) = f_P(w)$, it holds that $f_Q(u) = f_Q(v)$ if and only if $G[\pi_u \setminus \{w\}]$ contains a path between u and v .

Notice that since π satisfies Condition VII. as a witness for $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ and by the definition of a tree decomposition, we have that for all $v \in \beta(y)$ such that $f_P(v) = f_P(w)$ and $v \notin N_{G[\beta(x)]}(w)$, there exists $u \in N_{G[\beta(x)]}(w)$ such that $f_Q(v) = f_Q(u)$. Hence, $Q \in \mathcal{P}$. Let us now turn to show that the appropriate required condition is satisfied.

1. First, suppose that $f_{act}(w) = a_\emptyset$ or there exists $v \in \beta(y)$ such that $f_{act}(v) = f_{act}(w)$. By the inductive hypothesis, it is sufficient to show that there exists

π' that is a witness for $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$. We claim that such π' is simply $\pi|_{\beta(y)}$. Thus, the satisfaction of each individual condition by $\pi|_{\beta(y)}$ with respect to $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$ directly follows from the fact that π satisfies that condition with respect to $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ and from our definition of $f'_{act}, f'_{tot}, f'_{cur}$ and Q , where the satisfaction of Condition [I.](#) also relies on the condition of the current case.

2. Second, suppose that $f_{act}(w) \neq a_\emptyset$ and there does not exist $v \in \beta(y)$ such that $f_{act}(v) = f_{act}(w)$. Then, since π satisfies Conditions [I.](#), [II.](#) and [VIII.](#) of being a witness for $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$, and by the definition of a tree decomposition, we have that there does not exist $v \in \gamma(y)$ such that $\pi(v) = f_{act}(w)$. Thus, the satisfaction of each individual condition by $\pi|_{\beta(y)}$ with respect to $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$ directly follows from the fact that π satisfies that condition with respect to $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ and from our definition of $f'_{act}, f'_{tot}, f'_{cur}$ and Q .

In the second direction, suppose that we set the entry $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ to 1. Then, we have that $(*)$ for all $v \in \beta(y)$ satisfying $f_P(v) = f_P(w)$, there exists $u \in N_{G[\beta(x)]}(w)$ for which $f_P(u) = f_P(v)$. Moreover, the appropriate condition among the following two conditions is satisfied. (Here, we would implicitly rely on the fact that the current entry $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ is legal.)

1. If $f_{act}(w) = a_\emptyset$ or there exists $v \in \beta(y)$ such that $f_{act}(v) = f_{act}(w)$, then there exists $Q \in \mathcal{P}$ such that $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q] = 1$. By the inductive hypothesis, there exists π' that is a witness for $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q] = 1$. We define π as the assignment such that $\pi|_{\beta(y)} = \pi'$ and $\pi(w) = f_{act}(w)$. We claim that π is a witness for $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$. Since $f_{act}(w) = a_\emptyset$ or there exists $v \in \beta(y)$ such that $f_{act}(v) = f_{act}(w)$, and π' satisfies Condition [I.](#) as a witness for $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$, we have that π satisfies this condition with respect to $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$. Moreover, by $(*)$,

since $Q \in \mathcal{P}$ and because π' satisfies Conditions VII. and VIII. as a witness for $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$, we have that π satisfies these conditions with respect to $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$. The satisfaction of each remaining individual condition by π with respect to $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ directly follows from the fact that π' satisfies that condition with respect to $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$ and from our definition of $f'_{act}, f'_{tot}, f'_{cur}$ and Q .

2. Otherwise (if $f_{act}(w) \neq a_\emptyset$ and there does not exist $v \in \beta(y)$ such that $f_{act}(v) = f_{act}(w)$), then there exists $Q \in \mathcal{P}$ such that $\mathbf{M}[y, B, \widehat{B} \setminus \{f_{act}(w)\}, f'_{act}, f'_{tot}, f'_{cur}, Q] = 1$. By the inductive hypothesis, there exists π' that is a witness for $\mathbf{M}[y, B, \widehat{B} \setminus \{f_{act}(w)\}, f'_{act}, f'_{tot}, f'_{cur}, Q] = 1$. We define π as the assignment such that $\pi|_{\beta(y)} = \pi'$ and $\pi(w) = f_{act}(w)$. We claim that π is a witness for $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$. Since $f_{act}(w) \neq a_\emptyset$ and there does not exist $v \in \beta(y)$ such that $f_{act}(v) = f_{act}(w)$, and π' satisfies Conditions I. and VIII. as a witness for $\mathbf{M}[y, B, \widehat{B}, f'_{act}, f'_{tot}, f'_{cur}, Q]$, the definition of a tree decomposition implies that π satisfies this condition with respect to $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$. The rest of the argument for this case proceeds exactly as in the previous case.

This concludes the proof.

◇

Lemma 5.4.3. (Join Node) *The computation of every legal entry*

$[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ of $\mathbf{M}$ whose first argument x is a join node of T is correct; and can be performed in time $\mathcal{O}(2^{|\widehat{B}|} \cdot \mathbf{tw}^{\mathcal{O}(\mathbf{tw})}(n + p)^{\mathcal{O}(1)})$.

Proof. Let us first observe that the number of pairs of compatible sets in $\widehat{\mathcal{B}}$ is upper bounded by $|\widehat{\mathcal{B}}| = 2^{|\widehat{B}|}$. Moreover, $|\mathcal{F}_{cur}| = n^{\mathcal{O}(\mathbf{tw})}$ and $|\mathcal{P}| = n^{\mathcal{O}(\mathbf{tw})}$, and therefore it is clear that the computation can be performed in time $\mathcal{O}(2^{|\widehat{B}|} \cdot \mathbf{tw}^{\mathcal{O}(\mathbf{tw})}(n + p)^{\mathcal{O}(1)})$.

Next, we consider the correctness of the computation.

In the first direction, suppose that a correct computation of the entry $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ should set it to 1. Then, there exists an assignment $\pi : \gamma(x) \rightarrow B$ that is a witness for $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$. To show that we set $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ to 1, we need to show that there exist compatible $B', B'' \in \widehat{\mathcal{B}}$, compatible $f, f' \in \mathcal{F}_{cur}$ and compatible $Q, Q' \in \mathcal{P}$ such that $M[x, B', \widehat{B}, f_{act}, f_{tot}, f, Q] = M[x, B'', \widehat{B}, f_{act}, f_{tot}, f', Q'] = 1$.

For this purpose, denote $\pi' = \pi|_{\beta(y)}$ and $\pi'' = \pi|_{\beta(z)}$. Moreover, denote the image of π' by B' and the image of π'' by B'' . By the definition of a tree decomposition, and since π satisfies Conditions [I.](#), [II.](#), [VII.](#) and [VIII.](#) as a witness for $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$, we have that B' and B'' are compatible sets in $\widehat{\mathcal{B}}$. Now, define $f : \beta(y) \rightarrow [n]$ by setting $f(v) = |\pi'_v|$ for all $v \in \beta(y)$, and define $f' : \beta(y) \rightarrow [n]$ by setting $f'(v) = |\pi''_v|$ for all $v \in \beta(y)$. By the definition of a tree decomposition, and since π satisfies Conditions [II.](#) and [III.](#) as a witness for $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$, we have that f' and f'' are compatible functions in $\mathcal{F}_{act}$. Next, let Q (resp. Q') denote the partition of $\beta(x)$ such that for all $u, v \in \beta(x)$, it holds that $f_Q(u) = f_Q(v)$ if and only if $f_P(u) = f_P(v)$ and there exists a path between u and v in $G[\pi'_u]$ (resp. $G[\pi''_v]$). By the definition of a tree decomposition, since π satisfies Conditions [VII.](#) and [VIII.](#) as a witness for $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$, we have that Q and Q' are compatible partitions in $\mathcal{P}$. Finally, notice that the satisfaction of each individual condition by π' to be a witness for $M[x, B', \widehat{B}, f_{act}, f_{tot}, f, Q]$ directly follows from the fact that π satisfies that condition with respect to $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ and from our definition of B', f and Q . Similarly, we derive that π'' is a witness for $M[x, B'', \widehat{B}, f_{act}, f_{tot}, f', Q']$. Thus, by the inductive hypothesis, $M[x, B', \widehat{B}, f_{act}, f_{tot}, f, Q] = M[x, B'', \widehat{B}, f_{act}, f_{tot}, f', Q'] = 1$. This concludes the proof of the forward direction.

In the second direction, suppose that we set the entry $M[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ to 1. Then, there exist compatible $B', B'' \in \widehat{\mathcal{B}}$, compatible $f, f' \in \mathcal{F}_{cur}$ and

compatible $Q, Q' \in \mathcal{P}$ such that $\mathbf{M}[x, B', \widehat{B}, f_{act}, f_{tot}, f, Q] = \mathbf{M}[x, B'', \widehat{B}, f_{act}, f_{tot}, f', Q'] = 1$. By the inductive hypothesis, there exist a witness π' for $\mathbf{M}[x, B', \widehat{B}, f_{act}, f_{tot}, f, Q]$ and a witness π'' for $\mathbf{M}[x, B'', \widehat{B}, f_{act}, f_{tot}, f', Q']$. Since π' and π'' satisfy Condition II. as witnesses, it is well defined to let $\pi : \gamma(x) \rightarrow \widehat{B}$ denote the assignment such that for all $v \in \gamma(y)$, $\pi(v) = \pi'(v)$, and for all $v \in \gamma(z)$, $\pi(v) = \pi''(v)$. Since B' and B'' are compatible, f and f' are compatible and Q and Q'' are compatible, we have that as π' and π'' are witnesses and by the definition of a tree decomposition, it holds that π is a witness for $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$. This concludes the proof. $\diamond$

We now turn to prove the correctness and running time of the dynamic programming computation.

Lemma 5.4.4. *The table $\mathbf{M}$ is correctly filled in time $\mathcal{O}(4^p \cdot (n + p)^{\mathcal{O}(\mathbf{tw})})$.*

Proof. The statement that the table $\mathbf{M}$ is filled correctly directly follows from Observation 5.4.1 and 5.4.2 and from Lemmata 5.4.1, 5.4.2 and 5.4.3. Let us now turn to analyze the running time. For this purpose, from Observation 5.4.1 and 5.4.2 and from Lemmata 5.4.1, 5.4.2 and 5.4.3, we have that the running time to compute an entry $\mathbf{M}[x, B, \widehat{B}, f_{act}, f_{tot}, f_{cur}, P]$ is upper bounded by $\mathcal{O}(2^{|\widehat{B}|} \cdot \mathbf{tw}^{\mathcal{O}(\mathbf{tw})}(n + p)^{\mathcal{O}(1)})$. Observe that for every fixed B and $\widehat{B}$, the number of entries of $\mathbf{M}$ with B and $\widehat{B}$ as the second and third arguments is upper bounded by $(n + p)^{\mathcal{O}(\mathbf{tw})}$. Thus, since $|A| = p + 1$, the total running time is upper bounded by

$$\sum_{B \subseteq A} \sum_{\widehat{B} \subseteq B} 2^{|\widehat{B}|} \cdot \mathbf{tw}^{\mathcal{O}(\mathbf{tw})}(n + p)^{\mathcal{O}(1)} = 4^p \cdot (n + p)^{\mathcal{O}(\mathbf{tw})}.$$

This concludes the proof of the lemma. $\diamond$

Proof of Theorem 12. Our algorithm computes the tree decomposition (T, β) and then fills the table $\mathbf{M}$. Finally, it determines that the input is a yes-instance if and only if there exists $B \subseteq A$ containing $a_\emptyset$ s.t. $\mathbf{M}[r, B, B, h, h, h, ()] = 1$, where r

is the root of T and h is the function $h : \emptyset \rightarrow \emptyset$. In this context, we remind that as (T, β) is a nice tree decomposition, we have that $\beta(r) = \emptyset$.

By Lemma 5.4.4 and as (T, β) is computed in time $2^{\mathcal{O}(\mathbf{tw})} \cdot n$, the total running time is bounded by $\mathcal{O}(4^p \cdot (n + p)^{\mathcal{O}(\mathbf{tw})})$. Now, for all $B \subseteq A$ containing $a_\emptyset$, by Lemma 5.4.4, Conditions I, IV, V, VI and VIII, $\mathbf{M}[r, B, B, h, h, h, ()] = 1$ holds if and only if there exists an **fNsa** whose image is precisely B (that is a witness for $[r, B, B, h, h, h, ()]$). Hence, we conclude that our algorithm is correct. $\diamond$

5.5 Conclusion

In this chapter, we resolved several open problems posed in [80]. It would be interesting to determine whether **gNSGA** is FPT when parameterized by $\mathbf{tw}$ and the size of the set of activities p . That is, does there exist an algorithm with running time $f(p, \mathbf{tw}) \cdot n^{\mathcal{O}(1)}$. Another direction to pursue would be to determine the kernelization complexity of **gNSGA** when parameterized by p and the underlying network is a path, or a star or a tree. Finally, finding parameters which could make **gNSGA** tractable on broader classes of graph is an interesting research avenue.

Selection Problems

Chapter 6

Rainbow Matching

In the first part of the thesis we studied various allocation problems and mainly two notions of stability. In this part, instead of seeking an assignment from one set to another, we aim to select a subset of elements which satisfy some criteria.

The classical notion of matching has been extensively studied for several decades in the area of Combinatorial Optimization [39, 113]. Given an undirected graph G , a set of edges is called a *matching* if the edges are pairwise non-adjacent. That is, no two edges share a common vertex. In the MAXIMUM MATCHING problem, the objective is to find a matching of maximum size. The first polynomial time algorithm for MAXIMUM MATCHING was given by Edmonds [39] in his classic paper *Paths, Trees and Flowers*. It is important to remark that this is the paper which underlined the importance of study of polynomial time algorithms for the first time. After a series of improvements, the current fastest algorithm for MAXIMUM MATCHING was given by Micali and Vazirani and it runs in time $\mathcal{O}(m\sqrt{n})$ [125]. However, finding a matching that satisfies some additional constraints often immediately becomes NP-complete, where three notable examples are MINIMUM MAXIMAL MATCHING [157], INDUCED MATCHING [148] and MULTIPLE CHOICE MATCHING [88]. In this chapter, we study the NP-hard variant of MAXIMUM MATCHING called MULTIPLE CHOICE

MATCHING from the viewpoint of parameterized complexity.

The MULTIPLE CHOICE MATCHING problem, also called RAINBOW MATCHING, is one of the NP-hard variants of MAXIMUM MATCHING mentioned in the classical book by Garey and Johnson [61, Problem GT55]. In this work, we will stick to the name RAINBOW MATCHING. This problem is formally defined as follows.

RAINBOW MATCHING	Parameter: k
Input: An undirected graph G , a coloring function $\chi : E(G) \rightarrow \{1, \dots, q\}$ and a positive integer k .	
Question: Does there exist a matching of size at least k such that all the edges in the matching have distinct colors?	

A matching where all the edges have distinct colors will be called *colorful matching*. Itai et al. [88] showed, already in 1978, that RAINBOW MATCHING is NP-complete on (edge-colored) bipartite graphs. Close to three decades later, Le and Pfender [105] revisited the computational complexity of this problem. Specifically, they showed that the RAINBOW MATCHING problem is NP-complete even on (edge-colored) paths, complete graphs, P_8 -free trees in which every color is used at most twice, P_5 -free linear forests in which every color is used at most twice, and P_4 -free bipartite graphs in which every color is used at most twice. In this chapter, we consider this problem from the parameterized rather than classical complexity perspective. The notation $\mathcal{O}^*(\cdot)$ is used to hide factors polynomial in the input size.

6.1 Our Contribution

Our starting point is the FPT algorithm mentioned in the article of Le and Pfender [105]. This algorithm is based on the connection between RAINBOW MATCHING and 3-SET PACKING. In the 3-SET PACKING problem, we are given a universe U , a set family

$\mathcal{F}$ consisting of subsets of U of size at most 3 and an integer k , and the objective is to check whether there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ containing at least k pairwise-disjoint sets. Observe that given an instance $I = (G, \chi, k)$ of RAINBOW MATCHING, we can view I as an instance of 3-SET PACKING by setting $U = V(G) \cup \{1, \dots, q\}$, and letting $\mathcal{F}$ contain every set $\{u, v, \chi(e)\}$ corresponding to an edge $e = uv \in E(G)$. Now, observe that (G, χ, k) is a yes-instance of RAINBOW MATCHING if and only if $(U, \mathcal{F}, k)$ is a yes-instance of 3-SET PACKING. This immediately implies that known algorithms for 3-SET PACKING can be employed to solve RAINBOW MATCHING. In particular, using the known algorithms for 3-SET PACKING, we obtain the following algorithms for RAINBOW MATCHING: (1) a deterministic algorithm running in time $\mathcal{O}^*(8.097^k)$ [158]; (2) a randomized algorithm running in time $\mathcal{O}^*(1.4953^{3k}) = \mathcal{O}^*(3.3434^k)$ [16].

Rainbow Matching on Paths. Our first contribution concerns the RAINBOW MATCHING problem on paths. Recall that the problem is NP-hard even on paths [105]. We obtain the following algorithm, which is faster than the one that we design later for general graphs.

Theorem 13. *There exists a deterministic algorithm for RAINBOW MATCHING on paths that runs in time $\mathcal{O}^*\left(\left(\frac{1+\sqrt{5}}{2}\right)^k\right)$ and uses polynomial space.*

The proof of Theorem 13 is based on a combination of the classical method of bounded search trees [29, 37, 49, 50] together with a “divide-and-conquer-like” approach. The algorithm always maintains a family of vertex-disjoint paths $\mathcal{S}$ and a path P (vertex disjoint from $\mathcal{S}$), and the objective is to find a colorful matching of size k that uses exactly one edge from each path in $\mathcal{S}$ and $k - |\mathcal{S}|$ edges from P . We call this variant of RAINBOW MATCHING the DISJOINT SET RAINBOW MATCHING problem. Observe that when $\mathcal{S} = \emptyset$, then DISJOINT SET RAINBOW MATCHING is precisely RAINBOW MATCHING. To compactly represent potential partial solutions to our problem at every step of the recursion, that is, partial witnesses that there

may indeed exist a colorful matching that uses exactly one edge from each path in $\mathcal{S}$, our algorithm works as follows. The algorithm uses an auxiliary bipartite graph where we maintain a partial solution to our problem, in terms of a matching in this bipartite graph. This has the additional benefit that the measure becomes very simple: we just measure the size left to cover, *i.e.* $k - t$, where $t = |\mathcal{S}|$ denotes the size of the partial solution. (We remark that we are able to construct a solution in the same time as it takes to solve the decision version of the DISJOINT SET RAINBOW MATCHING problem.)

Rainbow Matching on General Graphs. Our second contribution is an algorithm on general graphs that is better than the known algorithms for 3-SET PACKING. In particular, we obtain the following result.

Theorem 14. *There exists a randomized algorithm for RAINBOW MATCHING with constant, one-sided error that runs in time $\mathcal{O}^*(2^k)$ and uses polynomial space.¹*

The proof of Theorem 14 is based on the general method described in 16 for solving various packing and matching problems. We tailor the analysis of Bjorklund et al. 16 to the RAINBOW MATCHING problem. This gives us the desired saving over the algorithm for 3-SET PACKING.

Kernelization. Finally, we turn to consider the question of kernelization. Here, we exploit the connection between our problem and 3-SET PACKING to design a kernelization algorithm. We also design a smaller kernel for RAINBOW MATCHING on paths, or more generally, for graphs of bounded degree.

Theorem 15. *RAINBOW MATCHING admits a kernel of size $\mathcal{O}(k^3)$ on general graphs. Moreover, it admits a kernel of size $\mathcal{O}(d^2 k^2)$ on graphs of maximum degree d .*

In the next section we strengthen our last result. Observe that since we have

¹Specifically, if the algorithm determines that an input instance is a yes-instance, then this answer is necessarily correct.

a kernel of size $\mathcal{O}(dk^2)$ for graphs with degree upper bounded by d , it follows that there is a kernel of size $\mathcal{O}(k^2)$ for paths. We generalize the result on paths to any family of sparse graphs.

Theorem 16. *RAINBOW MATCHING on general graphs admits a kernel with $\mathcal{O}(k^2)$ vertices and $\mathcal{O}(k^3)$ edges. Moreover, the kernel can be found in time $\mathcal{O}(n^2m)$, where n and m denote the number of vertices and edges in the input graph, respectively.*

Note that the statement of Theorem 16 does not readily imply that when the input graph is a forest, we have a kernel of size $\mathcal{O}(k^2)$. This is because, a kernelization algorithm for RAINBOW MATCHING on general graphs may not return a forest as part of the output instance, even if the input instance contains a forest. Hence, the bound alone does not imply the desired bound for forests. A closer look, however, at our reduction rules will reveal that at every step we either delete an edge or a vertex. Thus, the resulting subgraph will be a forest if the input graph is a forest. In fact, this observation allows us to state as a corollary of Theorem 16: RAINBOW MATCHING admits a kernel of size $\mathcal{O}(k^2)$ for a family of graphs $\mathcal{G}$, that is both (a) *sparse* (i.e. the number of edges is at most $\mathcal{O}(n)$, where n is the number of vertices) and (b) *subgraph closed* (i.e. if G is in $\mathcal{G}$ then any subgraph of G is in $\mathcal{G}$).

We will present a separate kernelization algorithm for forests in Section 6.5.2 that is conceptually simpler than the above. We define it as a separate problem, since the input graph is a special class of graphs, and hence the output of the kernelization algorithm must also belong to the same class of graphs.

RAINBOW MATCHING ON FORESTS

Parameter: k

Input: An undirected forest G , a coloring function $\chi : E(G) \rightarrow \{1, \dots, q\}$ and a positive integer k .

Question: Does there exist a matching in G of size at least k such that all the edges in the matching have distinct colors?

The kernelization algorithm for RAINBOW MATCHING ON FORESTS uses high-degree reduction rules that follow from fairly straight-forward counting arguments. Formally stated, we prove the following result.

Theorem 17. RAINBOW MATCHING ON FORESTS *admits a kernel of size $\mathcal{O}(k^2)$. Moreover, the kernel can be found in time $\mathcal{O}(n^3)$, where n denotes the number of vertices in the input graph (forest).*

6.1.1 Overview

The proof of Theorem [16](#) works as follows. Let (G, χ, k) be an instance of RAINBOW MATCHING. Let $\mathcal{E}_i$ denote the subset of $E(G)$ that are assigned color i by χ . Let G_i denote the subgraph of G with vertex set $V(G)$ and edge set $\mathcal{E}_i$.

We start by applying a reduction rule that ensures that there are at most $2(k - 1) + 1$ distinct colors on the edges incident to any vertex in G . The rule is simply to delete an (arbitrary) edge incident on a vertex v with the $2(k - 1) + 2$ nd color. This reduces the “color-degree” of every vertex to $\mathcal{O}(k)$.

The next reduction rule removes all the edges of $\mathcal{E}_i$, if G_i satisfies some property. In particular, we check if the size of the maximum matching in G_i is at least $2k - 1$, if the answer is yes, then we delete $\mathcal{E}_i$ and reduce the parameter by 1.

Next, we greedily try to find a colorful matching of size k . If we succeed we are done, else, we know that the greedy algorithm produces a colorful matching, denoted by M , that has size at most $k - 1$. Let $\mathcal{U}$ denote the colors used by χ that do not appear on the edges of the matching M . We observe that $V(M)$ is a vertex cover (includes at least one end-point of every edge) of the graph $G^* = \cup_{i \in \mathcal{U}} G_i$. On the other hand for every G_i , $i \notin \mathcal{U}$, the size of the maximum matching in G_i is at most $2k - 2$ (because the previous rule cannot be applied). Hence, G_i has a vertex cover,

C_i , of size at most $4k - 4$. Observe that we can obtain C_i greedily. It follows that

$$C = V(M) \bigcup_{i \notin \mathcal{U}} C_i$$

is a vertex cover of G of size $\mathcal{O}(k^2)$.

Let I be $V(G) \setminus C$. Our next reduction rule finds an “irrelevant vertex” in I and deletes it. This is done by applying a well-known combinatorial result, called the p -Expansion lemma, [29, pg 32], in the following manner. We define an auxiliary bipartite graph where vertices in one side, denoted by $\mathcal{B}$, correspond to vertices in I and the vertices on the other side, denoted by $\mathcal{A}$, correspond to (vertex, color) pairs. The vertices in $\mathcal{A}$ have the property that for every edge uv in G and color $j = \chi(uv)$, $\mathcal{A}$ contains a vertex corresponding to either (v, j) or (u, j) . Additionally, our reduction rule ensures that the size of $\mathcal{A}$ is $\mathcal{O}(k^2)$.

Next, we show that if $|\mathcal{B}| > |\mathcal{A}|$, then there is an irrelevant vertex in I . Thus, when none of the reduction rules apply, we have $|\mathcal{B}| \leq |\mathcal{A}|$. Hence, we may conclude that the size of I is at most $\mathcal{O}(k^2)$. Consequently, the size of whole vertex set in the resulting instance is upper bounded by $\mathcal{O}(k^2)$.

6.1.2 Related Work

Le and Pfender [105] gave a factor $(\frac{2}{3} - \epsilon)$ approximation algorithm for RAINBOW MATCHING on general graphs for every $\epsilon > 0$. They also designed a few polynomial time algorithms when the instances of RAINBOW MATCHING are restricted to special graph classes. As stated before, Le and Pfender [105] also related this problem to 3-SET PACKING, and showed that the problem is FPT. Moreover, they showed that the problem is FPT on P_5 free forests parameterized by the number of components. Colorful matchings have also been studied from graph theoretic and combinatorial perspectives. For example, they are related to Ryser’s famous conjecture regarding

Latin transversal [142]. In the language of colorful matchings, the conjecture says that every proper edge coloring of the complete bipartite graph $K_{2n+1,2n+1}$ with $2n+1$ colors contains a rainbow matching with $2n+1$ edges. Additional examples are studies of sufficient conditions for edge-colored graphs to guarantee the existence of a colorful matching of a certain size. Moreover, previous studies also examined what is the size of the largest colorful matching in an edge-colored graph with additional restrictions. For more information on these topics, we refer to [105] and references therein.

Wang and Li [154] conjectured that for $k > 4$, every edge colored graph with minimum *color degree* (the number of edges of the distinct colors incident on a vertex in the graph) at least k contains a rainbow matching of size at least $\lceil k/2 \rceil$. LeSaulnier and Stocker et al. [108] gave sufficient conditions for a rainbow matching of size at least $\lceil k/2 \rceil$ that holds for even value of k . Kostochka and Yancey [102] proved the conjecture of Wang and Li in full. Lo [111] showed that every edge-coloured graph on $n \geq 7k/2 + 2$ vertices with color degree at least k contains a rainbow matching of size at least k . Lo also showed that the graph can be edge-decomposed into at most $\lfloor tn/2 \rfloor$ rainbow matchings, where $t \geq 11$ and the maximum color degree is upper bounded by t . Aharoni and Berger [4] studied the problem of finding the minimum number of colors which guarantees a rainbow matching in r -partite r -graphs. Alon [5] gave an exponential bound on the maximum k for which there exists a collection of k matchings, each of size t , in some r -partite r -uniform hypergraph, such that there is no rainbow matching of size t . This bound was improved by Glebov, Sudakov et al. [66], in particular for every fixed r , they gave an upper bound which is polynomial in t . Aharoni and Berger conjectured that in every proper edge coloring of a bipartite multigraph by q colors, where each color is used on at least $q+1$ edges there is a rainbow matching that uses every color. Recently, an approximate version of this conjecture has been proved [133]. An *exactly q -edge-coloring* of a graph is an q -edge-coloring of the graph that uses all q colors. Fujita and Kaneko et al. [55]

showed that an exactly q -edge-colored complete graph of order n has a rainbow matching of size $k(\geq 2)$ if $q \geq \max\{\binom{2k-3}{2} + 2, \binom{k-2}{2} + (k-2)(n-k+2) + 2\}$, where $k \geq 2$ and $n \geq 2k + 1$; and this bound on q is optimal. There are other works that have examined the size of the largest colorful matching in an edge-colored graph with additional restrictions. For more information on these topics, we refer to [105] and references therein. Rainbow matchings have been studied in properly edge colored graphs as well [34, 153].

Finally, we note that colorful matchings belong to a family of problems called rainbow subgraph problems. A *rainbow subgraph* of an edge-colored graph is a subgraph whose edges have distinct colors. We refer to [92] for a survey containing results and questions regarding rainbow subgraphs.

For a comprehensive treatment of p -Expansion Lemma, and its usefulness in designing kernelization algorithms, we refer the interested reader to the book by Fomin, Lokshtanov et al. [54].

6.1.3 Preliminaries

p -Expansion Let G be a bipartite graph with vertex bipartition (A, B) . For a positive integer p , a set of edges $M \subseteq E(G)$ is called a *p -expansion of A into B* if

- every vertex of A is incident to exactly p edges of M ;
- M saturates exactly $p|A|$ vertices in B .

The following result is a slight modification of the one presented in Lemma 2.18 in the book [29]. The proof, however, is identical to the one presented in the book.

Proposition 1 (p -Expansion Lemma, [29, Lemma 2.18]). *Let $p \geq 1$ be a positive integer and H be a bipartite graph with vertex bipartition (A, B) , on n vertices and*

m edges, such that (i) $|B| > p|A|$, and (ii) there are no isolated vertices in B . Then, there exist nonempty vertex sets $X \subseteq A$ and $Y \subseteq B$ with $|X| < |Y|$ such that

- there is a p -expansion of X into Y , and
- no vertex in Y has a neighbor outside X , i.e. $N(Y) \subseteq X$.

Furthermore, the sets X and Y can be found in time $\mathcal{O}(pnm\sqrt{pn})$.

6.2 Algorithm for RAINBOW MATCHING on Paths

In this section we give a deterministic algorithm for RAINBOW MATCHING on paths that is faster than the algorithm we will present for RAINBOW MATCHING on general graphs in the next section. Towards that we will solve the following general problem, and from there solve RAINBOW MATCHING in paths.

DISJOINT SET RAINBOW MATCHING

Input: A path P , a collection $\mathcal{S}$ of (vertex disjoint) paths (vertex disjoint from P) of arbitrary lengths, a edge coloring $\chi : E(G) \rightarrow [q]$ (where G is the disjoint union of P and the paths in $\mathcal{S}$), and a positive integer k .

Parameter: k

Task: Does there exist a colorful matching of size k that uses exactly one edge from each path in $\mathcal{S}$ and $k - |\mathcal{S}|$ edges from P ?

Note that RAINBOW MATCHING is a special case of DISJOINT SET RAINBOW MATCHING, where P is the input graph (a path), $\mathcal{S} = \emptyset$ and k is the parameter. Thus, solving DISJOINT SET RAINBOW MATCHING will yield an algorithm for RAINBOW MATCHING.

Overview. To solve DISJOINT SET RAINBOW MATCHING, whenever possible we apply reduction rules, or solve the instance in polynomial time. In the absence of either of these, the algorithm branches on an edge, based on whether it is part of the solution or not.

Measure. We associate the measure $\mu(P, \mathcal{S}, k) = k - |\mathcal{S}|$ to the instance $(P, \mathcal{S}, k)$. We will use this measure to bound the number of nodes in the search tree. When the instance is clear from the context, we will simply use μ .

Auxiliary bipartite graph. At every step of the search we maintain a bipartite graph $\mathcal{B}(\mathcal{S})$ on the vertex set $([q], \mathcal{S})$ and edge set containing pairs $cP' \in [q] \times \mathcal{S}$ such that color c appears on an edge in the path P' in (the collection) $\mathcal{S}$.

We first prove a lemma which allows us to solve the DISJOINT SET RAINBOW MATCHING problem when the measure is at most one.

Lemma 6.2.1. *If $\mu(P, \mathcal{S}, k) \leq 1$, then we can test if $(P, \mathcal{S}, k)$ is a yes-instance in polynomial time.*

Proof. We divide the proof based on whether $\mu(P, \mathcal{S}, k) < 0$ or $\mu(P, \mathcal{S}, k) = 0$ or $\mu(P, \mathcal{S}, k) = 1$. If $\mu(P, \mathcal{S}, k) < 0$, then $k < |\mathcal{S}|$ and so clearly no matching of size k can exist which chooses exactly one edge from each path in $\mathcal{S}$.

If $\mu(P, \mathcal{S}, k) = 0$, then $k = |\mathcal{S}|$. Let $Q_1, \dots, Q_k$ denote the paths in $\mathcal{S}$. We will show that there exists a colorful matching of size k that uses exactly one edge from each path in $\mathcal{S}$ if and only if there is a matching in $\mathcal{B}(\mathcal{S})$ that saturates $\mathcal{S}$. Let $\mathcal{M}$ be a colorful matching of size k that uses exactly one edge from each path in $\mathcal{S}$. Furthermore, for each i ($1 \leq i \leq k$) let $m_i \in \mathcal{M}$ be the edge that is part of path Q_i . Then $\{\chi(m_i)Q_i \mid i \in [k]\}$ forms a matching that saturates $\mathcal{S}$ in $\mathcal{B}(\mathcal{S})$. In the reverse direction given a matching $\mathcal{M}'$ in $\mathcal{B}(\mathcal{S})$ that saturates $\mathcal{S}$, we obtain a colorful matching $\mathcal{M}$ that uses exactly one edge from each path Q_i in $\mathcal{S}$, as follows. Since $\mathcal{M}'$ saturates $\mathcal{S}$ we have that for every path $Q_i \in \mathcal{S}$ there is an edge jQ_i for some $j \in [q]$.

This implies that there is an edge, say m_i on Q_i such that $\chi(m_i) = j$, i.e. m_i has color j . Since, the paths in $\mathcal{S}$ are pairwise vertex disjoint, the set $\mathcal{M}^* = \{m_i \mid i \in [k]\}$ forms a matching in the graph G . Recall that $\mathcal{M}'$ is a matching in $\mathcal{B}(\mathcal{S})$ in which one of the endpoints of the edges are from the set $[q]$, thus, it follows that $\mathcal{M}^*$ is a colorful matching of size k that uses exactly one edge from each path in $\mathcal{S}$. Thus, implying that in this case we can check whether or not $(P, \mathcal{S}, k)$ is a yes-instance in polynomial time by checking if the bipartite graph $\mathcal{B}(\mathcal{S})$ has a matching that saturates $\mathcal{S}$ [74].

If $\mu(P, \mathcal{S}, k) = 1$, then $k = |\mathcal{S}| + 1$. In this case, we consider every edge $e = uv$ on P . Let us now consider a specific iteration concerning an edge $e = uv$ on P . Then, we construct $\mathcal{B}(\mathcal{S} \cup \{uv\})$. Similar to the case of $\mu(P, \mathcal{S} \cup \{uv\}, k) = 0$, we have now reduced the problem into checking whether or not, there is a matching in $\mathcal{B}(\mathcal{S} \cup \{uv\})$ that saturates $\mathcal{S} \cup \{uv\}$. This condition can be tested in polynomial time. If in at least one iteration, we found a saturating matching then we have a yes-instance, and otherwise we have no-instance. This completes the proof. $\diamond$

Lemma 6.2.1 yields the following reduction rule.

Reduction Rule 6.2.1. *If $\mu(P, \mathcal{S}, k) \leq 1$, then using Lemma 6.2.1 test whether or not $(P, \mathcal{S}, k)$ is a yes-instance. If Lemma 6.2.1 returns yes, then return that $(P, \mathcal{S}, k)$ is a yes-instance; else, return that $(P, \mathcal{S}, k)$ is a no-instance.*

The safeness of Reduction Rule 6.2.1 follows from Lemma 6.2.1. The next reduction rule allows us to identify a prefix of the path P such that there exists a colorful matching of size at least k that contains *exactly* one edge from the prefix. If P has three vertices, then a colorful matching of size at least k can contain *exactly* one edge from P . Otherwise, we apply the following rule.

Reduction Rule 6.2.2. *In the instance $(P, \mathcal{S}, k)$, let $P = v_1, v_2, \dots, v_{n-1}, v_n$. Suppose that for every index $i \in [n - 1]$ the following property is true: when the subpaths*

$P_{i-1} = v_1, \dots, v_{i-1}$ and $P' = v_i, v_{i+1}$ of P are added to $\mathcal{S}$, the size of a maximum matching in the new bipartite graph $\mathcal{B}(\mathcal{S} \cup \{P_{i-1}, P'\})$ (obtained after the addition of P_{i-1} and P' to $\mathcal{S}$, and suitable edges) is at most $|\mathcal{S}| + 1$. Then, return that $(P, \mathcal{S}, k)$ is a no-instance.

Next we show the correctness of Reduction Rule [6.2.2](#).

Lemma 6.2.2. *Reduction Rule [6.2.2](#) is safe.*

Proof. For the sake of contradiction, we assume that $(P, \mathcal{S}, k)$ is a yes-instance. In this case we will show that there exists an index $i \in [n - 1]$ with the following property: when the subpaths $P_{i-1} = v_1, \dots, v_{i-1}$ and $P' = v_i, v_{i+1}$ of P are added to $\mathcal{S}$, the size of a maximum matching in the new bipartite graph $\mathcal{B}(\mathcal{S} \cup \{P_{i-1}, P'\})$ is $|\mathcal{S}| + 2$. This will be contradiction, and thereby prove the lemma.

Since $(P, \mathcal{S}, k)$ is a yes-instance there exists a colorful matching of size k , denoted by $\mathcal{M}$, that uses k colors from $[q]$, exactly one edge from each path in $\mathcal{S}$, and $k - |\mathcal{S}|$ edges from P . Thus, there is a maximum matching in the bipartite graph on $\mathcal{B}(\mathcal{S})$ that saturates every vertex in the $\mathcal{S}$ -side. It is obtained by taking edges that connect a vertex in the $\mathcal{S}$ -side (*i.e.* a path in the collection $\mathcal{S}$) with the color that appears on the matching edge in $\mathcal{M}$ that is part of the same path. We use $\mathcal{M}'$ to denote this bipartite matching.

Let $\{e_{j_1}, e_{j_2}, \dots, e_{j_{k-|\mathcal{S}|}}\}$ denote the matching edges in $\mathcal{M}$ as they appear left to right in P . For some $i \geq 3$, let $v_i v_{i+1}$ denote the edge e_{j_2} (the matching edge with the second smallest index in P). It follows that edge e_{j_1} appears in the subpath $P_{i-1} = v_1, \dots, v_{i-1}$. Note that the vertex $\chi(e_{j_2}) \in [q]$ is not saturated by $\mathcal{M}'$ in $\mathcal{B}$ because e_{j_2} is part of the matching $\mathcal{M}$ while it is inside P . Similarly, the vertex $\chi(e_{j_1}) \in [q]$ is also not saturated by $\mathcal{M}'$ in $\mathcal{B}$. Hence, when paths P_{i-1} and P' are added to the bipartite graph, $\mathcal{M}'$ can be extended by exactly two more edges: edges between $\chi(e_{j_1})$ and P_{i-1} and $\chi(e_{j_2})$ and P' . Thus, the new bipartite graph has a

matching of size $|\mathcal{S}| + 2$. $\diamond$

The safeness of Reduction Rule [6.2.2](#), leads us to the following conclusion.

Lemma 6.2.3. *If $(P, \mathcal{S}, k)$ is a yes-instance on which Reduction Rules [6.2.1](#) and [6.2.2](#) are not applicable, then there exists an index $i \in [n - 1]$ such that there exists a colorful matching of size k that uses exactly one edge from the subpath $P_i = v_1, \dots, v_i$ of P . Furthermore, such an index i can be found in polynomial time.*

Proof. Since Reduction Rules [6.2.1](#) and [6.2.2](#) are not applicable we have that $\mu(P, \mathcal{S}, k) \geq 2$. This implies that $k \geq |\mathcal{S}| + 2$. Let i denote the *smallest* integer in $[n - 1]$ for which the following holds:

Property $(\star\star)$ when the subpaths $P_{i-1} = v_1, \dots, v_{i-1}$ and $P' = v_i, v_{i+1}$ of P are added to $\mathcal{M}$, the size of a maximum matching in the new bipartite graph $\mathcal{B}(\mathcal{S} \cup \{P_{i-1}, P'\})$ (obtained after the addition of P_{i-1} and P' to $\mathcal{S}$, and suitable edges) is $|\mathcal{S}| + 2$.

Observe that since $k \geq |\mathcal{S}| + 2 \geq 2$, such an integer i must exist.

Note that any colorful matching of size k uses at most *one* edge from P_i , else, it contradicts the fact that i is the smallest integer that satisfies Property $(\star\star)$. Also note that since we have a colorful matching of size at least two in P_{i+1} , so $i \geq 3$. Note that there always exists a colorful matching of size k that uses one of the first two edges of the path P . If none of the first two edges of P is contained in a colorful matching of size k , then, since P is a path, at least one of these two edges is vertex disjoint from the matching. Let this edge be denoted by e . We produce a new colorful matching of size at least k by adding e to the matching and removing the edge of color $\chi(e)$ (if any present) from the matching. Hence, we can conclude that there always exists a colorful matching of size k that must use one of the edges on P_i . Clearly, we can find the smallest integer described in the statement of the lemma in polynomial time. This concludes the proof. $\diamond$

Lemma 6.2.3 yields a branching rule that can be described as follows. Let P_i be the subpath of P (given by Lemma 6.2.3) such that there exists a colorful matching of size k that uses exactly one edge from it. We recursively solve two subproblems one where we assume that edge $v_i v_{i+1}$ is in the colorful matching of size k we are constructing, and the other where we assume that edge $v_i v_{i+1}$ is not part of the solution we are constructing. Note that this rule is exhaustive because an edge (in particular, $v_i v_{i+1}$) can either belong to a matching, or it does not.

Algorithm. Now we can describe the branching rule in detail along with the recursive call to a subproblem. Let $\mathcal{I} = (P = v_1, \dots, v_n, \mathcal{S}, k)$ be the instance of DISJOINT SET RAINBOW MATCHING, where none of the Reduction Rules 6.2.1 or 6.2.2 are applicable. Let P_i be the subpath of P as described in Lemma 6.2.3.

Branch 1: (The edge $v_i v_{i+1}$ belongs to a colorful matching of size k .)

We recursively solve the problem on the instance $(P \setminus \{v_1, \dots, v_{i+1}\}, \mathcal{S} \cup \{P_{i-1}\} \cup \{[v_i v_{i+1}]\}, k)$. Since the size of $\mathcal{S}$ increases by 2, the measure μ decreases by 2. Observe that by Lemma 6.2.3 we know that there exists a colorful matching of size k that uses exactly one edge from the subpath P_i . However, since we have assumed that the edge $v_i v_{i+1}$ belongs to the colorful matching of size k , thus, edge $v_{i-1} v_i$ cannot be part of the same matching, and so one of the edges in P_{i-1} must be part of the matching as well. Thus, in this case we know that two of the edges in our matching are due to the edge $v_i v_{i+1}$ and the other is from P_{i-1} .

Branch 2: (The edge $v_i v_{i+1}$ does not belong to a colorful matching of size k .)

In this case we recursively solve the problem on $(P \setminus \{v_1, \dots, v_i\}, \mathcal{S} \cup \{P_i\}, k)$. Since the size of $\mathcal{S}$ increases by 1 and k remains the same, the measure μ decreases by 1. The correctness of this step follows from Lemma 6.2.3.

If either of the branches returns “yes”, we return the same. Else, we return that the given instance is a no-instance.

The resulting branching vector for this algorithm is $(2, 1)$. Thus, solving the polynomial $x^2 \geq x + 1$ for a positive root yields $x \geq \frac{1}{2}(1 + \sqrt{5}) = 1.6181$. This upper bounds the running time of our algorithm. The correctness of the algorithm follows from Lemmas [6.2.1](#), [6.2.2](#) and [6.2.3](#).

Recall that as explained at the very onset of our discussion: Since RAINBOW MATCHING is a special case of DISJOINT SET RAINBOW MATCHING, hence our algorithm can solve RAINBOW MATCHING by using the algorithm for the latter on the instance $(G, \mathcal{S} = \emptyset, k)$. This completes the proof of Theorem [13](#).

6.3 FPT Algorithm for RAINBOW MATCHING on General Graphs

This section is inspired by the proof of Theorem 4 in [\[16\]](#), which solves 3-SET PACKING in time $\mathcal{O}^*(3.3434^k)$. We show that by an analysis tailored to RAINBOW MATCHING, we improve upon the time complexity $\mathcal{O}^*(3.3434^k)$. More precisely, the objective of this section is to prove Theorem [14](#).

Towards the proof of Theorem [14](#), we need to consider a problem called 3-SET PREPACKING, which was introduced in [\[16\]](#). The input of this problem consists of an n -element universe U , an n_1 -element subuniverse $U_1 \subseteq U$, a family $\mathcal{F}$ of 3-sets, a positive integer k , and three non-negative integers p_0 , p_1 and p_2 whose sum is k . The task is to determine whether there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size k such that the 3-sets in $\mathcal{F}'$ are pairwise-disjoint, and for all $i \in \{0, 1, 2\}$, there exist exactly p_i sets S in $\mathcal{F}'$ such that $|S \cap U_1| = i$. We would need to rely on the following result.

Proposition 6.3.1. [\[16\]](#) *There exists a randomized algorithm for 3-SET PREPACK-*

ING with constant, one-sided error that runs in time $\mathcal{O}^*(2^{3p_0+2p_1+p_2})$ and uses polynomial space. Specifically, if the algorithm determines that an input instance is a yes-instance, then this answer is necessarily correct.

Let us denote the algorithm given by Proposition [6.3.1](#) by **PrepackAlg**. We present a reduction from our problem to 3-SET PREPACKING. For this purpose, we describe a procedure **Reduce** that given an instance $(G, \chi : E(G) \rightarrow [q], k)$ of RAINBOW MATCHING, constructs an instance $\text{reduce}(G, \chi, k) = (U, U_1, \mathcal{F}, k, p_0, p_1, p_2)$ of 3-SET PREPACKING with the same parameter k . Let us denote $n_1 = |V(G)| = n - q$, where n would denote $|U|$. First, **Reduce** sets $U = V(G) \cup [q]$, $\mathcal{F} = \{\{u, v, \chi(uv)\} : uv \in E(G)\}$, $p_0 = 0$, $p_1 = 0$ and $p_2 = k$. Second, **Reduce** sets $U_1 = V(G)$. Let us now argue that we obtain an equivalent instance.

Lemma 6.3.1. *Let $(G, \chi : E(G) \rightarrow [q], k)$ be an instance of RAINBOW MATCHING. Then, **Reduce** $(G, \chi, k) = (U, U_1, \mathcal{F}, k, p_0, p_1, p_2)$ is a yes-instance of 3-SET PREPACKING if and only if $(G, \chi : E(G) \rightarrow [q], k)$ is a yes-instance of RAINBOW MATCHING.*

Proof. In the first direction suppose that **Reduce** $(G, \chi, k) = (U, U_1, \mathcal{F}, k, p_0, p_1, p_2)$ is a yes-instance of 3-SET PREPACKING. In particular, we then have that there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size k such that the 3-sets in $\mathcal{F}'$ are pairwise-disjoint. Let us denote $\mathcal{M} = \{uv \in E(G) : \exists S \in \mathcal{F}' \text{ s.t. } \{u, v\} \subseteq S\}$. Since $|\mathcal{F}'| = k$, we have that $|\mathcal{M}| = k$, and since the 3-sets in $\mathcal{F}'$ are pairwise-disjoint, we have that $\mathcal{M}$ is a colorful matching. Thus, $(G, \chi : E(G) \rightarrow [q], k)$ is a yes-instance of RAINBOW MATCHING.

In the other direction suppose $(G, \chi : E(G) \rightarrow [q], k)$ is a yes-instance of RAINBOW MATCHING. Then there exists a colorful matching $\mathcal{M}$ of size k . Let us denote $\mathcal{F}' = \{\{u, v, \chi(uv)\} : uv \in \mathcal{M}\}$. Since the size of $\mathcal{M}$ is k , we have that the size of $\mathcal{F}'$ is k and since $\mathcal{M}$ is a colorful matching we have that the sets in $\mathcal{F}'$ are pairwise

disjoint. Notice that every set in $\mathcal{F}'$ has exactly two elements from U_1 . Therefore, for all $i \in \{0, 1, 2\}$, there exist exactly p_i sets S in $\mathcal{F}'$ such that $|S \cap U_1| = i$. Thus, $(U, U_1, \mathcal{F}, k, p_0, p_1, p_2)$ is a yes-instance of 3-SET PREPACKING. $\diamond$

Let us now prove Theorem 14.

Proof of Theorem 14. Given an instance $(G, \chi : E(G) \rightarrow [q], k)$ of RAINBOW MATCHING, we construct the instance **Reduce** $(G, \chi, k) = (U, U_1, \mathcal{F}, k, p_0, p_1, p_2)$ of 3-SET PREPACKING. Then, we run the algorithm given by Proposition 6.3.1. We accept if and only if the algorithm from Proposition 6.3.1 accepted. The correctness follows from Proposition 6.3.1 and Lemma 6.3.1. Since, $p_0 = p_1 = 0$ and $p_2 = k$, by Proposition 6.3.1, the total running time is $\mathcal{O}^*(2^k)$. $\diamond$

6.4 Kernelization Algorithms

In this section we give a proof for Theorem 15. We first describe a kernel on general graphs. The kernelization algorithm on general graphs is actually a known kernel for 3-SET PACKING given in [29, Theorem 12.20] (also see [3, 33]). The best known kernel for 3-SET PACKING is given by Abu-Khzam [3] and it has $\mathcal{O}(k^2)$ elements and $\mathcal{O}(k^3)$ sets. However, as we explain now, we cannot use the kernel given by Abu-Khzam [3] directly for our purposes. This is in contrast to the fact that one can use the best known parameterized algorithms for 3-SET PACKING to design parameterized algorithms for RAINBOW MATCHING. Given an instance (G, χ, k) of RAINBOW MATCHING, we can transform it to an instance $(U, \mathcal{F}, k)$ of 3-SET PACKING as explained in the introduction. Recall that a kernelization algorithm always returns an equivalent instance of the same problem. So, if we apply a kernelization algorithm for 3-SET PACKING, then it will return an equivalent instance $(U', \mathcal{F}', k')$ of 3-SET PACKING and *not of* RAINBOW MATCHING. A priori, it

is not clear how we can transform $(U', \mathcal{F}', k')$ to an instance of RAINBOW MATCHING without increasing the size bounds on the kernel we obtain for RAINBOW MATCHING. Thus, to design a kernelization algorithm for RAINBOW MATCHING we give a kernelization algorithm for 3-SET PACKING such that it is easy to transform an instance of the latter to an instance of RAINBOW MATCHING. This kernel is given here mainly for *completeness*.

6.4.1 Kernelization on general graphs: Algorithm I

Now we give the kernelization algorithm alluded to in the first part of Theorem [15](#). Towards that we will use the sunflower lemma – a classical result of Erdős and Rado. We first define the terminology used in the statement of the lemma. A *sunflower* with k petals and a core Y is a collection of sets $S_1, \dots, S_k$ such that $S_i \cap S_j = Y$ for all $i \neq j$; the sets $S_i \setminus Y$ are petals and we require *none of them to be empty*. Note that a family of pairwise disjoint sets is a sunflower (with an empty core).

Proposition 6.4.1. [\[29, pg 38\]](#) (Sunflower lemma) *Let $\mathcal{A}$ be a family of sets (without duplicates) over a universe U , such that each set in $\mathcal{A}$ has cardinality exactly d . If $|\mathcal{A}| > d!(k-1)^d$, then $\mathcal{A}$ contains a sunflower with k petals and such a sunflower can be computed in time polynomial in $|\mathcal{A}|$, $|U|$, and k .*

Proof of first part of Theorem [15](#). Given an instance (G, χ, k) of RAINBOW MATCHING, we view this as an instance $\mathcal{J}$ of 3-SET PACKING as follows: $U = V(G) \cup [q]$ and $\mathcal{F}$ consists of a set $\{u, v, \chi(uv)\}$ corresponding to every edge $e = uv \in E(G)$.

Reduction Rule 6.4.1. *Let $(U, \mathcal{F}, k)$ be an instance of 3-SET PACKING and suppose that $\mathcal{F}$ contains a sunflower $\hat{S} = \{S_1, \dots, S_{3(k-1)+2}\}$ of cardinality $3k-1$ with core Y . Then, return $(U', \mathcal{F}', k)$, where $U' = \bigcup_{X \in \mathcal{F}'} X$, and $\mathcal{F}' = (\mathcal{F} \setminus \{S_1\})$ is obtained by deleting the set S_1 from $\mathcal{F}$.*

To show the correctness of Reduction Rule [6.4.1](#), we need to show the following lemma.

Lemma 6.4.1. *Reduction Rule [6.4.1](#) is safe.*

Proof. We will prove that $(U, \mathcal{F}, k)$ is a yes-instance of 3-SET PACKING if and only if $(U', \mathcal{F}', k)$ is a yes-instance of 3-SET PACKING. It is clear that if $(U', \mathcal{F}', k)$ is a yes-instance of 3-SET PACKING then so is $(U, \mathcal{F}, k)$; so the backward direction holds straightaway.

For the forward direction, we assume that we have a solution $\mathcal{S}$ to $(U, \mathcal{F}, k)$, *i.e.*, a set of k pairwise disjoint sets. If $\mathcal{S}$ does not contain S_1 , then it is also a solution for $(U', \mathcal{F}', k)$. So let us assume that $S_1 \in \mathcal{S}$. Observe that the number of elements appearing in the sets in $\mathcal{S}$, apart from those present in S_1 , is $3(k-1)$. Also, note that no set in $\mathcal{S} \setminus \{S_1\}$ intersects the core Y since $Y \subset S_1$ and the sets in $\mathcal{S}$ are pairwise disjoint. Thus, the number of sets in the sunflower $\hat{S}$ that intersects a set of $\mathcal{S}$ is upper bounded by $1 + 3(k-1)$ (the first one for S_1). This implies there exists a set $S^* \in \hat{S}$ that is pairwise disjoint with every set in $\mathcal{S} \setminus \{S_1\}$. Thus, $(\mathcal{S} \setminus \{S_1\}) \cup \{S^*\}$ is a solution of size k for $(U', \mathcal{F}', k)$. This completes the proof. $\diamond$

Now, we are ready to describe the kernelization algorithm. If the number of sets in $\mathcal{F}$ is more than $6(3k-2)^3$, then the kernelization algorithm applies the sunflower lemma to find a sunflower of size $3k-1$, and applies Reduction Rule [6.4.1](#) on this sunflower.

The algorithm applies this procedure exhaustively, and obtains a new family of sets $\mathcal{F}'$ of size at most $6(3k-2)^3$. This concludes the size bound on the family $(U', \mathcal{F}', k)$ of 3-SET PACKING. Observe that throughout the process, we have never reduced the size of any set in the family and each set in the family still corresponds to an edge and its color. Thus, given $(U', \mathcal{F}', k)$, let W be the vertices present in U' . Then we return $(G[W], \chi', k)$, where edge coloring χ' is the restriction of χ to the

edges present in $G[W]$. This concludes the description of the kernelization algorithm.

◇

6.4.2 Kernelization on graphs of bounded degree

In this section we design a small kernel for RAINBOW MATCHING on graphs of bounded degree. Let (G, χ, k) be an instance of RAINBOW MATCHING. Throughout this section we assume that the *maximum degree* of G is upper bounded by a fixed constant d .

For $i \in [q]$, let $E_i = \{e \in E(G) \mid \chi(e) = i\}$. We call the set of edges E_i as a *color class with color i* . We first prove the following useful observation.

Observation 6.4.1. *Let $\mathcal{M}$ be a matching of size t of G . Then $V(\mathcal{M})$ contains endpoints of at most $2t(d - 1)$ edges from $E(G) \setminus \mathcal{M}$.*

Proof. Observe that $\mathcal{M}$ has $2t$ distinct vertices and each vertex has degree at most d . Therefore, each vertex in $V(\mathcal{M})$ has at most $d - 1$ neighbors outside $\mathcal{M}$. So, there are at most $2t(d - 1)$ neighbors of vertices of $V(\mathcal{M})$ outside $\mathcal{M}$. That is, there are at most $2t(d - 1)$ edges incident to the vertices of $V(\mathcal{M})$ that are from $E(G) \setminus \mathcal{M}$. ◇

Next we give reduction rule that bounds the size of each color class.

Reduction Rule 6.4.2. *If there exists $i \in [q]$ such that $|E_i| \geq 2(d - 1)(k - 1) + 1$ then delete E_i and reduce k by 1. That is, we obtain an instance $(G', \chi', k - 1)$. Here, G' is obtained by deleting all the edges in E_i and edge coloring χ' is obtained by restricting χ to the edges in G' .*

Lemma 6.4.2. *Reduction Rule 6.4.2 is safe.*

Proof. We will prove that G has a colorful matching of size k if and only if G' has a colorful matching of size $k - 1$. We first prove the forward direction. If G has a

colorful matching $\mathcal{M}$ of size k that contains an edge $uv \in E_i$, then $\mathcal{M} \setminus \{uv\}$ is a colorful matching of size $k - 1$ in the graph G' . If $\mathcal{M}$ does not have an edge of color i , then $\mathcal{M}$ itself is a colorful matching of size at least $k - 1$ for G' .

For the backward direction, let $\mathcal{M}'$ be a colorful matching of size $k - 1$ of G' . Due to Observation [6.4.1](#), at most $2(d - 1)(k - 1)$ edges from E_i can share vertices with $\mathcal{M}'$. Since, $|E_i| \geq 2(d - 1)(k - 1) + 1$, E_i has at least one edge that does not share a vertex with $\mathcal{M}'$. Let that edge be uv . Then, it follows that $\mathcal{M}' \cup \{uv\}$ is a k size colorful matching of G , and our proof is complete. $\diamond$

We apply Reduction Rule [6.4.2](#) exhaustively. If the premise of the rule is not satisfied, then for each color i , we have that $|E_i| \leq 2(d - 1)(k - 1)$. Next we give a polynomial time procedure that either outputs a colorful matching of size at least k or bounds the number of colors.

Lemma 6.4.3. *Let (G, χ, k) be an instance of RAINBOW MATCHING for which Reduction Rule [6.4.2](#) is not applicable. Then, in polynomial time either we can conclude that (G, χ, k) is a yes-instance or the number of distinct colors in the instance is upper bounded by $2(d - 1)(k - 1)$.*

Proof. We iteratively try to build a colorful matching of size k . If we fail to do so, then it will enable us to bound the number of color classes. Let $\mathcal{M}$ be an empty set. We repeat the below procedure until the graph is empty.

1. Pick an edge uv of G arbitrarily, and add it to $\mathcal{M}$. Let the edges incident on u have colors $c_u^1, c_u^2, \dots, c_u^\ell$ and the edges incident on v have colors $c_v^1, c_v^2, \dots, c_v^p$.
2. Delete all the edges in $\bigcup_{i=1}^\ell E_{c_u^i}$ and $\bigcup_{i=1}^p E_{c_v^i}$. Let the resulting graph be also called G .

If we can continue the above process for k steps (*i.e.* $|\mathcal{M}| \geq k$) then $\mathcal{M}$ is a colorful matching of size at least k . In this case we output $\mathcal{M}$ as the desired colorful

matching. To see its correctness, observe that in every iteration we deleted the edges incident on both the endpoints of the added to $\mathcal{M}$. So, the edges we added to $\mathcal{M}$ are indeed pairwise vertex disjoint. Also, note that we delete all the edges with colors that are used on the edges that are incident to the edges that were added to $\mathcal{M}$. Hence, the edges in $\mathcal{M}$ have distinct color.

Otherwise, our procedure ends within at most $k - 1$ steps, and so $|\mathcal{M}| \leq k - 1$. Due to Observation [6.4.1](#), the number of edges incident to a vertex in $V(\mathcal{M})$ is at most $2(d - 1)(k - 1)$. Therefore, the number of color classes that contain a vertex from $V(\mathcal{M})$ is at most $2(d - 1)(k - 1)$. Following this we are left with an empty graph and every edge we deleted or picked in $\mathcal{M}$ is incident to $V(\mathcal{M})$. Thus, we have shown that in this case, we can have at most $2(d - 1)(k - 1)$ color classes. $\diamond$

Lemmas [6.4.2](#) and [6.4.3](#) together prove second part of Theorem [15](#).

6.5 Improved Kernelization Algorithms

In this section we design a polynomial kernel for RAINBOW MATCHING on general graphs. In particular, we give a proof of Theorem [16](#).

The section is organized as follows. We first give a few reduction rules and then use them to give a graph decomposition. Next, we use the graph decomposition along with some elementary reduction rules to design an $\mathcal{O}(k^2)$ kernel for RAINBOW MATCHING ON FORESTS. Finally, by applying a reduction rule based on the p -Expansion Lemma, we give an $\mathcal{O}(k^2)$ kernel for RAINBOW MATCHING on general graphs.

Let (G, χ, k) denote an input to RAINBOW MATCHING. Throughout this section, n and m denote the number of vertices and edges in G , respectively. For a subgraph G' of G , we use $\chi|_{G'}$ to denote χ when restricted to the edges of G' . Furthermore,

we will use E_i to denote the set of edges in $E(G)$ that have been assigned color i by χ . The graph induced by E_i will be denoted by $G[E_i]$. We will assume that χ assigns an integer in $[q]$.

6.5.1 Decomposing the graph into a vertex cover and an independent set

We start with the following simple reduction rule.

Reduction Rule 6.5.1. *If there is an isolated vertex v in G , then delete v . The resulting instance is $(G' = G - \{v\}, \chi|_{G'}, k)$.*

This reduction rule is safe because an isolated vertex cannot be part of any matching. Clearly, Reduction Rule 6.5.1 can be executed in time $\mathcal{O}(n^2)$ using adjacency matrix representation of G . In order to decompose our graph into a vertex cover and an independent set we need a bound on the vertex cover of the graph. Towards this we give the following reduction rule.

Reduction Rule 6.5.2. *Suppose that there exists $i \in [q]$ such that the size of a maximum matching in $G[E_i]$ is at least $2k - 1$. Then, delete all the edges in E_i and reduce k by 1. The resulting instance is denoted by $(G', \chi|_{G'}, k - 1)$.*

Lemma 6.5.1. *The Reduction Rule 6.5.2 is safe.*

Proof. We will prove that G has a colorful matching of size k if and only if G' has a colorful matching of size $k - 1$. We first prove the forward direction. If G has a colorful matching M of size k that contains an edge uv of color i then $M \setminus \{uv\}$ is a colorful matching of size $k - 1$ in the graph G' . If M does not have an edge of color i , then M itself is a colorful matching of size at least $k - 1$ for G' .

For the reverse direction, let M' be a colorful matching of size $k - 1$ of G' . Recall that the size of maximum matching of $G[E_i]$ is at least $2k - 1$, i.e, there are $2k - 1$

pairwise vertex disjoint edges of color i . Since M' has $2(k-1)$ distinct vertices, there is at least one edge in E_i that is not incident to any vertex in $V(M')$. Let uv denote that edge. Then, since the edges of color i are not present in G' , we can infer that $M' \cup \{uv\}$ is a colorful matching of size k in G . This completes the proof. $\diamond$

We can construct the graph $G[E_i]$, for each $i \in [q]$ in time $\mathcal{O}(m)$ and find maximum matching in $G[E_i]$ in time $\mathcal{O}(m\sqrt{n})$. The edge set of G can be reduced in time $\mathcal{O}(n^2)$ using the adjacency matrix representation of G . Hence, Reduction Rule [6.5.2](#) can be executed in time $\mathcal{O}(qm\sqrt{n} + n^2)$.

When Reduction Rule [6.5.2](#) cannot be applied to the current instance, then we know that for each color $i \in [q]$, the size of the maximum matching in $G[E_i]$ is at most $2(k-1)$. Next, we describe a polynomial time procedure that either outputs a colorful matching of size at least k or gives an upper bound on the size of a vertex cover of a graph on which Reduction Rule [6.5.2](#) is not applicable.

Lemma 6.5.2. *Let (G, χ, k) be an instance of RAINBOW MATCHING on which Reduction Rule [6.5.2](#) is not applicable. Let n and m denote the number of vertices and edges in G , respectively. Then, in time $\mathcal{O}(n^2m)$ either we can conclude that (G, χ, k) is a YES-instance or that G has a vertex cover of size at most $4(k-1)^2 + 2(k-1)$. Furthermore, in the later case, we can output a vertex cover $\mathcal{X}$ of G of size $4(k-1)^2 + 2(k-1)$ in time $\mathcal{O}(n^2m)$.*

Proof. We iteratively try to build a colorful matching of size k . If we fail to do so, then we can exhibit a vertex cover in G whose size is upper bounded by $4(k-1)^2 + 2(k-1)$.

We begin by initializing $\mathcal{M}$ to be an empty set. We repeat the following procedure until the graph has no edge, i.e it is an independent set.

- (1) Pick an edge uv of G arbitrarily and add it to $\mathcal{M}$.

- (2) Delete the edges incident on the endpoints u and v , and the edges in the color class $E_{\chi(uv)}$. Let the resulting graph be denoted by G .

If we can continue the above process for k steps (*i.e.* $|\mathcal{M}| \geq k$), then $\mathcal{M}$ is a colorful matching of size at least k . In this case we output $\mathcal{M}$ as the desired colorful matching and exit.

Before we prove the correctness of the procedure, we note that it takes time $\mathcal{O}(n^2m)$. This is because step (2) of the procedure takes $\mathcal{O}(n^2)$ and is executed at most m times.

To see the proof of correctness of the procedure, we observe that in every iteration we deleted the edges incident on the endpoints of the edge (most recently) added to $\mathcal{M}$. Hence, the edges we pick in $\mathcal{M}$ are indeed pairwise vertex disjoint. Also, note that we deleted all the edges that have the same color as uv , thereby ensuring that no two edges of the same color class are picked twice. Thus, $\mathcal{M}$ is a colorful matching of size at least k . Let $V(\mathcal{M})$ denote the set of vertices obtained by taking the end-points of the edges in $\mathcal{M}$.

If the process cannot continue beyond the first $k - 1$ steps, then $|\mathcal{M}| \leq k - 1$. Without loss of generality, we may assume that for some $\ell \leq k - 1$, the set of colors on the edges in $\mathcal{M}$ is denoted by $[\ell]$. Note that during the procedure, for any $i \in [q] \setminus [\ell]$, only the edges in E_i that are incident on a vertex in $V(\mathcal{M})$ have been deleted, and yet at the end we have an independent set. Thus, it follows that every edge in the induced subgraph $G[\cup_{j \in [q] \setminus [\ell]} E_j]$ shares a vertex with an edge in $\mathcal{M}$. In other words, the set $V(\mathcal{M})$ is a vertex cover of $G[\cup_{j \in [q] \setminus [\ell]} E_j]$. Recall that for each $i \in [q]$, the size of a maximum matching in $G[E_i]$ is at most $2(k - 1)$ and thus, has a vertex cover of size at most $4(k - 1)$ – just take the end-points of any maximal matching. In particular, for each $i \in [\ell]$, the induced subgraph $G[E_i]$ has a vertex cover of size at most $4(k - 1)$. Therefore, G must have a vertex cover of size at most

$$\sum_{i \in [\ell]} 4(k-1) + |V(\mathcal{M})| \leq 4(k-1)(k-1) + 2(k-1).$$

It is easy to see that we can get a vertex cover of G , denoted by $\mathcal{X}$, of size $4(k-1)^2 + 2(k-1)$ in polynomial time: the vertices in $V(\mathcal{M})$ and for each $i \in [\ell]$ a vertex cover for $G[E_i]$, obtained by taking the end-points of a maximal matching in $G[E_i]$. This concludes the proof. $\diamond$

Decomposition of the graph. Let (G, χ, k) be an instance on which Reduction Rule [6.5.2](#) is not applicable and Lemma [6.5.2](#) does not return that it is a YES-instance. Then, we show that an application of Lemma [6.5.2](#) yields a vertex cover $\mathcal{X}$. The following lemma follows from the details of the proof of Lemma [6.5.2](#).

Lemma 6.5.3. *Let (G, χ, k) be an instance of RAINBOW MATCHING. Let n and m denote the number of vertices and edges in G , respectively. Then, in time $\mathcal{O}(n^2m)$ either we can conclude that (G, χ, k) is a YES-instance or output a vertex cover $\mathcal{X}$ of G such that the following conditions hold.*

- $\mathcal{X} = \cup_{i \in [\ell]} X_i \cup V(\mathcal{M})$ where $\mathcal{M}$ is a colorful matching containing edges of color $\{1, \dots, \ell\}$ for some $\ell \leq k-1$.
- Each X_i is a vertex cover of the induced subgraph $G_i = G[E_i]$ such that $|X_i| \leq 4(k-1)$.
- Furthermore, $V(\mathcal{M})$ is the vertex cover of the induced subgraph $G[\cup_{i=\ell+1}^q E_i]$.

We will denote the complement of $\mathcal{X}$, the set $V(G) \setminus \mathcal{X}$, an independent set by $\mathcal{I}$.

6.5.2 Kernelization on Forests

In this section, we will design a kernel for RAINBOW MATCHING ON FORESTS of size $\mathcal{O}(k^2)$. This kernel is based on Lemma [6.5.3](#) and some elementary reduction rules. Recall that for general graphs, Theorem [15](#) gives a bound of $\mathcal{O}(k^3)$. Let (G, χ, k) be

an instance of RAINBOW MATCHING ON FORESTS on which Reduction Rules [6.5.1](#) and [6.5.2](#) do not apply. Furthermore, assume that an application of Lemma [6.5.3](#) gives us the decomposition of $V(G)$ into $\mathcal{X}$ and $\mathcal{I}$ where the size of $\mathcal{X}$ is $\mathcal{O}(k^2)$. Thus, to get the desired kernel all we need is to reduce the size of $\mathcal{I}$ to $\mathcal{O}(k^2)$.

We start with the following observation which is a direct consequence of the decomposition of G , a forest.

Lemma 6.5.4. *Let $V_{\geq 2} = \{v \in \mathcal{I} \mid \deg(v) \geq 2\}$. Then, $|V_{\geq 2}| \leq |\mathcal{X}| \leq 4k(k-1) + 2(k-1) - 1$.*

Proof. Towards the proof consider the induced subgraph $G[V_{\geq 2} \cup \mathcal{X}]$. Let $v \in V_{\geq 2}$ such that $\{x, y\} \subseteq \mathcal{X}$ are two of its neighbors. We put an edge between x and y , and delete v . This operation can also be viewed as contracting the vertex v to one of its neighbors. Thus, what we get is a minor of a forest and hence a forest. This gives a forest on $|\mathcal{X}|$ vertices, and a subset of edges that correspond to vertices in $V_{\geq 2}$. Since this forest has at most $|\mathcal{X}| - 1$ edges, we can conclude that $|V_{\geq 2}| \leq |\mathcal{X}|$. From Lemma [6.5.2](#) we know that $|\mathcal{X}| \leq 4k(k-1) + 2(k-1) - 1$, and so the proof is complete. $\diamond$

To bound the size of the independent set $\mathcal{I}$, we need to bound $V_{=1} = \mathcal{I} \setminus V_{\geq 2} = \{v \in \mathcal{I} \mid \deg(v) = 1\}$. In the upcoming discussion, we will solely focus on $V_{=1}$. The following reduction rule allows us to reduce the multiplicity of colors on vertices in $\mathcal{X}$.

Reduction Rule 6.5.1. *Let $u \in \mathcal{X}$. Suppose that for a color $c \in [q]$, there are at least two edges of color c in $E(\{u\}, V_{=1})$. Then, delete an edge of color c from $E(\{u\}, V_{=1})$. The resulting instance is denoted by $(G', \chi|_{G'}, k)$*

Lemma 6.5.5. *Reduction Rule [6.5.1](#) is safe.*

Proof. Let (G, χ, k) denote the instance before application of the reduction rule. We

will prove that G has a colorful matching of size k if and only if G' has a colorful matching of size k .

Let M be a solution for the instance (G, χ, k) . Let uv be the deleted edge. If $uv \in E(G) \setminus M$, then M is a solution for $(G', \chi|_{G'}, k)$. Suppose that $uv \in M$. Note that, there is at least 1 edge of color c in $E(\{u\}, V_{=1} \setminus \{v\})$. Let this edge be denoted by uv' . We claim that $M' = M \setminus \{uv\} \cup \{uv'\}$ is a colorful matching of size k . Notice that M' is matching because $M \setminus \{uv\}$ is a matching, and neither u nor v' is incident to any edge in $M \setminus \{uv\}$. So the edges of M' are pairwise vertex disjoint. It is a colorful matching because $M \setminus \{uv\}$ is colorful matching and $c = \chi(uv) = \chi(uv')$. Moreover, $|M| = |M'|$. Therefore, M' is a solution for $(G', \chi|_{G'}, k)$.

For the reverse direction, we note that G' is an induced subgraph of G , and so any solution in $(G', \chi|_{G'}, k)$ is also a solution of (G, χ, k) . $\diamond$

The next reduction rule enables us to reduce the number of colors (on edges) incident to vertices in $\mathcal{X}$.

Reduction Rule 6.5.2. *Let $u \in \mathcal{X}$. Suppose that the subset $E(\{u\}, V_{=1})$ has at least $k + 1$ edges of distinct colors. Then, delete an edge from $E(\{u\}, V_{=1})$. The resulting instance is denoted by $(G', \chi|_{G'}, k)$.*

Lemma 6.5.6. *The Reduction Rule 6.5.2 is safe.*

Proof. We will prove that G has a colorful matching of size k if and only if G' has a colorful matching of size k .

We first prove the forward direction. Let M be a solution for the instance (G, χ, k) . Let uv be the deleted edge. If $uv \in E(G) \setminus M$, then M is a solution for $(G', \chi|_{G'}, k)$. Suppose that $uv \in M$. If $|M| \geq k + 1$, then $M \setminus \{uv\}$ is a solution for $(G', \chi|_{G'}, k)$. Suppose that $|M| = k$. Then, we will show a colorful matching of size k exists in G' by adding an edge to $M \setminus \{uv\}$.

After we delete the edge uv there are at least k distinct colors that appear on edges adjacent to u . Therefore, at least one of these colors is not present in $M \setminus \{uv\}$. Let uv' denote this edge. We claim that $M' = M \setminus \{uv\} \cup \{uv'\}$ is a colorful matching of size k .

Note that M' is matching because $M \setminus \{uv\}$ is a matching, and neither u nor v' is incident to any edge in $M \setminus \{uv\}$. So the edges of M' are pairwise vertex disjoint, and so it is a matching. It is a colorful matching because $M \setminus \{uv\}$ is colorful and the $\chi(uv')$ is not present in the set $M \setminus \{uv\}$. Moreover, $|M'| = |M| = k$. Therefore, M' is a solution for $(G', \chi|_{G'}, k)$.

For the reverse direction, let M' be a colorful matching of size k of G' . Since every edge in G' is an edge in G , M' is a solution for (G, χ, k) . This completes the proof. $\diamond$

As a corollary to our color reducing rule and reducing edges of the same color we get the following.

Corollary 6.5.1. *If both Reduction Rules [6.5.1](#) and [6.5.2](#) are not applicable, then for any vertex $u \in \mathcal{X}$, we have $\deg_{V_{=1}}(u) \leq k$.*

Notice that each vertex $u \in \mathcal{X}$ along with its neighbors in $V_{=1}$ forms a star. Intuitively speaking, we will bound the size of each star. Recall that the vertices of the matching that we constructed in Lemma [6.5.3](#) (or Lemma [6.5.2](#)) is denoted by $V(\mathcal{M})$. Notice that $V(\mathcal{M}) \subseteq \mathcal{X}$ and is the vertex cover for the subgraph $G[\cup_{i \in [q] \setminus [\ell]} E_i]$ which implies that none of the edges in color classes $\cup_{i \in [q] \setminus [\ell]} E_i$ have any neighbors in $\mathcal{X} \setminus V(\mathcal{M})$.

Lemma 6.5.7. *Let $V_{=1}^{\ell+} = \{v \in V_{=1} \mid N(v) \cap V(\mathcal{M}) \neq \emptyset\}$. Then, $|V_{=1}^{\ell+}| = \mathcal{O}(k^2)$.*

Proof. Recall that $|V(\mathcal{M})| \leq 2(k-1)$. From Corollary [6.5.1](#), for each $u \in V(\mathcal{M})$, $\deg_{V_{=1}}(u) \leq k$. Therefore $|N_{V_{=1}}(V(\mathcal{M}))| \leq 2(k-1)k$. The set $N_{V_{=1}}(V(\mathcal{M}))$ is the

set vertices in $V_{=1}$ that has a neighbor in $V(\mathcal{M})$. That is, $V_{=1}^{\ell+} = N_{V_{=1}}(V(\mathcal{M}))$, and so the lemma is proved. $\diamond$

Next, for a fixed color $i \in [q]$, we bound the size of the set $V(E_i) \cap V_{=1}$. We do so by identifying several stars centered on $\mathcal{X}$ but with leaves in $V_{=1}$ such that each one has an edge of color i .

Reduction Rule 6.5.3. *Let $i \in [q]$. If there are at least $2k$ vertices in $\mathcal{X}$ that are adjacent to (some edges in) E_i , then delete an edge in E_i . The resulting instance is denoted by $(G', \chi|_{G'}, k)$*

Lemma 6.5.8. *The Reduction Rule 6.5.3 is safe.*

Proof. We will prove that G has a colorful matching of size k if and only if G' has a colorful matching of size k .

Let M be a solution for the instance (G, χ, k) . Let uv be the deleted edge. If $uv \in E(G) \setminus M$, then M is a solution for (G', χ', k) . Furthermore, we assume that $|M| = k$, or else $M \setminus \{uv\}$ is a solution for (G', χ', k) . We will produce a colorful matching of size k in G' . Let $M' = M \setminus \{uv\}$.

Since $|M'| = k - 1$, we have $|V(M')| = 2(k - 1)$. Thus, $|V(M') \cap \mathcal{X}| \leq |V(M')| = 2(k - 1)$. Note that $|V(M') \cap V_{=1}| \leq k - 1$ because at most one of the endpoints in the $k - 1$ edges in M' can be in the independent set $V_{=1}$. Thus, there can be at most $2(k - 1)$ edges in $E(\mathcal{X}, V_{=1})$ that have at least one endpoint in $V(M')$. In particular, there can be at most $2(k - 1)$ edges of color i in $E(\mathcal{X}, V_{=1})$ that have at least one endpoint in $V(M')$.

Thus, it follows that there is at least one edge in E_i whose both the endpoints are in $V(G') \setminus V(M')$. Let xy denote this edge. We claim that $M'' = M' \cup \{xy\}$ is a colorful matching of size k . Since $\{x, y\} \subseteq V(G') \setminus V(M')$, the edges of M'' are pairwise vertex disjoint, so M'' is indeed a matching. It is a colorful matching

because M is a colorful matching and $uv, xy \in \mathcal{E}_i$, meaning that they both have color i . Moreover, $|M| = |M''|$. Hence, we can conclude that M'' is a solution for $(G', \chi|_{G'}, k)$.

For the other direction, we note that G' is a induced subgraph of G . Thus, if M is a solution for (G', χ, k) , then M is a solution for (G, χ, k) . $\diamond$

Finally, we bound the number of remaining vertices in $V_{=1}$ i.e., the vertices in $V_{=1} \setminus V_{=1}^{\ell+}$.

Lemma 6.5.9. $|V_{=1} \setminus V_{=1}^{\ell+}| = \mathcal{O}(k^2)$.

Proof. Observe that $V_{=1}^{\ell+} = \{v \in V_{=1} | N(v) \in V(\mathcal{M})\}$. Since $V(\mathcal{M})$ is the vertex cover of $G[\cup_{i \in [q] \setminus [\ell]} \mathcal{E}_i]$, it follows that $V_{=1}^{\ell+}$ contains each of the vertices in $V_{=1}$ that is incident on an edge with color $i \in [q] \setminus [\ell]$. Therefore, the remaining set, $V_{=1} \setminus V_{=1}^{\ell+}$, is contained in the set of vertices which are incident on edges that have color $i \in [\ell]$. That is, $V_{=1} \setminus V_{=1}^{\ell+} \subseteq \cup_{i=1}^{\ell} V(E_i) \cap V_{=1}$. Due to Reduction Rule [6.5.3](#), for each $i \in [\ell]$, $|V(E_i) \cap V_{=1}| \leq 2(k-1) + 1$. Since $\ell \leq k-1$, the lemma follows. $\diamond$

Note that we can find an edge that satisfy the premise of Reduction Rule [6.5.1](#) in time $\mathcal{O}(n)$. The graph can be updated in time $\mathcal{O}(n^2)$. Hence, an application of the rule takes time $\mathcal{O}(n^2)$. Similarly, an application of Reduction Rule [6.5.2](#) and [6.5.3](#) can be executed in time $\mathcal{O}(n^2)$. Hence, Theorem [17](#) follows by combining Lemmas [6.5.3](#), [6.5.7](#) and [6.5.9](#).

6.5.3 Kernelization on General Graphs: Algorithm II

In this section we describe a kernel on general graphs. That is, we give a proof of Theorem [16](#). Recall that by [15](#) we know that RAINBOW MATCHING admits a kernel with $\mathcal{O}(k^3)$ edges and vertices. Let (G, χ, k) be an instance of RAINBOW MATCHING.

We start by applying the kernelization algorithm described in Section [6.4.1](#) on (G, χ, k) and obtain an equivalent instance (G', χ', k') such that G' has $\mathcal{O}(k^3)$ edges and $k' \leq k$. For the sake of clarity of presentation we will denote the reduced instance also by (G, χ, k) . Each application of the reduction rules in this section either deletes an edge or deletes a vertex. Hence, the number of edges in the resulting instance cannot increase. Hence, from now onwards we assume that the instance returned by our kernelization algorithm has $\mathcal{O}(k^3)$ edges.

Let (G, χ, k) be an instance of RAINBOW MATCHING. In the next reduction rule, we bound the the number of colors (on edges) incident on a vertex. Observe that we had similar rule on forests but those were only applicable to “star” kind of vertices.

Reduction Rule 6.5.4. *Let $v \in V(G)$. If the number of distinct colors incident on v is at least $3k - 1$, then delete an edge incident on v . The resulting instance is denoted by $(G', \chi|_{G'}, k)$*

Lemma 6.5.10. *The Reduction Rule [6.5.4](#) is safe.*

Proof. We will prove that G has a colorful matching of size k if and only if G' has a colorful matching of size k .

We first prove the forward direction. Let M denote a solution for the instance (G, χ, k) . Let uv denote the deleted edge. If $uv \in E(G) \setminus M$, then M is a solution for (G', χ', k) . Suppose that $uv \in M$. If $|M| \geq k + 1$, then $M \setminus \{uv\}$ is a solution for (G', χ', k) . Suppose that $|M| = k$. We will produce a new colorful matching of size k by adding an edge to $M \setminus \{uv\}$. Let M' denote the colorful matching $M \setminus \{uv\}$, and let $V(M')$ denote the set of vertices incident on edges in M' . Thus, we have $|V(M')| = 2(k - 1)$.

Note that after we delete uv there are at least $3k - 2$ edges of distinct color that are incident on v . So we can infer that at least k of the edges incident on v in G' are not incident on a vertex in $V(M')$ and are of distinct colors. We denote

these edges by $\{vu_1, \dots, vu_h\}$, where $h \geq k$ and $\{u_i \mid 1 \leq i \leq h\} \cap V(M') = \emptyset$. By definition, edges $vu_1, \dots, vu_h$ have distinct colors. Since there are $k - 1$ colors in M' , there exists $j \in [h]$ such that none of the edges in M' have the color $\chi(vu_j)$. Let $M'' = M' \cup \{vu_j\}$. Clearly, M'' has edges of distinct colors because of the choice of $\chi(vu_j)$ and the fact that M' is a colorful matching. To see that M'' is indeed a matching, that is, it is a set of vertex-disjoint edges we note that M' is a matching and that $\{v, u_j\} \cap V(M') = \emptyset$. Thus, M'' is a colorful matching of size k , and so is a solution for (G', χ', k) .

For the reverse direction, let M' denote a colorful matching of size k in G' . Since every edge in G' is an edge in G , M' is a solution for (G, χ, k) . This completes the proof. $\diamond$

Next we will use **Proposition 1** to bound the size of the independent set $\mathcal{I}$. But prior to that, we will create a more enriched set from $\mathcal{X}$.

An “expansion graph”

We begin by recalling the graph decomposition described in Lemma 6.5.3: $\mathcal{X} = \cup_{i \in [\ell]} X_i \cup V(\mathcal{M})$ is a vertex cover and $\mathcal{I} = V \setminus \mathcal{X}$ is the resulting independent set in G .

For each $v \in V(\mathcal{M})$, we define $\mathcal{C}(v)$ to be a inclusion-wise maximal set of distinct colors incident on v . We define a bipartite (helper) graph $\mathcal{H}_G$ with the vertex bipartition $(\mathcal{A}, \mathcal{B})$ where $\mathcal{B} = \mathcal{I}$ and the subset $\mathcal{A}$ is defined as follows: For each vertex $x \in X_j$, $j \in [\ell]$, $\mathcal{A}$ contains the vertex x^j . Additionally, for each $x \in V(\mathcal{M})$ and each $j \in \mathcal{C}(x)$, $\mathcal{A}$ contains the vertex x^j . Observe that vertices x^i and x^j are distinct if $i \neq j$. Indeed, a vertex can appear in both X_i and $V(\mathcal{M})$ and thus we might get duplicates of a vertex in $\mathcal{A}$, in which case we delete all but one copy of a vertex. We define the edges as follows. For $a^i \in \mathcal{A}$ and $b \in \mathcal{I}$, $a^i b \in E(\mathcal{H}_G)$ if and

only if $ab \in E(G)$ and color $\chi(ab) = i$.

Now we show the following properties about the graph $\mathcal{H}_G$.

Lemma 6.5.11. $|\mathcal{A}| = \mathcal{O}(k^2)$.

Proof. Since we apply Reduction Rule [6.5.4](#) exhaustively, for any vertex $v \in V(G)$, the set of edges with distinct colors incident on v is at most $3k - 2$. Thus, for any $v \in V(\mathcal{M})$, $|\mathcal{C}(v)| \leq 3k - 2$. Hence, $|\cup_{x \in V(\mathcal{M})} \mathcal{C}(x)| \leq 2(k - 1)(3k - 2)$ because $|V(\mathcal{M})| \leq 2(k - 1)$.

Moreover, from Lemma [6.5.3](#), we know that $|X_i| \leq 4(k - 1)$, for each $i \in [\ell]$. Hence, $|\{x^i \mid x \in X_i\}| \leq 4(k - 1)$, for each $i \in [\ell]$. Thus, $|\cup_{i \in [\ell]} \{x^i \mid x \in X_i\}| \leq 4(k - 1)(k - 1)$ because $\ell \leq k - 1$. Hence, $|\mathcal{A}| = |\cup_{i \in [\ell]} \{x^i \mid x \in X_i\} \cup \cup_{x \in V(\mathcal{M})} \mathcal{C}(x)| \leq 4(k - 1)^2 + 2(k - 1)(3k - 2) = \mathcal{O}(k^2)$. $\diamond$

Lemma 6.5.12. $\mathcal{B}$ does not have an isolated vertex.

Proof. We recall that $\mathcal{B} = \mathcal{I}$. Let $b \in \mathcal{B}$. Due to Reduction Rule [6.5.1](#), b is not an isolated vertex in G . That is, $N_G(b) \neq \emptyset$. Let $u \in N_G(b)$ and $j = \chi(ub)$. Since, $\mathcal{B} \subseteq \mathcal{I}$ and $\mathcal{X}$ is a vertex cover of G , it follows that $N_G(b) \subseteq \mathcal{X}$. We claim that $u^j \in \mathcal{A}$, and therefore, $u^j b \in E(\mathcal{H}_G)$. We prove the claim as follows.

If $j \in [\ell]$, then consider the graph $G[E_j]$, it contains the edge ub . Due to the fact that $b \in \mathcal{I}$, it is clear that the vertex cover X_j of $G[E_j]$ must contain the vertex u . Hence, $u^j \in \mathcal{A}$ follows straightaway. Suppose that $j \in \{\ell + 1, \dots, q\}$. Then, the edge ub is in the subgraph $G[\cup_{j \in [q] \setminus [\ell]} E_j]$. Recall that $V(\mathcal{M})$ is a vertex cover for the graph $G[\cup_{j \in [q] \setminus [\ell]} E_j]$. Thus, the edge ub must be “covered” by some vertex in $V(\mathcal{M})$. Since $V(\mathcal{M}) \subseteq \mathcal{X}$ and $b \in \mathcal{I}$, it follows that $u \in V(\mathcal{M}) \setminus \cup_{i \in [\ell]} E_i$. Thus, it follows that $\chi(uv) = j \in \mathcal{C}(u)$. This implies that $u^j \in \mathcal{A}$, and so our proof is complete. $\diamond$

Using Lemma [6.5.12](#), there is no isolated vertices in $\mathcal{B}$. Moreover, if $|\mathcal{B}| > |\mathcal{A}|$, then we apply 1-Expansion Lemma (Proposition [1](#)) on $\mathcal{H}_G$ to find a redundant vertex

that can be deleted. The details are presented in the following reduction rule. Note that the conditions required to apply the algorithm provided by the lemma are satisfied.

Reduction Rule 6.5.5. *Suppose that in the bipartite graph $\mathcal{H}_G$, we have $|\mathcal{B}| > |\mathcal{A}|$. Then, we invoke Proposition [1](#) to compute sets $X \subseteq \mathcal{A}$ and $Y \subseteq \mathcal{B}$ with $|X| < |Y|$ such that X has a 1-expansion, denoted by $\mathcal{M}^*$, into Y in $\mathcal{H}_G$ and $N(Y) \subseteq X$. Let $y \in Y \setminus V(\mathcal{M}^*)$, a vertex that is not incident to any edges in $\mathcal{M}^*$.*

The resulting instance is denoted by $(G', \chi|_{G'}, k)$, where G' is the graph obtained from G by deleting y and all edges incident on it.

Next we prove the correctness of Reduction Rule [6.5.5](#).

Lemma 6.5.13. *Reduction Rule [6.5.5](#) is safe.*

Proof. We will prove that G has a colorful matching of size k if and only if G' has a colorful matching of size k . We first prove the forward direction. Let M be a solution for the instance (G, χ, k) .

If the deleted vertex $y \in V(G) \setminus V(M)$, then M does not contain any edge that was deleted from G . Thus, M is also a solution of $(G', \chi|_{G'}, k)$. Suppose that $y \in V(M)$, there exists an edge $uy \in M$ and $|M| = k$. Using the 1-expansion of X into Y (denoted by $\mathcal{M}^*$) we will show that there exists a colorful matching of size k in G' . Given a graph G , a matching L and a vertex $v \in V(L)$ by $L(v)$ we denote the *matching partner* of the vertex v . In other words, the other end-point of the edge in which the vertex v appears.

Let $M = \{x_i y_i \mid 1 \leq i \leq k\}$, where $y_k = y$ denotes the deleted vertex. Thus, it follows that $y_k \in Y$. Furthermore, let the color $c_i = \chi(x_i y_i)$ for each $i \in [k]$. Since $N(Y) \subseteq X$, we can infer that $x_k^{c_k} \in X$. We conclude that the set of edges $\{x_i^{c_i} y_i \mid i \in [k], c_i = \chi(x_i y_i)\}$ is present in $\mathcal{H}_G$. The 1-expansion $\mathcal{M}^*$ in $\mathcal{H}_G$ yields

the set $\{x^c \mathcal{M}^*(x^c) \mid x^c \in X \text{ and } \mathcal{M}^*(x^c) \in Y\}$ in $\mathcal{H}_G$. We will use a subset of this to exhibit a colorful matching in G' .

Consider $X_G = \{x \mid \exists j \in [q], x^j \in X\}$, the set of vertices in G that correspond to vertices in $X \subseteq \mathcal{A}$. Since $x_k^{c_k} \in X$, it follows that $x_k \in X_G$. Without loss of generality, let $V(M) \cap X_G$ be denoted by $\{x_i \mid i \in [k] \setminus [r]\}$ for some $r \geq 1$. Then, it follows that $\{x_i^{c_i} \mid i \in [k] \setminus [r]\} \subseteq X$. It follows that $\{\mathcal{M}^*(x_i^{c_i}) \mid i \in [k] \setminus [r]\} \subseteq Y$, and that G' contains the edges $\{x_i \mathcal{M}^*(x_i^{c_i}) \mid x_i \in X_G \cap V(M)\}$.

The following claim completes the proof of the forward direction.

Claim 1. $\widetilde{M} = M \setminus \{x_i y_i \mid i \in [k] \setminus [r]\} \cup \{x_i \mathcal{M}^*(x_i^{c_i}) \mid i \in [k] \setminus [r]\}$ is a colorful matching of size k in G' .

Proof. We will first prove that $\widetilde{M}$ is a set of pairwise vertex-disjoint edges, *i.e.* $\widetilde{M}$ is indeed a matching. First we note that $M \setminus \{x_i y_i \mid i \in [k] \setminus [r]\} = \{x_i y_i \mid i \in [r]\}$ is a set of pairwise vertex-disjoint edges. Clearly, these edges do not share any vertex with $\{x_i \mid i \in [k] \setminus [r]\}$. We claim that nor do they share a vertex with $\{\mathcal{M}^*(x_i^{c_i}) \mid i \in [k] \setminus [r]\}$, that in fact, $\{y_i \mid i \in [r]\} \cap \{\mathcal{M}^*(x_i^{c_i}) \mid i \in [k] \setminus [r]\} = \emptyset$.

For the sake of contradiction, we assume that $y_a = \mathcal{M}^*(x_b^{c_b})$ where $a \in [r]$ and $b \in [k] \setminus [r]$. We know that in the graph $\mathcal{H}_G$, $\mathcal{M}^*(x_b^{c_b}) \in Y$, so now $y_a \in Y$. Since $N_{\mathcal{H}_G}(Y) \subseteq X$, it follows that $x_a^{c_a} \in X$ because $x_a^{c_a} \in N_{\mathcal{H}_G}(y_a)$. Thus, in the graph G , we have $x_a \in X_G$. Furthermore, $x_a \in V(M) \cap X_G$ because edge $x_a y_a \in M$. But, by definition $V(M) \cap X_G = \{x_i \mid i \in [k] \setminus [r]\}$, a contradiction. Hence, we can conclude that $\widetilde{M}$ is a matching and $|\widetilde{M}| = |M| = k$.

Next we will prove that $\widetilde{M}$ is colorful. Clearly, $M \setminus \{x_i y_i \mid i \in [k] \setminus [r]\}$ is colorful because M is colorful. The set of colors appearing in the set $\{x_i \mathcal{M}^*(x_i^{c_i}) \mid i \in [k] \setminus [r]\}$ is same as that in $\{x_i y_i \mid i \in [k] \setminus [r]\}$ which is in fact, $\{c_i \mid i \in [k] \setminus [r]\}$. Thus, $\widetilde{M}$ has the exact same colors as M .

Hence, the claim is proved. $\diamond$

For the reverse direction, let M' be a colorful matching of size k of G' . Since every edge in G' is an edge in G , M' is a solution for (G, χ, k) . This completes the proof. $\diamond$

If we cannot apply Reduction Rule [6.5.5](#) on an instance (G, χ, k) , then $|\mathcal{B}| \leq |\mathcal{A}|$ in the graph $\mathcal{H}_G$. Hence, $|\mathcal{B}| \leq |\mathcal{A}| = \mathcal{O}(k^2)$, by Lemma [6.5.11](#), and G has $\mathcal{O}(k^2)$ vertices. An application of Reduction Rule [6.5.4](#) takes time $\mathcal{O}(n^2)$. We can construct the helper bipartite graph in time $\mathcal{O}(qn)$. Hence, Reduction Rule [6.5.5](#) can be executed in time $\mathcal{O}(nm\sqrt{n})$. Thus, Theorem [16](#) is proved.

6.6 Conclusion, Discussion and Open Problems

In this chapter, we considered RAINBOW MATCHING from the viewpoint of parameterized complexity, and designed faster parameterized algorithms as well as kernels for this problem. RAINBOW MATCHING is easily seen as a generalization of another well studied problem in parameterized algorithms, namely 3-DIMENSIONAL MATCHING, when we allow the input graph to be a multigraph. In this problem, we are given a set family $(U, \mathcal{F})$, together with a partition $U = \uplus_{i=1}^3 U_i$ and a positive integer k . Here, every set $F \in \mathcal{F}$ has the property that for all $i \in [3]$, $|F \cap U_i| = 1$. The question is whether there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ containing k pairwise-disjoint sets. We first show that RAINBOW MATCHING is indeed a generalization of 3-DIMENSIONAL MATCHING. Towards proving this, we give a polynomial time parameter-preserving reduction from 3-DIMENSIONAL MATCHING to RAINBOW MATCHING. That is, we give the following ppt reduction,

$$3\text{-DIMENSIONAL MATCHING} \leq_{\text{ppt}} \text{RAINBOW MATCHING}.$$

Here, let us only present a rough sketch of the proof. The idea of the con-

struction is as follows. In the bipartite graph of the constructed instance of RAINBOW MATCHING, one side represents the elements of U_1 , and the other side represents the elements of U_2 . Then, for every set $\{u_1, u_2, u_3\}$ in $\mathcal{F}$, where $u_i \in U_i$ for all $i \in [3]$, we add an edge between u_1 and u_2 whose color is u_3 . It is easy to see that a solution for the original problem instance can be directly translated to a solution for the new problem instance, and vice versa. Moreover, the parameter k in both instances is set to be the same.

We also saw that there is a ppt reduction from RAINBOW MATCHING to 3-SET PACKING and thus we have the following chain of reductions.

$$3\text{-DIMENSIONAL MATCHING} \leq_{\text{ppt}} \text{RAINBOW MATCHING} \leq_{\text{ppt}} 3\text{-SET PACKING}.$$

It is known that 3-DIMENSIONAL MATCHING admits a randomized algorithm with running time $\mathcal{O}^*(2^k)$ [16] and a deterministic algorithm with running time $\mathcal{O}^*(2.5961^{2k}) = \mathcal{O}^*(6.7398^k)$ [158]. We gave in the introduction a deterministic algorithm for RAINBOW MATCHING that is the same as the one for 3-SET PACKING. However, we remark that the algorithm for 3-DIMENSIONAL MATCHING given in [158] can actually be used to solve RAINBOW MATCHING in $\mathcal{O}^*(6.7398^k)$ time. Can we design a faster randomized or deterministic algorithm for RAINBOW MATCHING or even 3-DIMENSIONAL MATCHING?

We designed a kernel with $\mathcal{O}(k^2)$ vertices on general graphs. This also implied a kernel with $\mathcal{O}(k^2)$ vertices on paths, forests and planar graphs. We conclude with the following open problems. Does there exist a linear kernel on paths or forests? Is the $\mathcal{O}(k^3)$ bound on the edges of the kernel for RAINBOW MATCHING optimal for general graphs?

Finally, by a direct application of our randomized parameterized algorithm for RAINBOW MATCHING running in time $\mathcal{O}^*(2^k)$, we have that there exists a randomized

algorithm for RAINBOW MATCHING running in time $\mathcal{O}^*(2^{n/2}) = \mathcal{O}^*(1.4143^n)$. Here, n is the number of vertices in the input graph and the $n/2$ is an upper bound on the maximum size of a colorful matching in a graph. Using a simple dynamic programming algorithm, it is possible to design a $\mathcal{O}^*(2^n)$ algorithm for RAINBOW MATCHING. Designing a deterministic algorithm for RAINBOW MATCHING running in time $(2 - \epsilon)^n$ for some fixed $\epsilon > 0$ is another interesting open problem.

Chapter 7

Stable Committee Selection

In the previous chapter we gave kernel and FPT algorithms for a type of matching problem. Here, we shift our attention from graph theoretic problem to a problem in computational social choice. We will give an algorithm to select a subset of vertices (candidates) that satisfy some criteria. Whereas, in the previous chapter we wanted the selected set of edges to be colorful, here we want the selected set of vertices to satisfy a stability condition. This stability condition is different from the ones we have seen in Part I of the thesis.

An important question in social choice theory is—“how to choose a non-controversial committee of size k ?” Such a question arises while electing parliaments in modern democracies, selecting a group of representatives in an organization, in taking business decisions or shortlisting tasks. In voting theory, all these scenarios can be captured by *multiwinner elections*. In particular, the problem of selecting a committee can be formulated as follows. Given a set of candidates and a set of voters with strict preference ranking over the candidates, find a committee of size k satisfying certain acceptability criteria. However, what acceptability criteria should be chosen? For a single winner election, Condorcet [26] suggested that a candidate can be considered as winner if s/he is preferred by at least half of the voters over every other candidate.

Of course, such a candidate may not exist. Fishburn [48] generalized the idea of Condorcet from a single winner election to a committee. This definition of *Condorcet committee* requires that each voter has explicit preferences over the committees, or there is some way to infer these preferences. According to Fishburn, a committee is a Condorcet committee if it is preferred by at least half of the voters over any other committee of the same size. Darmann [31] defined two notions of Condorcet committee, *weak* and *strong*, where preferences over the committees are implicit. Specifically, a committee of a given size k is weak (strong) Condorcet, if it is preferred over any other committee of size k by at least (more than) half of the voters. He used *scoring functions*, which are used in multiwinner voting rules, for comparing two committees. The problems corresponding to finding a weak (strong) Condorcet committee of size k are WEAK (STRONG) CONDORCET k -COMMITTEE.

Gehrlein [64] defined a new notion of Condorcet committee by considering each candidate of the committee instead of whole committee. According to his definition, a committee is Condorcet if a candidate in the committee is preferred by at least half of the voters to each non-member. Note that such a committee might not exist. Moreover, a committee is called *weakly (strongly) Gehrlein-stable* if every candidate c in the committee is preferred by at least (more than) half of the voters in the pairwise election between c and every non-committee member d . We would like to point out here that when the number of voters is odd, weakly and strongly Gehrlein-stable committees are equivalent. However, this is not the case when the number of voters is even. We remark that, in the literature, there are also other notions of Condorcet committee [42, 95].

For the committee selection problem, extensive research has been conducted to study voting rules and their stability in the context of selecting a committee [25, 41, 91, 144]. Darmann [31] analyzed the computational complexity of WEAK and STRONG CONDORCET k -COMMITTEE. He studied the problem with different voting rules,

including Borda voting, plurality voting, antiplurality voting, and t -approval, where $t \geq 2$. Specifically, he proved that WEAK and STRONG CONDORCET k -COMMITTEE are coNP-hard under Borda and 2-approval voting schemes. Furthermore, he showed that WEAK and STRONG CONDORCET k -COMMITTEE are polynomial time solvable under plurality and antiplurality voting schemes. For more literature on multiwinner elections, we refer to [45].

Recently, Aziz et al. [7, 8] studied the computational complexity of finding a Gehrlein-stable committee of size k . They proved that finding a strongly Gehrlein-stable committee of size k (and determining that one exists) can be done in polynomial time. However, computing a weakly Gehrlein-stable committee of size k is NP-hard. Aziz et al. [7, 8] proposed to study this problem from the perspective of parameterized complexity and exact exponential time algorithms. In this article, we initiate a systematic study of finding a weakly Gehrlein-stable committee of size k in the realm of parameterized complexity. We call this problem as GEHRLEIN STABLE COMMITTEE SELECTION (GSCS).

We first show that GSCS is W[1]-hard when parameterized by the size of the committee. That is, the problem is unlikely to admit an algorithm with running time $f(k)n^{\mathcal{O}(1)}$. To overcome this intractability result, we seek relevant alternate parameters that could lead to tractable algorithms. To achieve this, we consider a model of GSCS as a problem on directed graphs and then use parameters that measure the “structure” of these directed graphs. In particular, we consider the majority graph [104], defined as follows. Given any election $\mathcal{E} = (\mathcal{C}, \mathcal{V})$, we define the majority graph $\mathcal{M}_{\mathcal{E}} = (\mathcal{C}, \mathcal{A})$ on the vertex set $\mathcal{C}$ and an arc $(c, c') \in \mathcal{A}$ if and only if candidate c is more popular than c' in the election $\mathcal{E}$ (denoted by $c >_{\mathcal{E}} c'$). In other words, $(c, c') \in \mathcal{A}$ if and only if the number of voters that *prefer* c over c' is strictly more than those preferring c' over c . For example, Figure 7.1 illustrates the majority graph corresponding to the election in Table 7.1.

Table 7.1: Example: Voting profile; v_1, v_2, v_3, v_4 are the voters, $\{1, 2, \dots, 10\}$ is the set of candidates.

Voters	Preference Ranking over the Candidates									
v_1	6	3	7	8	5	1	2	4	9	10
v_2	6	1	7	4	9	5	8	3	10	2
v_3	2	10	8	9	1	5	7	6	3	4
v_4	2	10	4	5	3	8	7	1	6	9

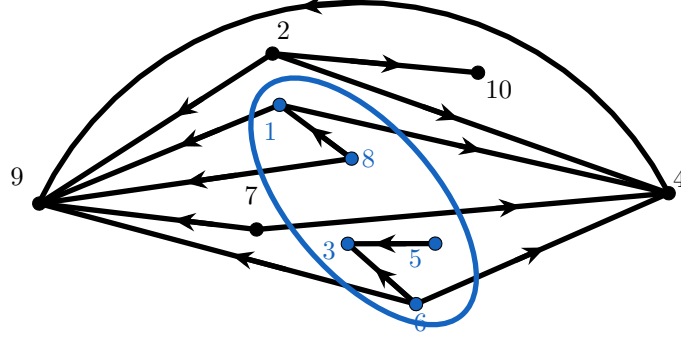


Figure 7.1: Example: The **blue** vertices in the set S is a weakly Gehrlein stable committee of size 5 for the voting profile given in Table 7.1.

Now, note that if a committee S of size k is stable in an election $\mathcal{E} = (\mathcal{C}, \mathcal{V})$, then there does not exist a candidate $u \in \mathcal{C} \setminus S$ that is preferred over any candidate in S in the pairwise election between u and the candidates of S . This implies that there does not exist a vertex $u \in \mathcal{C} \setminus S$ such that $(u, v) \in \mathcal{A}(\mathcal{M}_{\mathcal{E}})$ for some $v \in S$. Hence, for any $v \in S$, all the in-neighbors of v in the graph $\mathcal{M}_{\mathcal{E}}$ belong to S . Thus, the problem of finding WGS committee of size k corresponds to finding a vertex subset X of size k in the majority graph with the following property: the set X contains no vertex that has an in-neighbor outside X . We will use this formulation of the problem in the chapter. In particular, we will study the following problem.

GEHRLEIN STABLE COMMITTEE SELECTION (GSCS)

Input: A majority graph $\mathcal{M}_{\mathcal{E}} = (\mathcal{C}, \mathcal{A})$ for an election $\mathcal{E}$ and a positive integer k such that $k \leq |\mathcal{C}|$.

Task: Does there exist a subset of vertices $S \subseteq \mathcal{C}$, $|S| = k$ such that for every $v \in S$, each of the in-neighbors of v (if any) lies in S ?

Such a set S is a *solution* to the problem.

Our algorithmic contributions. One way to discover relevant parameters for studying a graph problem is to find a family of graphs, say $\mathcal{F}$, where the problem is polynomial time solvable; then, the problem is studied with an *edit distance*—the number of vertices/edges deleted (or edges added) to transform the input graph into a graph in $\mathcal{F}$ —as a parameter. Aziz et al. [7, 8] showed that GSCS is polynomial time solvable when the number of voters is odd. Note that when the number of voters is odd, then the majority graph is a tournament. Thus, natural parameters to study our problem correspond to vertex/edge editing operations into the family of tournaments. We study GSCS with two “editing parameters”: the number of missing arcs in the given directed graph (l) and the size of a *tournament vertex deletion set* ($\mathbf{tvd}$)(q)—that is, a subset of vertices whose deletion from the given directed graph results in a tournament. The number l corresponds to the number of pairs of candidates which are *tied* among each other in the pairwise election and q could be thought of as the smallest subset of candidates who are in a tie with some candidate(s) such that the deletion of this subset will render the resulting majority graph a tournament. Since $\mathbf{tvd}$ is smaller than the number of candidates who are in a tie, it makes $\mathbf{tvd}$ a natural parameter to study from a computational perspective. Note that $\mathbf{tvd}$ is a vertex cover (a set of vertices such that each edge is incident to at least one vertex of the set) of the complement graph of the underlying undirected graph of $\mathcal{M}_\mathcal{C}$. Hence, it can be computed using the FPT algorithm proposed by Chen et al. [24] in time $\mathcal{O}^*(1.2738^q)$. We show that the problem is fixed parameter tractable (FPT) and admits linear kernels with respect to the parameters l and q . In particular, we obtain the following results.

- GSCS can be solved optimally in $\mathcal{O}^*(1.2207^n)$ time. Here, n denotes the number of candidates ($|\mathcal{C}|$). This resolves a question asked in the conclusion of

¹The $\mathcal{O}^*$ notation suppresses the polynomial dependence on the input size.

Aziz et al. [8].

- GSCS admits an FPT algorithm with running time $\mathcal{O}^*(1.2738^q)$.
- GSCS admits a kernel with $4q + 1$ vertices. That is, there is a polynomial time algorithm that given an instance $(\mathcal{M}_{\mathcal{E}}, k)$ of GSCS returns an instance $(\mathcal{M}'_{\mathcal{E}}, k')$ of GSCS such that $(\mathcal{M}_{\mathcal{E}}, k)$ is a Yes-instance if and only if $(\mathcal{M}'_{\mathcal{E}}, k')$ is a Yes-instance and $|V(\mathcal{M}'_{\mathcal{E}})| \leq 4q + 1$.
- GSCS admits an algorithm with running time $\mathcal{O}^*(1.2207^l)$ and has a kernel with $2l + 1$ vertices. These results are obtained as corollaries to the results for the parameter q .

7.1 Preliminaries

We recall some of the basic notations for directed graphs here.

Graphs. Let G be a directed graph. We denote an arc from u to v by an ordered pair (u, v) , and say that u is an in-neighbor of v and v is an out-neighbor of u . For $x \in V(G)$, $N_G^-(x) = \{y \in V(G) : (y, x) \in \mathcal{A}(G)\}$ and $N_G^+(x) = \{y \in V(G) : (x, y) \in \mathcal{A}(G)\}$. For $X \subseteq V(G)$, $G - X$ and $G[X]$ denote subgraphs of G induced on the vertex set $V(G) \setminus X$ and X , respectively. For $v_1, v_t \in V(G)$, a directed path from v_1 to v_t is denoted by $P = (v_1, v_2, \dots, v_t)$, where $V(P) \subseteq V(G)$ and for each $i \in [t - 1]$, $(v_i, v_{i+1}) \in \mathcal{A}(G)$. In a directed graph G , we say a vertex u is reachable from a vertex v , if there is directed path from v to u . A graph is called a *strongly connected component* if every vertex in the graph is reachable from every other vertex. Let $X \subseteq V(G)$. A strongly connected component, $G[X]$, is called maximal if there does not exist a vertex $v \in V(G) \setminus X$ such that $G[X \cup \{v\}]$ is also a strongly connected component. Let $x \in V(G)$. We define two sets $R_G^-(x)$ and $R_G^+(x)$ as follows. $R_G^-(x) = \{x\} \cup \{y \in V(G) : x \text{ is reachable from } y\}$ and

$R_G^+(x) = \{x\} \cup \{y \in V(G) : y \text{ is reachable from } x\}$. We call $R_G^-(x)$ and $R_G^+(x)$ as *in-reachability* set and *out-reachability* set of x in G , respectively. For $S \subseteq V(G)$, $R_G^-(S) = \cup_{v \in S} R_G^-(v)$ and $R_G^+(S) = \cup_{v \in S} R_G^+(v)$. The subscript in the notation for the neighborhood and the reachability sets may be omitted if the graph under consideration is clear from the context. A *topological ordering* of a directed graph G is an ordering, denoted by τ , of the vertices of $V(G)$ such that for every arc $(u, v) \in \mathcal{A}(G)$, we have $\tau(u) < \tau(v)$. Given an undirected graph G , complement of G is a graph G' such that $V(G') = V(G)$ and $E(G') = \{uv : u, v \in V(G) \text{ and } uv \notin E(G)\}$.

7.2 Structural Observations

We start by making some simple observations that are crucial for most of our arguments. The proof of the next lemma follows from the definition of solution.

Lemma 7.2.1. *Let $(\mathcal{M}_\varepsilon, k)$ be a Yes-instance of GSCS, and let S be a solution. Furthermore, let v_1 and v_t denote two vertices in $\mathcal{M}_\varepsilon$ such that there exists a path from v_1 to v_t in $\mathcal{M}_\varepsilon$. If $v_t \in S$, then $v_1 \in S$.*

Proof. Let $(v_1, \dots, v_t)$ denote a path from v_1 to v_t . Suppose that $v_t \in S$ and $v_1 \notin S$. Then, there exists some i ($2 \leq i \leq t$) such that $\{v_j : i \leq j \leq t\} \subseteq S$, and $v_{i-1} \notin S$. But then the arc (v_{i-1}, v_i) contradicts the definition of S . $\diamond$

As a corollary to Lemma [7.2.1](#), we get the following.

Corollary 7.2.1. *Let $(\mathcal{M}_\varepsilon, k)$ be a Yes-instance of GSCS, and let S be a solution. Furthermore, let X denote a maximal strongly connected component in $\mathcal{M}_\varepsilon$.*

- *If $S \cap V(X) \neq \emptyset$, then $V(X) \subseteq S$*
- *If the vertex $v \in S$, then $R_{\mathcal{M}_\varepsilon}^-(v) \subseteq S$. Also, for every vertex $v \in V(\mathcal{M}_\varepsilon) \setminus S$, we have that $R_{\mathcal{M}_\varepsilon}^+(v) \cap S = \emptyset$.*

We also need the following hereditary property of the solution.

Lemma 7.2.2. *Let S be a solution of GSCS for $(\mathcal{M}_\varepsilon, k)$ and G be a subgraph of $\mathcal{M}_\varepsilon$. Then, $S' = S \cap V(G)$ is a solution of GSCS for $(G, |S'|)$.*

Proof. Towards the contradiction, suppose that S' is not a solution of GSCS for $(G, |S'|)$. Then, there exists vertices $w \in S'$ and $w' \in V(G) \setminus S'$ such that $(w', w) \in \mathcal{A}(G)$. Since $w' \in V(G) \setminus S'$, $w' \in V(\mathcal{M}_\varepsilon) \setminus S$. Furthermore, since G is a subgraph of $\mathcal{M}_\varepsilon$, $(w', w) \in \mathcal{A}(\mathcal{M}_\varepsilon)$, this contradicts the fact that S is a solution to $(\mathcal{M}_\varepsilon, k)$.
 $\diamond$

7.3 Hardness

In this section, we show that GSCS is W[1]-hard when parameterized by the solution size k . Towards this, we give a parameterized reduction from CLIQUE, a well-known W[1]-hard problem [36], to GSCS running in polynomial time. CLIQUE is formally defined as follows.

CLIQUE

Input: A graph G , and an integer k

Parameter: k

Task: Does there exist a set $Z \subseteq V(G)$ of size at least k such that $G[Z]$ is a clique?

Given an instance of CLIQUE, we create an instance of GSCS as follows.

Construction. Let (G, k) be an instance of CLIQUE. We construct the majority graph D in the following way (see Fig. 7.2).

1. For each vertex $v \in V(G)$, we introduce a vertex v and a directed cycle C_v passing through v of size k^2 in D . We call the vertex v as the *node vertex*, and vertices of C_v (including v) as the *indicator vertices* of v .
2. For each $e \in E(G)$, we introduce a vertex w_e . We will refer to these vertices as *edge vertices*.
3. For each edge $e \in E(G)$ with endpoints u and v , we introduce the arcs uw_e and vw_e in D .

We set the size of the solution to be $k' = k^3 + k(k-1)/2$. It is a well known fact that every directed graph is a majority graph of some election [43]. So D is a majority graph. This completes the description of the construction of an instance (D, k') of GSCS. Note that the steps of the reduction can be executed in polynomial time.

The intuitive idea of the reduction is the following. The construction enforces that when an edge vertex w_e is selected in a solution, both the endpoints of e are selected in the solution of GSCS. Moreover, when a node vertex v is selected, the directed cycle C_v is also selected in the solution. Intuitively, this indicates that v is in the solution to CLIQUE. The edge vertices in the solution of GSCS correspond to the edges that are in the solution of CLIQUE. We will show that due to size constraint of the solution there are exactly k^3 indicator vertices and $k(k-1)/2$ edge vertices.

Now, we formally prove the equivalence between the instance (G, k) of CLIQUE and (D, k') of GSCS.

Correctness. We start by observing following property of D .

Observation 7.3.1. $V(D) = \bigsqcup_{v \in V(G)} C_v \sqcup \bigsqcup_{e \in E(G)} w_e$.

Now for the correctness we show the following equivalence.

Lemma 7.3.1. (G, k) is a Yes-instance of CLIQUE if and only if $\mathcal{I} = (D, k')$ is a Yes-instance of GSCS.

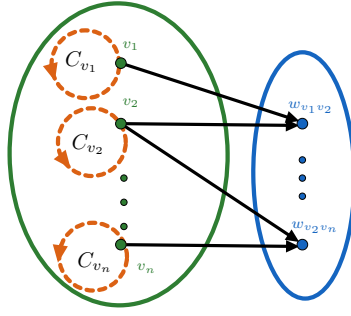


Figure 7.2: Construction of D . Here, $n = |V(G)|$, the **green** and **blue** vertices are the node vertices and edge vertices respectively, the vertices in the **green** set is the set of indicator vertices, and the **orange** dashed lines show the directed cycles of length k^2 in D .

Proof. For the forward direction, suppose that $Z = \{v_1, v_2, \dots, v_k\}$ is a solution to (G, k) . We construct a set $S \subseteq V(D)$ from Z as follows.

$$S = \bigsqcup_{i \in [k]} V(C_{v_i}) \uplus \bigsqcup_{\{i,j\} \subseteq [k]} w_{v_i v_j}.$$

Now, we prove that S is a solution to (D, k') . Note that $|S| = k^3 + k(k-1)/2$. Therefore, it is sufficient to prove that there is no arc (x, y) in D such that $x \in V(D) \setminus S$ and $y \in S$. Equivalently, we prove that for each $y \in S$, $N^-(y) \subseteq S$, by considering the type of the vertex y .

If y is an indicator vertex, i.e., $y \in V(C_{v_i})$, for some $i \in [k]$, then, by the construction of D , y has exactly one in-neighbor which is in the cycle C_{v_i} . Hence, $N^-(y) \subseteq V(C_{v_i})$, i.e., $N^-(y) \subseteq S$. Suppose that y is an edge vertex. Let $y = w_{v_i v_j}$. Then, the in-neighbors of y are v_i and v_j (see Fig. 7.2). Notice that v_i, v_j are in the clique. Hence, both of them are in S . Therefore, for both the cases, $N^-(y) \subseteq S$. This proves the forward direction.

For the reverse direction, let S be a solution to (D, k') . Let $\mathcal{I}' = S \cap \bigsqcup_{v \in V(G)} V(C_v)$, and $\mathcal{E}' = S \cap \bigsqcup_{e \in E(G)} w_e$. Due to Observation 7.3.1, $\mathcal{I}'$ and $\mathcal{E}'$ are mutually disjoint. That is, $S = \mathcal{I}' \uplus \mathcal{E}'$. Also, note that since for each vertex $v \in V(G)$, C_v is a strongly

connected component, by Corollary [7.2.1](#), if $V(C_v) \cap S \neq \emptyset$, then $V(C_v) \subseteq S$.

Now we define $V^* = \{v \in V(G) : V(C_v) \subseteq \mathcal{I}'\}$ and $E^* = \{e \in E(G) : w_e \in \mathcal{E}'\}$. We will prove that $G' = (V^*, E^*)$ is a solution to CLIQUE to (G, k) .

Claim 7.3.1. $|V^*| = k$.

Proof. Suppose $|V^*| = k^* < k$. For each $v \in V^*$, $V(C_v) \subseteq \mathcal{I}'$. Hence, the number of indicator vertices in S is k^*k^2 , which is less than k^3 . Since $k' = k^3 + k(k-1)/2$, there must be strictly more than $k(k-1)/2$ edge vertices in S . However, since each edge vertex has two node vertices as in-neighbors, and there are k^* node vertices in S , there are at most $\binom{k^*}{2}$ edge vertices in S . So the number of vertices in S is $k^*k^2 + \binom{k^*}{2}$. This contradicts that the size of S is at most $k' = k^3 + k(k-1)/2$.

Now, suppose $|V^*| = k^* > k$. Then, there are at least k^*k^2 vertices in S . Since $k^*k^2 \geq (k+1)k^2$, we have that $|S| > k'$, a contradiction. $\diamond$

Claim 7.3.2. $|E^*| = k(k-1)/2$.

Proof. From Claim [7.3.1](#), there are k node vertices in S . Therefore, S has k^3 many indicator vertices. From Observation [7.3.1](#), we know that the remaining vertices of S are from the set of edge vertices. Since $|S| = k^3 + k(k-1)/2$, there are $k(k-1)/2$ edge vertices. $\diamond$

Now, we prove that the vertices are consistent with the edges. That is, we prove that if the edge $uv \in E^*$, then the vertices $u, v \in V^*$. Note that if $w_{uv} \in S$, then since u, v are the in-neighbors of the edge vertex w_{uv} , we have that $u, v \in S$. Hence, $V(C_u) \subseteq S$ and $V(C_v) \subseteq S$. Therefore, if $uv \in E^*$, then $u, v \in V^*$. Moreover, since $G' = (V^*, E^*)$, $|E^*| = k(k-1)/2$, and $|V^*| = k$, we can infer that G' is a clique of size k . $\diamond$

Hence, we have proved the following theorem.

Theorem 18. *GSCS is W[1]-hard when parameterized by the size of solution.*

7.4 Exact Algorithms for GEHRLEIN STABLE COMMITTEE SELECTION

Let $(\mathcal{M}_{\mathcal{E}} = (\mathcal{C}, \mathcal{A}), k)$ be an instance of GSCS. Furthermore, let n denote the number of candidates or the number of vertices in $\mathcal{M}_{\mathcal{E}}$. Observe that we can design an algorithm for GSCS by enumerating all vertex subsets of size k and checking whether it forms a solution. This algorithm runs in time $\mathcal{O}^*\left(\binom{n}{k}\right) = \mathcal{O}^*(2^n)$. So a natural question is whether we can design an exact algorithm that improves over this brute-force enumeration algorithm.

In this section, we design a non-trivial exact algorithm for GSCS running in time $\mathcal{O}^*(1.2207^n)$. The main idea of the algorithm is inspired by Corollary [7.2.1](#). We find a subset of vertices with the property that either all of them go to the solution or none of them go to the solution. Once we have identified such a subset we recursively solve two subproblems, one where these vertices are part of a solution we are constructing, and the other where none of these vertices are part of the solution. Observe that a maximal strongly connected component provides a natural candidate of subset vertices. The algorithm indeed branches on strongly connected component of sufficiently large size and when we do not have a maximal strongly connected component of sufficiently big size we solve the problem in polynomial time. We first give the polynomial time subcase of our problem and then design the promised exact algorithm.

7.4.1 A polynomial time subcase

In this section, we give a polynomial time algorithm for GSCS when the majority graph, $\mathcal{M}_{\mathcal{E}}$, is a disjoint union of directed acyclic graphs and strongly connected components. That is, if we look at the connected components of underlying undirected graph of majority graph (that is, consider majority graph without the edge orientations), then they are either a directed acyclic graph or a strongly connected component in the majority graph. We denote such a family of graphs by $\mathcal{F}_{\text{dag}+\text{scc}}$. Furthermore, let $\mathcal{F}_{\text{scc}}$ denote the family of disjoint union of strongly connected components, and $\mathcal{F}_{\text{dag}}$ denote the family of disjoint union of directed acyclic graphs.

We first give algorithms for GSCS on $\mathcal{F}_{\text{scc}}$ and $\mathcal{F}_{\text{dag}}$, and then use these to give our algorithm on $\mathcal{F}_{\text{dag}+\text{scc}}$. We design an algorithm for GSCS on $\mathcal{F}_{\text{scc}}$, by reducing it to the well-known SUBSET SUM problem. In the SUBSET SUM problem, given a set of integers $X = \{x_1, \dots, x_n\}$, and an integer W , the goal is to find a set $X' \subseteq X$ such that $\sum_{x_i \in X'} x_i = W$.

Lemma 7.4.1. *GSCS on $\mathcal{F}_{\text{scc}}$ can be reduced to SUBSET SUM in $\mathcal{O}(n)$ time.*

Proof. Given an instance $(\mathcal{M}_{\mathcal{E}}, k)$ of GSCS such that $\mathcal{M}_{\mathcal{E}} \in \mathcal{F}_{\text{scc}}$, we construct an instance (X, W) of SUBSET SUM as follows. Let $\mathcal{M}_{\mathcal{E}} = \uplus_{i \in [p]} C_i$. Then, $X = \{x_i = |V(C_i)| : |V(C_i)| \leq k, i \in [p]\}$, and $W = k$. Note that the instance (X, W) can be constructed in $\mathcal{O}(n)$ time. Next, we show the equivalence between the instance $(\mathcal{M}_{\mathcal{E}}, k)$ of GSCS and (X, W) of SUBSET SUM. In the forward direction, suppose that S is a solution to $(\mathcal{M}_{\mathcal{E}}, k)$. By Corollary 7.2.1, we know that if $S \cap V(C_i) \neq \emptyset$, then $V(C_i) \subseteq S$. Let $X' = \{x_i \in X : V(C_i) \subseteq S\}$. Since $|S| = k$, it follows that $\sum_{x_i \in X'} x_i = k$. This completes the proof in the forward direction. In the backward direction, let $X' \subseteq X$ such that $\sum_{x_i \in X'} x_i = k$. Let $S = \{V(C_i) : x_i \in X'\}$. Note that $|S| = k$, and either $S \cap V(C_i) = \emptyset$, or $V(C_i) \subseteq S$, $i \in [p]$. Moreover, since $\mathcal{M}_{\mathcal{E}}$ belongs to $\mathcal{F}_{\text{scc}}$, it follows that S is a solution to GSCS. $\diamond$

Lemma 7.4.2. [98] *Given an instance (X, W) of SUBSET SUM, there exists an algorithm that solves SUBSET SUM in $\mathcal{O}(nW)$ time, where $n = |X|$.*

Using Lemmas [7.4.1] and [7.4.2], we get the following result.

Lemma 7.4.3. *GSCS can be solved in $\mathcal{O}(nk)$ time on $\mathcal{F}_{\text{scc}}$. Here, n is number of vertices in the input graph, and k is the size of solution.*

Now, we give a polynomial time algorithm for GSCS on $\mathcal{F}_{\text{dag}}$. The algorithm just selects the first k vertices of the topological ordering in the solution.

Lemma 7.4.4. *GSCS can be solved in $\mathcal{O}(n + m)$ time on $\mathcal{F}_{\text{dag}}$. Here, n is the number of vertices in the input graph, and m is the number of arcs in the graph.*

Proof. Given an instance $(\mathcal{M}_{\mathcal{E}}, k)$ of GSCS, where $\mathcal{M}_{\mathcal{E}} \in \mathcal{F}_{\text{dag}}$, we first find a topological ordering τ of $\mathcal{M}_{\mathcal{E}}$. Let $S = \{v \in V(\mathcal{M}_{\mathcal{E}}) : \tau(v) \leq k\}$. Note that $|S| = k$. We also note that if $u \in V(\mathcal{M}_{\mathcal{E}}) \setminus S$, then by the construction of S , $\tau(u) > \tau(v)$, for all $v \in S$. Since τ is a topological ordering, $(v, u) \notin \mathcal{A}(\mathcal{M}_{\mathcal{E}})$. Hence, S is a solution to $(\mathcal{M}_{\mathcal{E}}, k)$. $\diamond$

Now, we are ready to give a polynomial time algorithm for GSCS on $\mathcal{F}_{\text{dag+scc}}$. The algorithm first guesses how many vertices a solution contains from strongly connected components and directed acyclic graphs, respectively. Then, it runs the algorithms given in Lemmas [7.4.3] and [7.4.4] and compute the desired solution.

Theorem 19. *GSCS can be solved in $\mathcal{O}(nk^2 + m)$ time on $\mathcal{F}_{\text{dag+scc}}$. Here, n and m are the number of vertices and arcs in the input graph, respectively, and k is the size of solution.*

Proof. Let $(\mathcal{M}_{\mathcal{E}}, k)$ be an instance of GSCS where $\mathcal{M}_{\mathcal{E}} \in \mathcal{F}_{\text{dag+scc}}$. Furthermore, let $\mathcal{M}_{\mathcal{E}} = \mathcal{C} \uplus \mathcal{D}$, where $\mathcal{C}$ belongs to $\mathcal{F}_{\text{scc}}$, and $\mathcal{D}$ belongs to $\mathcal{F}_{\text{dag}}$. We first guess the number of vertices k_1 in the solution from $V(\mathcal{D})$. Clearly, $k - k_1$ vertices in

the solution are from $V(\mathcal{C})$. Let S be a solution for $(\mathcal{M}_\varepsilon, k)$ containing k_1 vertices from $V(\mathcal{D})$, and $k - k_1$ vertices from $V(\mathcal{C})$. Then using Lemma 7.2.2, $S \cap V(\mathcal{C})$ is a solution for $(\mathcal{C}, k - k_1)$, and $S \cap V(\mathcal{D})$ is a solution for $(\mathcal{D}, k_1)$. Therefore, using Lemmas 7.4.3 and 7.4.4, we find a solution S_1 for $(\mathcal{D}, k_1)$, and S_2 for $(\mathcal{C}, k - k_1)$. Now, $S = S_1 \uplus S_2$ is a solution for $(\mathcal{M}_\varepsilon, k)$ as $\mathcal{M}_\varepsilon$ is the disjoint union of $\mathcal{C}$ and $\mathcal{D}$. The running time of the algorithm follows from Lemmas 7.4.3 and 7.4.4, and the fact that there are at most k choices for k_1 as the solution size is k , and we only need to compute topological ordering once. $\diamond$

7.4.2 Exact exponential time algorithm

Now, we proceed towards presenting the exact exponential algorithm for GSCS. Towards this, we first prove the following structural result.

Lemma 7.4.5. *Let $(\mathcal{M}_\varepsilon, k)$ be an instance of GSCS such that $\mathcal{M}_\varepsilon \notin \mathcal{F}_{\mathbf{dag}+\mathbf{scc}}$. Then, $\mathcal{M}_\varepsilon$ has a strongly connected component X of size at least three such that either $|R^-(X)| \geq 4$ or $|R^+(X)| \geq 4$.*

Proof. If there does not exist a strongly connected component of $\mathcal{M}_\varepsilon$ of size at least three, then since $\mathcal{M}_\varepsilon$ does not contain parallel edges and self loops, $\mathcal{M}_\varepsilon$ is a **dag**, a contradiction to the fact that $\mathcal{M}_\varepsilon$ does not belong to $\mathcal{F}_{\mathbf{dag}+\mathbf{scc}}$. Thus, we know that there exists a strongly connected component of size at least 3.

Let X be a maximal strongly connected component of $\mathcal{M}_\varepsilon$ such that $|X|$ is maximized. If $|X| \geq 4$, we are done. Furthermore, observe that if there exists a maximal strongly connected component X , such that $|X| = 3$, and either $|R^-(X)| \geq 4$ or $|R^+(X)| \geq 4$, we are done. This implies that every maximal strongly connected component of size 3 is a connected component in itself in the underlying undirected graph of $\mathcal{M}_\varepsilon$. If we remove these components from $\mathcal{M}_\varepsilon$ we get a directed graph that does not have any strongly connected component and hence it is a **dag**. This

implies that $\mathcal{M}_\varepsilon$ belongs to $\mathcal{F}_{\text{dag}+\text{scc}}$, a contradiction. This concludes the proof. $\diamond$

Now, we are ready to present the algorithm. Let $(\mathcal{M}_\varepsilon, k)$ be an instance of GSCS. We apply the following branching rule exhaustively.

Branching Rule 7.4.1. *Suppose that X is a strongly connected component in $\mathcal{M}_\varepsilon$ such that $|X| \geq 3$ and either $|R^-(X)| \geq 4$ or $|R^+(X)| \geq 4$. Branch by adding $R^-(X)$ to the solution or deleting $R^+(X)$ from $\mathcal{M}_\varepsilon$. Recurse on the instances $(\mathcal{M}_\varepsilon - R^-(X), k - |R^-(X)|)$ and $(\mathcal{M}_\varepsilon - R^+(X), k)$, respectively.*

Lemma 7.4.6. *Branching Rule 7.4.1 is correct.*

Proof. We claim that $(\mathcal{M}_\varepsilon, k)$ is a Yes-instance of GSCS if and only if either $(\mathcal{M}_\varepsilon - R^-(X), k - |R^-(X)|)$ is a Yes-instance of GSCS or $(\mathcal{M}_\varepsilon - R^+(X), k)$ is a Yes-instance of GSCS. In the forward direction, let $(\mathcal{M}_\varepsilon, k)$ be a Yes-instance of GSCS and S be one of its solutions. Consider a strongly connected component, X , of $\mathcal{M}_\varepsilon$. If $x \in S \cap X$, then by Corollary 7.2.1, we have that $R^-(X) \subseteq S$. Using Lemma 7.2.2, $S \setminus R^-(X) = S \cap V(\mathcal{M}_\varepsilon - R^-(X))$ is a solution of GSCS for $(\mathcal{M}_\varepsilon - R^-(X), k - |R^-(X)|)$. Now, suppose that $X \cap S = \emptyset$. By Corollary 7.2.1, $R^+(X) \cap S = \emptyset$. Therefore, by Lemma 7.2.2, S is also a solution to $(\mathcal{M}_\varepsilon - R^+(X), k)$. This completes the proof in the forward direction.

In the backward direction, let S be a solution to GSCS for $(\mathcal{M}_\varepsilon - R^-(X), k - |R^-(X)|)$. We claim that $S' = S \cup R^-(X)$ is a solution to $(\mathcal{M}_\varepsilon, k)$. Suppose not, then there exists $u \in S'$ and $v \in V(\mathcal{M}_\varepsilon) \setminus S'$ such that $(v, u) \in \mathcal{A}(\mathcal{M}_\varepsilon)$. If $u \notin R^-(X)$, then (v, u) also belongs to $\mathcal{A}(\mathcal{M}_\varepsilon - R^-(X))$, a contradiction to the fact that S is a solution to $(\mathcal{M}_\varepsilon - R^-(X), k - |R^-(X)|)$. Now, suppose that $u \in R^-(X)$. Since $v \notin S'$, $v \notin R^-(X)$, a contradiction to the fact that $(v, u) \in \mathcal{A}(\mathcal{M}_\varepsilon)$. This proves that S' is a solution to $(\mathcal{M}_\varepsilon, k)$. Now suppose that S is a solution to GSCS for $(\mathcal{M}_\varepsilon - R^+(X), k)$. We claim that S is also a solution to $(\mathcal{M}_\varepsilon, k)$. Suppose not, then there exists $u \in S$ and $v \in V(\mathcal{M}_\varepsilon) \setminus S$ such that $(v, u) \in \mathcal{A}(\mathcal{M}_\varepsilon)$. If $v \notin R^+(X)$,

then (v, u) also belongs to $\mathcal{A}(\mathcal{M}_\mathcal{E} - R^+(X))$, a contradiction to that S is a solution to $(\mathcal{M}_\mathcal{E} - R^+(X), k)$. Now, suppose that $v \in R^+(X)$. Since $u \in S$, $u \notin R^+(X)$, a contradiction to that $(v, u) \in \mathcal{A}(\mathcal{M}_\mathcal{E})$. $\diamond$

Theorem 20. *GSCS can be solved in $\mathcal{O}^*(1.2207^n)$ time optimally, where n is the number of vertices in $\mathcal{M}_\mathcal{E}$.*

Proof. Given an instance $(\mathcal{M}_\mathcal{E}, k)$ of GSCS, we first check whether $\mathcal{M}_\mathcal{E}$ belongs to the family $\mathcal{F}_{\text{dag+scc}}$. If yes, then we can solve the problem in polynomial time using Theorem [19]. Otherwise, using Lemma [7.4.5] there exists a strongly connected component, X of size at least three such that either $|R^-(X)| \geq 4$ or $|R^+(X)| \geq 4$. Now, we apply Branching Rule [7.4.1]. The safeness of algorithm follows from the safeness of branching rule. The running time of the algorithm is governed by the recurrence, $T(n) \leq T(n-3) + T(n-4)$, which solves to $\mathcal{O}^*(1.2207^n)$. $\diamond$

7.5 FPT Algorithms for GSCS

Given an instance $(\mathcal{M}_\mathcal{E}, k)$ of GSCS, let q be the size of **tv**d of $\mathcal{M}_\mathcal{E}$ and $l = \binom{n}{2} - |\mathcal{A}(\mathcal{M}_\mathcal{E})|$. In this section, we design fixed parameter algorithms for GSCS with respect to parameters q and l .

We first give an FPT algorithm (Algorithm [7.5.1]) for GSCS when parameterized by q . First, we state a known result which is the starting point of our algorithm.

Proposition 7.5.1. [7] *GSCS can be solved in the polynomial time if the majority graph $\mathcal{M}_\mathcal{E}$ is a tournament. Moreover, such a solution is unique, if exists.*

Let X be a *tv*d of $\mathcal{M}_\mathcal{E}$. Note that X is a vertex cover - a set of vertices such that each edge is incident to at least one vertex of the set - of the complement graph of the underlying undirected graph of $\mathcal{M}_\mathcal{E}$. Hence, it can be computed in $\mathcal{O}^*(1.2738^q)$

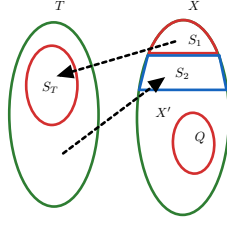


Figure 7.3: An illustration of Algorithm 7.5.1 where vertices in the **red** sets are in the solution

time, where $q = |X|$, using the FPT algorithm proposed by Chen et al. [24]. Note that $T = \mathcal{M}_\mathcal{E} - X$ is a tournament. Proposition 7.5.1 says that every tournament has a unique solution. Thus, for our algorithm we first guess how many vertices from T are present in our potential solution to $(\mathcal{M}_\mathcal{E}, k)$. Once, we have guessed this number k_1 , we run the algorithm mentioned in Proposition 7.5.1 and find the unique solution of size k_1 , say S_T , if exists. Having found the set S_T , we know that no vertex of $T - S_T$ goes into the solution. Hence, we now apply Corollary 7.2.1 and reduce the problem to a directed graph induced on a subset of X . At this stage we run the exact exponential time algorithm described in Theorem 20 and we are done. See Fig. 7.3 for illustration of the algorithm. A detailed description of the algorithm is given in Algorithm 7.5.1.

Algorithm 7.5.1: FPT for GSCS

Input: A majority graph $\mathcal{M}_\mathcal{E}$, a **tvd**, X of $\mathcal{M}_\mathcal{E}$, and an integer k .

Output: S , which is a solution of GSCS for $(\mathcal{M}_\mathcal{E}, k)$, if non-empty

```

1  $S_1 = \emptyset, Q = \emptyset;$ 
2 for each  $k_1 \in [k]$  do
3   if  $T$  has a solution of GSCS of size  $k_1$  then
4     let  $S_T$  be the solution of  $(T, k_1)$  computed using Proposition 7.5.1;
      $S_1 = R_{\mathcal{M}_\mathcal{E}}^-(S_T) \cap X, S_2 = R_{\mathcal{M}_\mathcal{E}}^+(V(T) \setminus S_T) \cap X;$ 
5     if  $S_1 \cap S_2 = \emptyset$  then
6        $X' = X \setminus (S_1 \cup S_2), k_2 = k - |S_T \uplus S_1|;$ 
7       if  $\mathcal{M}_\mathcal{E}[X']$  has a solution of GSCS of size  $k_2$  then
8         let  $Q$  be a solution of  $(\mathcal{M}_\mathcal{E}[X'], k_2)$  computed using Theorem
          20;
9          $S = S_T \uplus S_1 \uplus Q;$ 
10        return  $S$ .
11 return  $S = \emptyset$ 
```

Now, we prove the correctness of this algorithm.

Lemma 7.5.1. *Algorithm 7.5.1 is correct.*

Proof. To prove the correctness of algorithm, we prove that if Algorithm 7.5.1 returns a non-empty set S , then it is a solution of GSCS for $(\mathcal{M}_\mathcal{E}, k)$, otherwise $(\mathcal{M}_\mathcal{E}, k)$ does not have a solution of GSCS.

Case A: Suppose that $S \neq \emptyset$. We claim that S is a solution to $(\mathcal{M}_\mathcal{E}, k)$. Suppose not, then either $|S| \neq k$ or there exists $w \in S$, and $w' \in V(\mathcal{M}_\mathcal{E}) \setminus S$ such that $(w', w) \in \mathcal{A}(\mathcal{M}_\mathcal{E})$. By the construction of S , if $S \neq \emptyset$, then $|S| = k$. Now, suppose that there exists $w \in S$ and $w' \in V(\mathcal{M}_\mathcal{E}) \setminus S$ such that $(w', w) \in \mathcal{A}(\mathcal{M}_\mathcal{E})$. Following four cases are possible.

Case 1: Suppose that $w, w' \in V(T)$. Since $S \cap V(T) = S_T$, $w \in S_T$ and $w' \notin S_T$. Since T is an induced subgraph of $\mathcal{M}_\mathcal{E}$, $(w', w) \in \mathcal{A}(T)$, this contradicts the fact that S_T is a solution to $(T, |S_T|)$.

Case 2: Suppose that $w, w' \in X$. Note that by the construction of X' , $X \setminus X' = S_1 \uplus S_2$. Note that $S_1 \subseteq S$ and $S_2 \cap S = \emptyset$. Following four cases are possible.

Case(i): Suppose that $w, w' \in X \setminus X'$. Since $w \in S$, it follows that $w \in S_1$. Since $w' \notin S$, $w' \in S_2$. Since $(w', w) \in \mathcal{A}(\mathcal{M}_\mathcal{E})$, and w is in the *in-reachability* set of S_T in $\mathcal{M}_\mathcal{E}$, w' also belongs to *in-reachability* set of S_T in $\mathcal{M}_\mathcal{E}$. Hence, $w' \in S_1$, a contradiction to the fact that S_1 and S_2 are disjoint.

Case(ii): Suppose that $w \in X \setminus X'$ and $w' \in X'$. Since $w \in S$, $w \in S_1$. As argued above, since $(w', w) \in \mathcal{A}(\mathcal{M}_\mathcal{E})$, w' also belongs to S_1 , a contradiction to that X' and S_1 are disjoint.

Case(iii): Suppose that $w \in X'$ and $w' \in X \setminus X'$. Since $w' \notin S$, we have that $w' \in S_2$. Since $(w', w) \in \mathcal{A}(\mathcal{M}_\mathcal{E})$, and w' is in *out-reachability*

set of $V(T) \setminus S_T$ in $\mathcal{M}_\mathcal{E}$, w also belongs to *out-reachability* set of $V(T) \setminus S_T$ in $\mathcal{M}_\mathcal{E}$ and hence $w \in S_2$, a contradiction to that X' and S_2 are disjoint.

Case(iv): Suppose that $w, w' \in X'$. Since $Q \subseteq S$, $w \in Q$ and $w' \notin Q$. Since $\mathcal{M}_\mathcal{E}[X']$ is an induced subgraph of $\mathcal{M}_\mathcal{E}$, we have that $(w', w) \in \mathcal{A}(\mathcal{M}_\mathcal{E}[X'])$, this contradicts that Q is a solution to $(\mathcal{M}_\mathcal{E}[X'], |Q|)$.

Case 3: Suppose that $w \in V(T)$ and $w' \in X$. Since $w \in S$, $w \in S_T$. Since $w' \notin S$, there are two cases, either $w' \in S_2$ or $w' \in X' \setminus Q$. Since $(w', w) \in \mathcal{A}(\mathcal{M}_\mathcal{E})$ and $w \in S_T$, w' belongs to the *in-reachability* set of S_T in $\mathcal{M}_\mathcal{E}$ and hence $w' \in S_1$. If $w' \in S_2$, then it contradicts that S_1 and S_2 are disjoint. If $w' \in X'$, then it contradicts that S_1 and X' are disjoint.

Case 4: Suppose $w \in X$ and $w' \in V(T)$. Since $w \in S$, w either belongs to S_1 or Q . Since $(w', w) \in \mathcal{A}(\mathcal{M}_\mathcal{E})$ and $w' \in V(T) \setminus S$, clearly, w belongs to the *out-reachability* set of $V(T) \setminus S$ in $\mathcal{M}_\mathcal{E}$. Therefore, $w \in S_2$. If $w \in S_1$, then it contradicts that S_1 and S_2 are disjoint. If $w \in Q$, then it contradicts that Q and S_2 are disjoint.

Case B: Suppose that $S = \emptyset$. We claim that $(\mathcal{M}_\mathcal{E}, k)$ does not have a solution of GSCS. Towards the contrary, let Z be a solution to $(\mathcal{M}_\mathcal{E}, k)$. Let $Z_T = V(T) \cap Z$ and $k' = |Z_T|$. Using Lemma 7.2.2, Z_T is a solution to (T, k') . Since T is a tournament, by uniqueness of solution of tournament (Proposition 7.5.1), $S_T = Z_T$, where S_T is a set returned in Step 4 of Algorithm 7.5.1 when $k_1 = k'$. Let $Z_1 = R_{\mathcal{M}_\mathcal{E}}^-(Z_T) \cap X$ and $Z_2 = R_{\mathcal{M}_\mathcal{E}}^+(V(T) \setminus Z_T) \cap X$. Using Corollary 7.2.1, $Z_1 \subseteq Z$ and $Z_2 \cap Z = \emptyset$. Therefore, Z_1 and Z_2 are disjoint. Clearly, $S_1 = Z_1$ and $S_2 = Z_2$ in Step 4 of Algorithm 7.5.1. Let $X' = X \setminus (Z_1 \cup Z_2)$. Note that $X = X' \uplus Z_1 \uplus Z_2$. Since $Z_1 \subseteq Z$ and $Z_2 \cap Z = \emptyset$, $Z \cap X = Z_1 \uplus (Z \cap X')$. Using Lemma 7.2.2, $Z' = Z \cap X'$ is a solution to $(\mathcal{M}_\mathcal{E}[X'], |Z'|)$. Since there exist a solution to $(\mathcal{M}_\mathcal{E}[X'], |Z'|)$, algorithm finds a solution Q

to $(\mathcal{M}_\mathcal{E}[X'], |Z'|)$ in Step 8. Since $Z = Z_T \uplus (Z \cap X)$ and $Z \cap X = Z_1 \uplus Z'$, $|Z'| = k - |Z_T \uplus Z_1| = k - |S_T \uplus S_1|$. Therefore, Algorithm 7.5.1 returns $S = S_T \uplus S_1 \uplus Q$, a contradiction to that $S = \emptyset$.

◇

Lemma 7.5.2. *The running time of Algorithm 7.5.1 is $\mathcal{O}^*(1.2207^q)$, where q is the size of **tv**d of majority graph $\mathcal{M}_\mathcal{E}$.*

Proof. In the algorithm, set S_T (Step 4) can be computed in polynomial time using Proposition 7.5.1 and set Q (Step 8) can be obtained using Theorem 20 in $\mathcal{O}^*(1.2207^q)$ time. Hence, the running time of the algorithm is $\mathcal{O}^*(1.2207^q)$.

◇

Theorem 21. *GSCS can be solved in $\mathcal{O}^*(1.2738^q)$ time, where q is the size of **tv**d of majority graph $\mathcal{M}_\mathcal{E}$.*

Proof. Given an instance $(\mathcal{M}_\mathcal{E}, k)$ of GSCS, we first compute vertex cover, X , of the complement graph of underlying undirected graph of $\mathcal{M}_\mathcal{E}$ using an FPT algorithm which runs in $\mathcal{O}^*(1.2738^q)$ time, where q is the size of vertex cover [24]. Now using Algorithm 7.5.1, we compute a solution S of GSCS to $(\mathcal{M}_\mathcal{E}, k)$, if exists. The correctness of algorithm follows from Lemma 7.5.1. Since using Lemma 7.5.2, the running time of Algorithm 7.5.1 is $\mathcal{O}^*(1.2207^q)$, and we compute X in $\mathcal{O}^*(1.2738^q)$ time, it follows that GSCS can be solved in $\mathcal{O}^*(1.2738^q)$ time.

◇

Now, we give an FPT algorithm for GSCS when the number of pairs of candidates which are tied among each other in the pairwise majority contest is bounded.

Theorem 22. *GSCS can be solved in $\mathcal{O}^*(1.2207^l)$ time when the number of missing arcs in majority graph is l .*

Proof. Given an instance $(\mathcal{M}_\mathcal{E}, k)$ of GSCS, let X be a set of vertices obtained by adding a vertex from each of the missing arcs. Note that X is a **tv**d of $\mathcal{M}_\mathcal{E}$, and

$|X| \leq l$. Now, using Algorithm [7.5.1](#), we compute a solution S of GSCS for $(\mathcal{M}_{\mathcal{E}}, k)$, if exists. The correctness of algorithm follows from Lemma [7.5.1](#). Since $|X| \leq l$, using Lemma [7.5.2](#), we can solve GSCS in $\mathcal{O}^*(1.2207^l)$ time. $\diamond$

7.6 A linear vertex kernel for GSCS

In this section, we show that GSCS admits a kernel with $\mathcal{O}(q)$ vertices, where q is the size of **tv**d of $\mathcal{M}_{\mathcal{E}}$. That is, we give a polynomial time algorithm that given an instance $(\mathcal{M}_{\mathcal{E}}, k)$ of GSCS returns an instance $(\mathcal{M}_{\mathcal{E}'}, k')$ of GSCS such that $(\mathcal{M}_{\mathcal{E}}, k)$ is a Yes-instance of GSCS if and only if $(\mathcal{M}_{\mathcal{E}'}, k')$ is a Yes-instance of GSCS and $|V(\mathcal{M}_{\mathcal{E}'})| \leq 4q + 1$.

Let $(\mathcal{M}_{\mathcal{E}}, k)$ be an instance of GSCS. Let X be a set such that $T = \mathcal{M}_{\mathcal{E}} - X$ is a tournament. Let $t = |V(T)|$. We know that every tournament T has a Hamiltonian path—a path that visits every vertex exactly once. Furthermore, a Hamiltonian path in tournament can be computed in polynomial time [\[73\]](#). Let $\mathcal{H} = (v_1, v_2, \dots, v_t)$ be one such path. Now notice that no vertex in $\{v_{k+1}, \dots, v_t\}$ belongs to any solution to $(\mathcal{M}_{\mathcal{E}}, k)$ and thus, we should be able to find a reduction rule that can reduce the size of T to $k + 1$. However, this is still not the desired kernel. Next, we change our perspective and see which vertices from T must be in a solution of size k . Once we detect such a vertex, we can use Corollary [7.2.1](#) to find a desired reduction rule.

Before diving into the details of the algorithm, we give an alternate polynomial time algorithm to find a solution of GSCS, when the majority graph is a tournament. This will be crucially used in designing the kernelization algorithm.

Lemma 7.6.1. *Let (G, k) be an instance of GSCS, where G is a tournament. Let $|V(G)| = t$, and $\mathcal{H} = (v_1, v_2, \dots, v_t)$ be a Hamiltonian path in G . Furthermore, let $S = \{v_1, v_2, \dots, v_k\}$. If $|R_G^-(S)| = k$, then S is a solution of GSCS to (G, k) . Moreover, it is the unique solution and can be computed in polynomial time.*

Proof. Since $|R_G^-(S)| = k$, for every $v \in S$, $N_G^-(v) \subseteq S$. Hence, S is a solution of GSCS to (G, k) . Next, we will prove that it is the unique solution. Suppose not, then let $S' (\neq S)$ be a solution of GSCS for (G, k) . Since $|S'| = |S|$, there exists a vertex $v^* \in S' \setminus S$, i.e., $v^* \in \{v_{k+1}, \dots, v_t\}$. Note that $P = (v_1, v_2, \dots, v^*)$ is a subpath of $\mathcal{H}$. Each vertex in P can reach v^* via the path P . Hence, $V(P) \subseteq R_G^-(v^*)$. Also, since $v^* \notin \{v_1, v_2, \dots, v_k\}$, the number of vertices in P is at least $k + 1$. Hence, $|R_G^-(v^*)| \geq k + 1$. Since $v^* \in S'$, using Corollary 1, $R_G^-(v^*) \subseteq S'$, this contradicts that S' is a solution to (G, k) . Since the Hamiltonian path in a tournament can be found in polynomial time [90], S can be computed in polynomial time. $\diamond$

Now, we are ready to give the detailed description of the algorithm. First, we describe how to find a **tvd**, X , of size at most $2q$. Recall that every **tvd** of $\mathcal{M}_\mathcal{E}$ is also a vertex cover of the complement graph, $\overline{G}$, of the underlying undirected graph of $\mathcal{M}_\mathcal{E}$. Thus, to get the desired X , we find a vertex cover of $\overline{G}$ using a well-known factor 2-approximation algorithm for the VERTEX COVER problem [11]. Note that $T = \mathcal{M}_\mathcal{E} - X$ is a tournament. Next, we define a sequence of reduction rules. At any point of time we apply the lowest indexed reduction rule that is applicable. That is, a rule is applied only when none of the preceding rules can be applied. After an application of any rule, we reuse the notation $\mathcal{M}_\mathcal{E}$ to denote the reduced majority graph.

Reduction Rule 7.6.1. *Let $(\mathcal{M}_\mathcal{E}, k)$ be an instance of GSCS and let $T = \mathcal{M}_\mathcal{E} - X$. Furthermore, let $\mathcal{H} = (v_1, v_2, \dots, v_t)$ be a Hamiltonian path in T . If $t > k + 1$, then construct the majority graph $\mathcal{M}_{\mathcal{E}'}$ as follows. Initially, $\mathcal{M}_{\mathcal{E}'} = \mathcal{M}_\mathcal{E} - \{v_t\}$. Then, update the out-neighborhood of v_{t-1} as $N_{\mathcal{M}_{\mathcal{E}'}}^+(v_{t-1}) = N_{\mathcal{M}_\mathcal{E}}^+(v_{t-1}) \cup N_{\mathcal{M}_\mathcal{E}}^+(v_t) \setminus \{v_{t-1}\}$. The resulting instance is $(\mathcal{M}_{\mathcal{E}'}, k)$.*

Lemma 7.6.2. *Reduction Rule [7.6.1] is safe.*

Proof. Suppose that $(\mathcal{M}_\mathcal{E}, k)$ is a Yes-instance of GSCS and S is a solution to

$(\mathcal{M}_\varepsilon, k)$. We prove that S is also a solution to $(\mathcal{M}_{\varepsilon'}, k)$. To prove this, first, we claim that $v_t \notin S$. By Lemma 7.2.2, $S \cap V(T)$ is a solution to $(T, |S \cap V(T)|)$ and by Lemma 7.6.1, we know that it is the unique solution to $(T, |S \cap V(T)|)$. Since $t > k + 1$, by Lemma 7.6.1, we have that $v_t \notin S \cap V(T)$. Since $v_t \in V(T)$, it follows that $v_t \notin S$. Suppose S is not a solution to $(\mathcal{M}_{\varepsilon'}, k)$, then there exists $x \in S$, and $y \in V(\mathcal{M}_{\varepsilon'}) \setminus S$ such that $(y, x) \in \mathcal{A}(\mathcal{M}_{\varepsilon'})$. If $y \neq v_{t-1}$, then (y, x) also belongs to $\mathcal{A}(\mathcal{M}_\varepsilon)$, a contradiction to the fact that S is a solution to $(\mathcal{M}_\varepsilon, k)$. Suppose $y = v_{t-1}$, then by the construction of $\mathcal{M}_{\varepsilon'}$, either $(v_{t-1}, x) \in \mathcal{A}(\mathcal{M}_\varepsilon)$ or $(v_t, x) \in \mathcal{A}(\mathcal{M}_\varepsilon)$. If $(v_{t-1}, x) \in \mathcal{A}(\mathcal{M}_\varepsilon)$, then it contradicts the fact that S is a solution to $(\mathcal{M}_\varepsilon, k)$. Suppose $(v_t, x) \in \mathcal{A}(\mathcal{M}_\varepsilon)$. Since we have already shown that $v_t \notin S$, we get a contradiction to the fact that S is a solution to $(\mathcal{M}_\varepsilon, k)$.

For the other direction, suppose S' is a solution to $(\mathcal{M}_{\varepsilon'}, k)$. We prove that S' is also a solution to $(\mathcal{M}_\varepsilon, k)$. Suppose not, then there exists $x \in S'$, and $y \in V(\mathcal{M}_\varepsilon) \setminus S'$ such that $(y, x) \in \mathcal{A}(\mathcal{M}_\varepsilon)$. If $y \neq v_t$, then (y, x) also belongs to $\mathcal{A}(\mathcal{M}_{\varepsilon'})$, a contradiction to that S' is a solution to $(\mathcal{M}_{\varepsilon'}, k)$. Suppose $y = v_t$. Since $(v_1, \dots, v_{t-1})$ is a path in T , then $(v_{t-1}, v_t) \in \mathcal{A}(\mathcal{M}_\varepsilon)$. Therefore, $x \neq v_{t-1}$. Since $y = v_t$, and $(y, x) \in \mathcal{A}(\mathcal{M}_{\varepsilon'})$, by construction of $\mathcal{M}_{\varepsilon'}$, we have that $(v_{t-1}, x) \in \mathcal{A}(\mathcal{M}_{\varepsilon'})$. Now, since $v_{t-1} \notin S'$, it contradicts that S' is a solution to $(\mathcal{M}_{\varepsilon'}, k)$. $\diamond$

If Reduction Rule 7.6.1 is not applicable, then $|V(T)| \leq k + 1$. Since $|X| \leq 2q$, we have that $\mathcal{M}_\varepsilon$ has at most $2q + k + 1$ vertices. Hence, the next lemma follows.

Lemma 7.6.3. *If $k \leq 2q$, and Reduction Rule 7.6.1 is not applicable, then $|V(\mathcal{M}_\varepsilon)| \leq 4q + 1$.*

Now, it remains to bound the number of vertices in T by $\mathcal{O}(q)$ when $k > 2q$.

Reduction Rule 7.6.2. *Let $(\mathcal{M}_\varepsilon, k)$ be an instance of GSCS and let $T = \mathcal{M}_\varepsilon - X$. Furthermore, let $\mathcal{H} = (v_1, v_2, \dots, v_{|V(T)|})$ be a Hamiltonian path in T . If $k > 2q$ and $|R_T^-(\{v_1, \dots, v_{k-2q}\})| > k$, then output a No instance of constant size.*

Lemma 7.6.4. *Reduction Rule 7.6.2 is safe.*

Proof. Suppose that $k > 2q$, and $|R^-(\{v_1, \dots, v_{k-2q}\})| > k$. We prove that $(\mathcal{M}_\mathcal{E}, k)$ is a no instance of GSCS. Suppose not, let S be a solution of GSCS for $(\mathcal{M}_\mathcal{E}, k)$. Since $k > 2q$, and $|X| \leq 2q$, we have that any k size solution for $\mathcal{M}_\mathcal{E}$ must contain vertices outside X . That is, it must contain at least $k - 2q$ vertices of T . Note that using Lemma 7.2.2, $S \cap V(T)$ is a solution for $(T, |S \cap V(T)|)$ and using Lemma 7.6.1, it is the unique solution for $(T, |S \cap V(T)|)$. Since $|S \cap V(T)| \geq k - 2q$, using Lemma 7.6.1, $\{v_1, \dots, v_{k-2q}\} \subseteq S \cap V(T)$. Hence $\{v_1, \dots, v_{k-2q}\} \subseteq S$. Since $R_T^-(\{v_1, \dots, v_{k-2q}\}) > k$, $|S| > k$, a contradiction to that S is a solution to $(\mathcal{M}_\mathcal{E}, k)$. $\diamond$

Reduction Rule 7.6.3. *Let $(\mathcal{M}_\mathcal{E}, k)$ be an instance of GSCS and let $T = \mathcal{M}_\mathcal{E} - X$. Furthermore, let $\mathcal{H} = (v_1, v_2, \dots, v_t)$ be a Hamiltonian path in T . If $k > 2q$, then delete $R_{\mathcal{M}_\mathcal{E}}^-(v_1)$ from $\mathcal{M}_\mathcal{E}$. The reduced instance is $(\mathcal{M}_{\mathcal{E}'}, k')$, where $\mathcal{M}_{\mathcal{E}'} = \mathcal{M}_\mathcal{E} - R_{\mathcal{M}_\mathcal{E}}^-(v_1)$ and $k' = k - |R_{\mathcal{M}_\mathcal{E}}^-(v_1)|$.*

Lemma 7.6.5. *Reduction Rule 7.6.3 is safe.*

Proof. Suppose that $(\mathcal{M}_\mathcal{E}, k)$ is a Yes instance of GSCS and S is one of its solutions. Since $k > 2q$, and $|X| \leq 2q$, S must contain vertices of T . Therefore, by Lemma 7.6.1, the first vertex in $\mathcal{H}$ must be in S . Hence, by Corollary 7.2.1, $R_{\mathcal{M}_\mathcal{E}}^-(v_1) \subseteq S$. Therefore, by Lemma 7.2.2, $S \cap V(\mathcal{M}_{\mathcal{E}'}) = S \setminus R_{\mathcal{M}_\mathcal{E}}^-(v_1)$ is a solution to $(\mathcal{M}_{\mathcal{E}'}, k')$.

For the reverse direction, suppose that S' is a solution to $(\mathcal{M}_{\mathcal{E}'}, k')$. We claim that $S = S' \cup R_{\mathcal{M}_\mathcal{E}}^-(v_1)$ is solution to $(\mathcal{M}_\mathcal{E}, k)$. Suppose not, then there exists $x \in S$ and $y \in V(\mathcal{M}_\mathcal{E}) \setminus S$ such that $(y, x) \in \mathcal{A}(\mathcal{M}_\mathcal{E})$. Since $R_{\mathcal{M}_\mathcal{E}}^-(v_1) \subseteq S$, and $y \notin S$, it follows that $y \notin R_{\mathcal{M}_\mathcal{E}}^-(v_1)$. This implies that $x \notin R_{\mathcal{M}_\mathcal{E}}^-(v_1)$. Hence, $x \in S'$, and $y \in V(\mathcal{M}_{\mathcal{E}'})$. Since $\mathcal{M}_{\mathcal{E}'}$ is an induced subgraph of $\mathcal{M}_\mathcal{E}$, $(y, x) \in \mathcal{A}(\mathcal{M}_{\mathcal{E}'})$, a contradiction to the fact that S' is a solution to $(\mathcal{M}_{\mathcal{E}'}, k')$. $\diamond$

Now, we give the main result of this section.

Theorem 23. *GSCS admits a kernel with $4q + 1$ vertices.*

Proof. Consider an instance $(\mathcal{M}_\mathcal{E}, k)$ of GSCS. Let $\overline{G}$ denote the complement graph of underlying undirected graph of $\mathcal{M}_\mathcal{E}$. We first find a vertex cover, X , of $\overline{G}$ of size at most $2q$ using factor 2-approximation algorithm for VERTEX COVER [11]. Note that X is a **tvd** of $\mathcal{M}_\mathcal{E}$. Therefore, $T = \mathcal{M}_\mathcal{E} - X$ is a tournament. Suppose that Reduction Rule [7.6.1] is not applicable, then $|V(T)| \leq k + 1$. If $k \leq 2q$, then using Lemma [7.6.3], $|V(\mathcal{M}_\mathcal{E})| \leq 4q + 1$. Now, suppose that $k > 2q$. Then, either Reduction Rule [7.6.2] or [7.6.3] is applicable. After exhaustive application of Reduction Rules [7.6.2] and [7.6.3], either we return a no-instance of constant size or $k \leq 2q$. As argued above if $k \leq 2q$, we have that $|V(\mathcal{M}_\mathcal{E})| \leq 4q + 1$. Note that each of the reduction rules can be applied in polynomial time, and each of them either declare that the given instance is a No instance or reduces the size of the graph. Therefore, the overall running time is polynomial in the input size. $\diamond$

Using Theorem [23], we get the following result.

Corollary 7.6.1. *GSCS admits a kernel with $2l + 1$ vertices, where l is the number of missing arcs in the majority graph.*

Proof. Let X be a set of vertices obtained by adding a vertex from each of the missing arcs. Note that X is a **tvd** of $\mathcal{M}_\mathcal{E}$, and $|X| \leq l$. Note that in the proof of Theorem [23], by the exhaustive application of Reduction Rules [7.6.1] to [7.6.3], we bound the number of vertices in $\mathcal{M}_\mathcal{E}$ by $2|X| + 1$. Hence, we obtain the desired kernel. $\diamond$

7.7 Conclusion

In this chapter we studied GEHRLEIN STABLE COMMITTEE SELECTION problem in the realm of parameterized complexity. We put forward a parameterized complexity map of the problem, by way of W -hardness, fixed-parameterized tractability, and kernelization. We showed that the problem is $W[1]$ -hard when parameterized by the size of the committee, yet it admits fixed parameter tractable algorithms and linear kernels with respect to alternate structural parameters which encode the “closeness” of the underlying majority graph of the given election to a tournament. Another natural direction is to study the problem with respect to other relevant parameters.

Manipulation Problems

Chapter 8

Stable Extension of Partial Matching

In the previous parts we gave algorithms for finding stable assignment or stable selection. In this part we give algorithms to manipulate an input instance so that our desired outcome is stable.

We first consider the STABLE MATCHING problem and give algorithms to manipulate an instance of STABLE MATCHING so that a given matching is produced as an output of the Gale-Shapley algorithm in the modified instance. We give algorithms for both the case of complete and incomplete preference lists. We first develop a structure to study the problem in Section 8.3. In Section 8.4 we give an algorithm for complete preference lists and also prove matching lower bounds for this algorithm. In Section 8.5 we extend our algorithm for incomplete lists.

STABLE MATCHING together with its numerous variants are among the most well-studied problems in matching theory, driven by applications to economics, business, and more recently to medical sciences such as organ donation and exchange, [2]. Roth and Sotomayor in [139] give a detailed exposition on the applications of stable matching in economics and business. The model of the two-sided STABLE

MATCHING problem that we study in this chapter is slightly more restricted than the ones we studied in Chapters 3 and 4. Here, we are given two sets of agents of equal size. Recall that the sets are referred to as *men* and *women*, where each person submits a ranked list of all the members of the opposite sex.

Ever since the theoretical framework for STABLE MATCHING was laid down by Gale and Shapley [58] to study the then current heuristic used to assign medical residents to hospitals in New England, the topic has received considerable attention from theoreticians and practitioners alike. In particular, it is one of the foundational problems in social choice theory, where a matching is viewed as an *allocation* or *assignment* of resources to relevant agents, whereby the nature of the assignment can vary greatly depending on the scenario/marketplace they are modeling. We refer the reader to books [71, 139, 114] for an in-depth introduction to stable matching and its variants.

As stated in earlier chapters, Gale and Shapley [58] showed that every instance of the STABLE MATCHING problem admits a stable matching. In other words, given any set of preference lists of men and women there exists at least one stable matching. In fact, they gave a polynomial time algorithm to find a stable matching. This algorithm is widely used in both practice and theory, and it exists in two versions: the *men-proposing* and the *women-proposing*. The men (respectively, women)-proposing algorithm produces a *men-optimal* (respectively, *women-optimal*) stable matching, so named to emphasize the fact that one side prefers one matching over the other. Both variants are defined analogously. As the name suggests, the men-optimal stable matching is a stable matching that is no worse than any other stable matching, in terms of the preferences of the men. In other words, no man can get a better partner in any other stable matching when compared to the men-optimal stable matching. The men-proposing version of the algorithm works as follows. A man who is not yet matched to a woman, *proposes* to the woman who is at the top of his

current list, which is obtained by removing from his original preference list, all the women who have rejected him at an earlier step. On the woman's side, when a woman w receives a proposal from a man m , she accepts the proposal if it is her first proposal, or if she prefers m to her current partner. If w prefers her current partner to m , then w *rejects* m . If m is rejected by w , then m removes w from his list. This process continues until there is no unmatched man. The output of this algorithm is the men-optimal stable matching. For more details, see [71]. It has been customary to use the men-proposing version of the algorithm, and our analysis here will stick to that convention. Henceforth, unless explicitly stated otherwise, any mention of a stable matching should be interpreted by the reader as such. We will use $(\mathcal{L}^M, \mathcal{L}^W)$ to denote the set of preference lists of men and women, and the men-optimal matching with respect to these lists is denoted by $\text{GS}(\mathcal{L}^M, \mathcal{L}^W)$.

8.1 Our problem and motivation

Kobayashi and Matsui [101, 100] studied manipulation in the stable matching model, where agents are manipulating with the goal of attaining a specific matching target. Formally speaking, they considered the following class of problems. An input consists of two sets M and W of *men* and *women*, respectively, each of size n ; along with the preference list of every man (expressed as a strict ordering on the set of women), denoted by $\mathcal{L}^M$, and a matching on (M, W) . The said matching can either be *perfect* (if it contains n pairs), or *partial* (possibly, fewer than n pairs). Furthermore, for a couple of problems, we are given a set of preference lists of women, $\mathcal{L}^{W'}$, where $W' \subseteq W$. The goal is to decide if there exists a set of preference lists of women, $\mathcal{L}^W$, containing $\mathcal{L}^{W'}$, such that the men-optimal stable matching algorithm when used in conjunction with $\mathcal{L}^M$ yields a matching that contains all the pairs in the stated matching. Of these problems, two are directly related to our work in this chapter. Let us consider the following two problems, and compare and contrast their

computational complexity.

ATTAINABLE STABLE MATCHING (ASM)

- Input:* A set of preference lists $\mathcal{L}^M$ of men over women W , and a perfect matching μ on (M, W) .
- Task:* Does there exist a set of preference lists of women $\mathcal{L}^W$, such that $\text{GS}(\mathcal{L}^M, \mathcal{L}^W) = \mu$?

Kobayashi and Matsui in [101, 100] showed that ASM is polynomial time solvable, and exhibited an $\mathcal{O}(n^2)$ algorithm that computes the set $\mathcal{L}^W$, if it exists. Or else, reports “none exists”. The following problem is identical to the above, except in one key aspect: *the target matching need not be perfect*. The authors show that this problem is NP-complete.

STABLE EXTENSION OF PARTIAL MATCHING (SEOPM)

- Input:* A set of preference lists $\mathcal{L}^M$ of men M over women W , and a partial matching μ' on (M, W) .
- Task:* Does there exist preferences of women $\mathcal{L}^W$, such that $\mu' \subseteq \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$?

These two problems and their differing computational complexities represent a dichotomy with respect to the size of target matching. Kobayashi and Matsui solve ASM by designing a novel combinatorial structure called the *suitor graph*, which encodes enough information about the men’s preferences and the matching pairs, that it allows an efficient search of the possible preference lists of women, which are $n \cdot n!$ in number. The same approach falls short when the stated matching is partial.

Our work falls thematically within the area of strategic results relating to the stable matching problem. There is a long history of results centered around the

question as to whether an individual agent, or a coalition of agents can misstate their true preference lists (either by truncating, or by permuting the list), with the objective of obtaining a better partner (assessed in terms of the true preferences of the manipulating agents) than would otherwise be possible under the men-optimal stable matching algorithm. SEOPM is to be viewed as a manipulation game in which a coalition of agents (in this case the subset of women who are matched under the partial matching) have decided upon a specific partner. These agents are colluding, with co-operation from the other women who are not matched, to produce a perfect matching, which gives each of the manipulating agents their target partners. There exists a strategy to attain this objective if and only if there exists a set of preference list of women that yields a perfect matching that contains the partial matching.

It is to be noted that a (successful) manipulation strategy for one game need not be a (successful) manipulation strategy for a game induced by some other mechanism. That is, manipulation strategies can only be defined for a known game, one where the mechanism is pre-defined. Alternately stated, a *manipulation game* is induced by a fixed mechanism. Gale-Shapley algorithm is perhaps the best known stable matching algorithm in theory and practice; its use in the manipulation games is ubiquitous. Our analysis in this chapter is about a manipulation game induced by the Gale-Shapley algorithm.

Combinatorial tools such as the suitor graph are by construction inextricably related to the men-optimal stable matching. However, it is not infeasible that for other notions of stable matchings we can design an equivalent combinatorial characterization such as a suitor graph, and a succinct description of all possible stable extensions of partial matchings such as our universal suitor graph.

Since SEOPM has been shown to be NP-complete, it is natural to study this problem in computational paradigms that are meant to cope with NP-hardness. We

attempt such a study in the area of exact exponential time algorithms. Manipulation and strategic issues in voting have been well-studied in the field of exact algorithms and parameterized complexity; see the survey [21] for an overview. But one cannot say the same regarding the strategic issues in the stable matching model. These problems hold a lot of promise and remain hitherto unexplored in the light of exact algorithms and parameterized complexity, with exceptions that are few and far between [116, 117].

To the best of our knowledge, Cseh and Manlove [28] initiated this type of analysis by studying an NP-hard variant of the stable marriage and *stable roommates* problems¹ where the input consists of each of the preference lists, as well two subsets of (not necessarily pairwise disjoint) pairs of agents, representing the *forbidden pairs* and the *forced pairs*. The goal is to find a matching that does not contain any of the forbidden pairs, and contains each of the forced pairs, while simultaneously minimizing the number of blocking pairs.

8.1.1 Our Contributions

Throughout the article, n is used to denote $n = |M| = |W|$. The most basic algorithm for SEOPM would be to guess the set of permutations for all women (that is, the set of preferences of women, $\mathcal{L}^W$) and check whether $\mu' \subseteq \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$. However, this algorithm will take $(n!)^n n^2 = 2^{\mathcal{O}(n^2 \log n)}$. One can obtain an improvement over this naïve algorithm by using the polynomial time algorithm for ASM [100]. That is, using the algorithm for ASM, which given a matching μ can check in polynomial time whether there exists $\mathcal{L}^W$ such that $\mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$. The faster algorithm for SEOPM, using the algorithm for ASM, tries all possible extensions μ of the

¹In the stable roommates problem, the matching market consists of agents of the same type, as opposed to the market modeled the stable marriage problem that consists of agents of two types, men and women. Roommates assignments in college housing facilities is a real world application of the stable roommates problem.

partial matching μ' and checks in polynomial time whether there exists $\mathcal{L}^W$ such that $\mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$. Thus, if the size of the partial matching is k , this algorithm would have to try $(n - k)!$ possibilities. In the worst case this can take $(n!)n^{\mathcal{O}(1)} = 2^{\mathcal{O}(n \log n)}$.

In this article we give a $2^{\mathcal{O}(n)}$ algorithm, which not only breaks the naïve bound, but also uses an idea which connects SEOPM to the problem of COLORED SUBGRAPH ISOMORPHISM (given two graphs G and H , the objective is to test whether H is isomorphic to some subgraph of G). We establish this connection by introducing a combinatorial tool, the *universal suitor graph* that extends the notion of the rooted *suitor graph* devised by Kobayashi and Matsui in [101, 100], to solve ASM. It is shown in [100] that an input instance $(\mathcal{L}^M, \mu)$ of ASM is a YES-instance if and only if the corresponding rooted suitor graph has an *out-branching*: a spanning subgraph in which every vertex has at most one in-coming arc, and is reachable from the root. The universal suitor graph satisfies the property that $(\mathcal{L}^M, \mu')$, an instance of SEOPM is a YES-instance if and only if the corresponding universal suitor graph contains a subgraph that is isomorphic to the out-branching corresponding to $(\mathcal{L}^M, \mu)$ where μ is the perfect matching that “extends” μ' . Thus, the universal suitor graph succinctly encodes all “possible suitor graphs” and is only polynomially larger than the size of a suitor graph. That is, the size of universal suitor graph is $\mathcal{O}(n^2)$. This is our main conceptual contribution and we believe that the concept of the universal suitor graph is likely to be of independent interest, useful in characterizing existence of strategies in other manipulation games.

Using ideas from exact exponential algorithms and parameterized complexity; in particular by using as a subroutine the algorithm that enumerates all non-isomorphic out-branchings in a (given) rooted directed graph [14, 130], and a parameterized algorithm for COLORED SUBGRAPH ISOMORPHISM [52, 53], we can search for a subgraph in the universal suitor graph that is isomorphic to an out-branching corresponding to an extension of μ' . We complement our algorithmic finding by

showing that unless Exponential Time Hypothesis (ETH) fails, our algorithm is asymptotically optimal. That is, unless ETH fails, there is no algorithm for SEOPM running in time $2^{o(n)}$. Using the tools developed for SEOPM we can solve SEOPMI, the problem of finding a stable extension of a partial matching when the preference lists of men are not necessarily complete. This problem is stated in Section [8.5](#)

8.2 Preliminaries

As introduced earlier, M and W denote the set of men and women, respectively, and we assume that $|M| = |W| = n$. Each $m \in M$ has a preference list, denoted by $P(m)$, which is a total ordering of W . The set of preference lists of all men is denoted by $\mathcal{L}^M$. Similarly, each $w \in W$ has a preference list, denoted by $P(w)$ which is a total ordering of M . The set of preference lists of all women is denoted by $\mathcal{L}^W$. It is helpful to view (M, W) as the bipartitions of a complete bipartite graph, and a perfect matching in (M, W) as a set of vertex disjoint edges that matches every vertex in $M \cup W$. Similarly, a partial matching in (M, W) can be viewed as a set of vertex disjoint edges that does not necessarily match every vertex in $M \cup W$. Given a matching μ (perfect or partial), and a vertex $v \in M \cup W$, $\mu(v)$ denotes the matched partner of the man/woman v . We note that for a perfect matching μ : $m \in M$ if and only if $\mu(m) \in W$, and similarly $w \in W$ if and only if $\mu(w) \in M$. But, when we have a partial matching, μ , it may be that some vertices (male or female) are not matched under it, we denote that symbolically as $\mu(v) = v$ for any man/woman $v \in M \cup W$ who is not matched in μ . A matching μ is said to be **an extension** of a matching μ' if $\mu' \subseteq \mu$, that is μ contains the set of edges in μ' . For any matching μ , and a man m matched in μ , we define $\delta^+(m) = \{w \in W \mid m \text{ strictly prefers } w \text{ to } \mu(m)\}$, and conversely for any woman $w \in W$ (not necessarily matched in μ) we define $\delta^-(w) = \{m \in M \mid m \text{ strictly prefers } w \text{ to } \mu(m)\}$; all preferences are in terms of lists in $\mathcal{L}^M$ and $\mathcal{L}^W$.

Throughout the chapter, we use the standard notations about directed graphs. Given a directed graph D , and a vertex $v \in V(D)$, we use $N^-(v)$ to denote the set of vertices that are in-neighbors of v : $N^-(v) = \{u \mid (u, v) \in E(D)\}$. Similarly, we use $N^+(v)$ to denote the set of vertices that are out-neighbors of v : $N^+(v) = \{u \mid (v, u) \in E(D)\}$. Following the usual notations, a source is a vertex v such that $N^-(v) = \emptyset$ and a sink is a vertex v such that $N^+(v) = \emptyset$. An **out-branching** is a directed graph with a special vertex, called the *root*, where each vertex is reachable from the root by exactly one directed path. Essentially, this is a rooted tree with all arcs oriented away from the root. For any directed edge or an arc, *tail* is the vertex from where the arc originates and the *head* is the vertex at which it ends.

8.3 Generalization of Suitor Graph

The main tool we use to obtain our exact exponential time algorithm is the notion of a *universal suitor graph* – a generalization of the *suitor graph* introduced by Kobayashi and Matsui [100]. We start the section by introducing the definition of a suitor graph, followed by the definition of a universal suitor graph.

Suitor Graph and Rooted Suitor Graph. Given a set of preference lists $\mathcal{L}^M$ of men over set of women W and a partial matching μ' , $G(\mathcal{L}^M, \mu')$ denotes a directed bipartite graph, called a *suitor graph*, where $V(G) = M \cup W$ and a set of directed arcs $E(G)$ defined as follows,

$$\begin{aligned} E(G) = & \left\{ (w, \mu'(w)) \in W \times M \mid w \text{ is matched in } \mu' \right\} \\ & \cup \left\{ (m, w) \in M \times W \mid m \text{ is matched in } \mu', w \in \delta^+(m) \right\} \end{aligned}$$

Observe that the arcs for which a woman is the tail are the (only) arcs that correspond to the matched pairs in μ' .

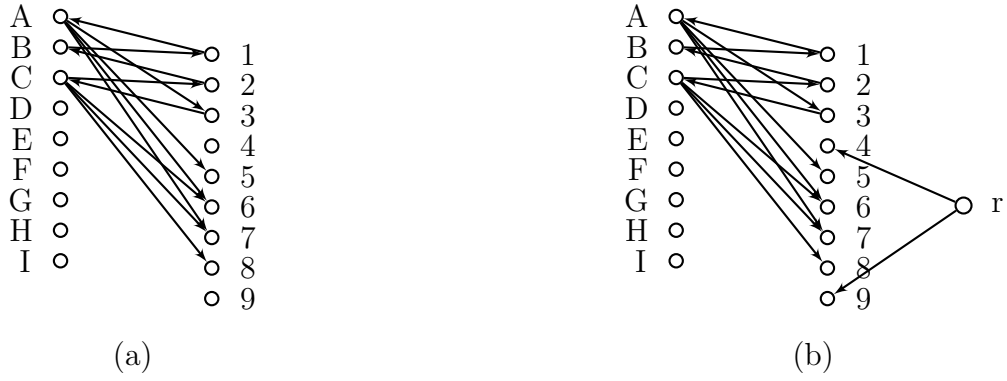


Figure 8.1: (a) Suitor Graph, (b) Rooted Suitor Graph

For a given suitor graph $G(\mathcal{L}^M, \mu')$, the associated *rooted suitor graph* is a directed graph $\bar{G}(\mathcal{L}^M, \mu')$ defined as follows. We introduce an artificial vertex r , called the *root*, to $G(\mathcal{L}^M, \mu')$ and add arcs (r, w) for every woman $w \in W$ who has no incoming arc in $G(\mathcal{L}^M, \mu')$; such a vertex w is called a *source* in $G(\mathcal{L}^M, \mu')$. We give an example of a suitor graph and a rooted suitor graph. Figure 8.1 shows the suitor graph and the rooted suitor graph for the preference lists given in Table 8.1 and the partial matching $\{(A, 1), (B, 2), (C, 3)\}$. The vertex marked as r is the root vertex.

Man	Preference over women									
A	3	7	6	5	1	9	8	4	2	
B	1	2	4	3	9	5	8	7	6	
C	2	7	6	8	3	4	9	1	5	
D	2	7	6	8	3	4	9	1	5	
E	3	7	6	5	1	9	8	4	2	
F	1	2	4	3	9	5	8	7	6	
G	3	7	6	5	1	9	8	4	2	
H	1	2	4	3	9	5	8	7	6	
I	2	7	6	8	3	4	9	1	5	

Table 8.1: Example: Preference List of Men over Women

Our main motivation for suitor graph and its generalization is the following result proved by Kobayashi and Matsui [100].

Proposition 8.3.1. [100, Theorem 2] Let $\mathcal{L}^M$ be a set of preference lists for M , and μ be a perfect matching between (M, W) , then the following holds: There exists $\mathcal{L}^W$,

a set of preference lists for W such that $\text{GS}(\mathcal{L}^M, \mathcal{L}^W) = \mu$ if and only if the rooted suitor graph $\bar{G}(\mathcal{L}^M, \mu)$ has an out-branching.

There exists a polynomial time algorithm (**Algorithm Q1** in [100]) that takes as input $(\mathcal{L}^M, \mu)$ and outputs $\mathcal{L}^W$ (if one exists) such that $\text{GS}(\mathcal{L}^M, \mathcal{L}^W) = \mu$. Otherwise, it reports “none exists”. We remark that the preference lists in $\mathcal{L}^M$ and $\mathcal{L}^W$ (if it exists) are complete. We will be using **Algorithm Q1** as a subroutine in Algorithm 8.4.2.

Universal Suitor Graph. Next we define *universal suitor graph (USG)*. The idea is to construct a graph that given a set of preference lists $\mathcal{L}^M$ of men over women captures all possible suitor graphs succinctly. Then we make use of this to solve our problem. Formally, given a set of preference lists $\mathcal{L}^M$ of men over women, universal suitor graph, $U(\mathcal{L}^M)$, is defined as follows. We make n different copies of each man $m_i \in M$, denoted by $M_i = \{m_i^1, \dots, m_i^n\}$. Recall that for every $m_i \in M$, the preference list $P(m_i) \in \mathcal{L}^M$ is given. We define $P(m_i^j) = P(m_i)$, for $1 \leq j \leq n$. Thus, the vertex set of the graph is $V(U(\mathcal{L}^M)) = \biguplus_{i=1}^n M_i \cup W$. The arc set, $E(U(\mathcal{L}^M))$, is defined as follows. For every $w_i \in W$, the graph contains arcs (w_i, m_j^i) for all $1 \leq j \leq n$. Additionally, the graph contains the arc (m_j^i, w_k) if m_j prefers w_k to w_i in $P(m_j)$, where $w_k, w_i \in W$. This condition is depicted notationally as $w_k >_{m_j} w_i$. The intuition behind the construction is the following: for any matching μ , the possibility that man m_j will be matched with woman w_i , is captured by the edge (w_k, m_j^k) . Furthermore, using other copies of m_j we imitate connections with women whom he prefers to w_k . In particular, the i^{th} copy of every man is “paired” to w_i , i.e., $N^+(w_i) = \{m_1^i, m_2^i, \dots, m_n^i\}$. This idea of pairing is captured by the fact that every male vertex in USG (consider m_k^i) has a unique in-neighbor (the female vertex w_i).

Universal Suitor Graph for a Partial Matching. For a given partial matching μ on the set (M, W) , we define the graph, $U(\mathcal{L}^M, \mu)$, as follows. A man $m \in M$ is matched under μ if and only if $\mu(m) \in W$, and analogously for a woman $w \in W$,

w is matched under μ if and only if $\mu(w) \in M$. We refer to the following set of operations collectively as the **pruning** of $U(\mathcal{L}^M)$ w.r.t. μ .

Matched Women : Let $\mu(w_i) = m_j$. Then delete vertices $\{m_k^i \mid 1 \leq k \leq n, k \neq j\}$, from the graph. This ensures that every matched female vertex w_i , has a unique out-going arc to the i^{th} copy of the man $\mu(w_i)$. In other words, only the arc (w_i, m_j^i) , where $\mu(w_i) = m_j$, survives.

Unmatched Women : Let $\mu(w_i) \notin M$. Then delete vertices $\{m_k^i \mid 1 \leq k \leq n, \mu(m_k) \in W\}$. That is, delete the i^{th} copy of a man who is matched under μ . This ensures that in the subgraph, every unmatched female vertex w_i has out-going arcs to the vertices in the set $\{m_k^i \mid m_k \text{ is unmatched in } \mu\}$.

This completes the description of the pruning operations. Thus, to obtain the graph $U(\mathcal{L}^M, \mu)$ we start with $U(\mathcal{L}^M)$ and apply the pruning operations defined above with respect to the matching μ . Edges in $U(\mathcal{L}^M)$ that are not deleted during the above pruning operations, are said to have *survived pruning* w.r.t. μ . We give an example of a universal suitor graph for a partial matching. Figure [8.2](#) shows the universal suitor graph and the rooted universal suitor graph [described later in Section [8.4.2](#) for the preference lists given in Table [8.1](#) and for the partial matching $\mu = \{(A, 1), (B, 2), (C, 3)\}$. To keep the figure clear, we only show the copies of male vertices for two women. The edges going out of these copies of the male vertices are omitted.

We conclude this discussion with a useful lemma that will be invoked in several arguments.

Lemma 8.3.1. *Let μ denote a partial matching. If a male vertex m_j^i survives pruning w.r.t. μ , then either $\mu(m_j) = w_i$, or else both m_j and w_i are unmatched in μ . Furthermore, the out-going arcs from m_j^i are also not deleted during pruning operations.*

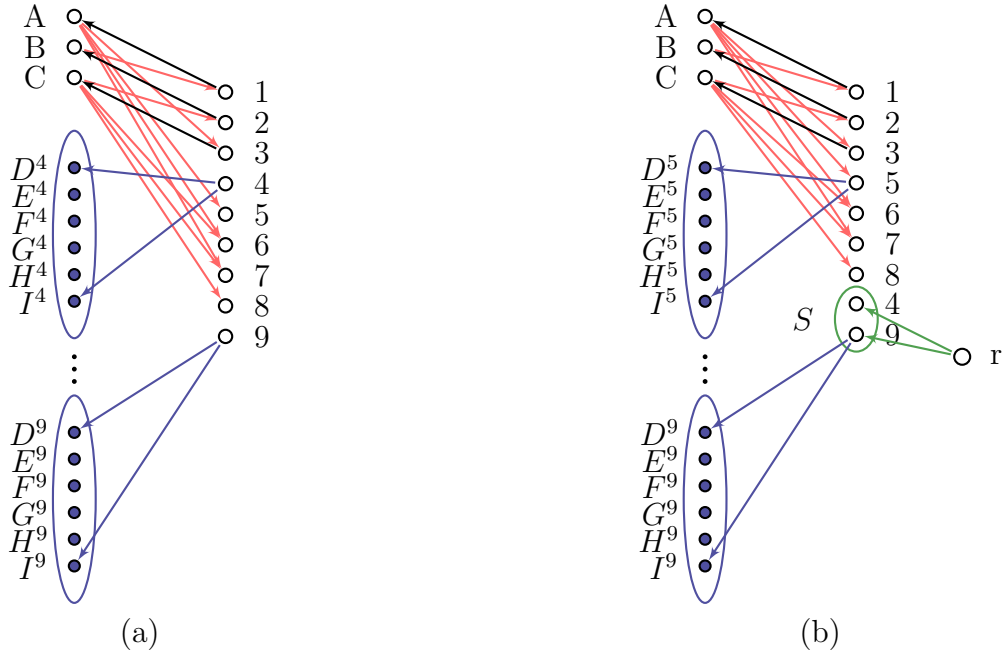


Figure 8.2: (a) Universal suitor graph on vertices $M = \{A, B, C\} \uplus_{X \in \{D, \dots, I\}} \{X^i \mid 4 \leq i \leq 9\}$ and $W = \{1, \dots, 9\}$, (b) Rooted universal suitor graph [described later in Section [8.4.2](#)] for the partial matching $\mu = \{(A, 1), (B, 2), (C, 3)\}$ with source $\{4, 9\}$. Blue vertices are copies of the unmatched male vertices for each unmatched female vertex. The outgoing edges from blue vertices are not shown. Black edges represent the matching edges in μ , red edges represent the preferences of men matched in μ , while the blue edges represent edges from an unmatched woman to her own copies of the unmatched men. The green ellipse represents the set of source vertices, $\{4, 9\}$, that are connected from the root r .

Proof. We begin by noting that if w_i is matched to someone other than m_j , then the vertex m_j^i must be deleted during the pruning step; this is a contradiction. Suppose that w_i is unmatched, and $\mu(m_j) = w_\ell$. Then, the arc (w_ℓ, m_j^ℓ) survives, but the vertex m_j^i must be deleted, again a contradiction. Hence, the fact that m_j^i survives pruning w.r.t. μ , implies that either $\mu(m_j) = w_i$, or both m_j, w_i are unmatched.

Additionally, we note that if $\mu(m_j) = w_i$, then m_j^i is the sole member of M_j that survives the pruning steps. Also note that regardless of whether m_j is matched or unmatched, the out-going arcs from m_j^i survive the pruning process. $\diamond$

8.4 Exact Algorithm for SEOPM

In this section we design a moderately exponential time algorithm for SEOPM.

Towards this we will combine the following three ingredients:

- the notion of a universal suitor graph defined in the previous section;
- a parameterized algorithm for SUBGRAPH ISOMORPHISM when the pattern graph has bounded treewidth; and
- the fact that the number of non-isomorphic (i.e. unlabelled) trees on n vertices is at most $2.956^n n^{\mathcal{O}(1)}$.

We start this section by giving an overview of our algorithms. Towards this we first give the relevant notions and definitions.

Definition 8.4.1. *Two digraphs G_1 and G_2 are said to be isomorphic if there is a function $f : V(G_1) \rightarrow V(G_2)$ that satisfies the following properties:*

1. *f is a bijective function, i.e., f^{-1} is a function from $V(G_2)$ to $V(G_1)$;*
2. *for every edge $(u, v) \in E(G_1)$, we have $(f(u), f(v)) \in E(G_2)$.*

A function such as f is called an isomorphism function. This function can be extended to sets of vertices analogously. That is, for all $V_1 \subseteq V(G_1)$, $f(V_1) = \{f(v) \mid v \in V_1\} \subseteq V(G_2)$. We write $G_1 \simeq G_2$ to denote the two graphs are isomorphic.

Now we are ready to define the COLORED SUBGRAPH ISOMORPHISM problem. The COLORED SUBGRAPH ISOMORPHISM problem is formally defined as follows.

COLORED SUBGRAPH ISOMORPHISM (COL-SUB-ISO)

Input: A host graph G , a pattern graph H , and a coloring $\chi : V(G) \rightarrow \{1, 2, \dots, |V(H)|\}$.

Parameter: $|V(H)|$

Task: Is there a subgraph G' in G such that $G' \simeq H$, and the vertices of G' have distinct colors?

We obtain the desired algorithm by making $2^{\mathcal{O}(n)}$ instances of the COL-SUB-ISO problem where the pattern graph has size $2n + 3$ and treewidth 3, and the given instance of SEOPM is a YES instance if and only if one of the constructed instances is a YES instance of the COL-SUB-ISO problem. Our host graph will be a universal suitor graph (of size $\mathcal{O}(n^2)$) corresponding to an instance of SEOPM. We refer the reader to [29] for definitions of treewidth and tree decomposition. To solve COL-SUB-ISO we will use known algorithms, in particular, the algorithm alluded to in the following result by Amini, Fomin, and Saurabh [6, Theorem 15].

Proposition 8.4.1 ([6]). *Let G and H denote two graphs on N and q vertices, respectively such that the treewidth of H is at most t . Furthermore, there is a coloring $\chi : V(G) \rightarrow [q]$ of G . Then, there is a deterministic algorithm for COL-SUB-ISO that runs in time $2^q(Nt)^{t+\mathcal{O}(1)}$, and outputs (if there exists one) a subgraph of G that has a distinct color on every vertex, and is isomorphic to H .*

To give the desired reduction to COL-SUB-ISO we essentially enumerate all non-isomorphic trees on $2n + 1$ vertices. In the past, rooted (undirected) trees have been studied mainly, not much study has taken place on rooted directed trees or out-branchings. However, every rooted tree can be made an out-branching by orienting every edge away from the root and every out-branching can be transformed into a rooted tree by disregarding all edge orientations. Thus, rooted trees and out-branchings are equivalent, and thus, the results obtained for the former are

applicable to the latter. Otter [130] showed that the number of non-isomorphic out-branchings on n vertices is $t_n = 2.956^n n^{\mathcal{O}(1)}$. We can generate all non-isomorphic rooted trees on n vertices using the algorithm of Beyer and Hedetniemi [14] of runtime $\mathcal{O}(t_n)$. We summarize the above in the following result.

Proposition 8.4.2 ([14, 130]). *The number of non-isomorphic out-branchings on n vertices is $t_n = 2.956^n n^{\mathcal{O}(1)}$. Furthermore, we can enumerate all non-isomorphic rooted trees on n vertices in time $\mathcal{O}(t_n)$.*

8.4.1 Universality of Universal Suitor Graph

In this section we show the “universality” of the universal suitor graph. That is, how given a set of preference lists, $\mathcal{L}^M$, of men over women, universal suitor graph encodes all potential suitor graphs. Universal suitor graph for a partial matching encodes all suitor graphs of all potential extensions of the given partial matching. In particular, we show the following result.

Lemma 8.4.1. *Let $\mathcal{L}^M$ denote a set of preference lists of men over women and let μ' denote a partial matching on the set (M, W) . If there exists a perfect matching μ such that $\mu' \subseteq \mu$ (as a set of edges), then $U(\mathcal{L}^M, \mu)$ is a subgraph of $U(\mathcal{L}^M, \mu')$, and is isomorphic to the suitor graph $G(\mathcal{L}^M, \mu)$.*

Proof. Let M' and W' denote the subset of men and women who are matched under μ' , respectively. Let $\mu' \subseteq \mu$, in terms of a subset of edges. We will refer to the suitor graphs $G(\mathcal{L}^M, \mu')$ and $G(\mathcal{L}^M, \mu)$ as simply suitor graphs for μ' and μ , respectively.

Consider the universal suitor graph for μ , denoted by $U(\mathcal{L}^M, \mu)$, obtained from $U(\mathcal{L}^M)$ by pruning w.r.t. μ . Since $\mu' \subseteq \mu$ for every $w \in W'$ ($m \in M'$) we have $\mu(w) = \mu'(w)$ ($\mu(m) = \mu'(m)$). Thus, it is easy to see that $U(\mathcal{L}^M, \mu)$ is a subgraph of $U(\mathcal{L}^M, \mu')$, and can be obtained from the latter by applying the pruning operation to every female vertex $w_i \in W \setminus W'$. The next claim completes the proof of the lemma

because it leads to the conclusion that the suitor graph $G(\mathcal{L}^M, \mu)$ is isomorphic to the universal suitor graph $U(\mathcal{L}^M, \mu)$.

Claim 2. *Suitor graph $G(\mathcal{L}^M, \mu)$ is isomorphic to $U(\mathcal{L}^M, \mu)$.*

Proof. By the construction of $G(\mathcal{L}^M, \mu)$, we know the suitor graph of μ has arcs $(w, \mu(w))$ for every $w \in W$. Since $U(\mathcal{L}^M, \mu)$ is obtained from $U(\mathcal{L}^M)$ by pruning w.r.t. μ , hence we know that $U(\mathcal{L}^M, \mu)$ contains $2|\mu|$ vertices

$$\bigsqcup_{w_i \in W} \{w_i, m_j^i \mid \mu(w_i) = m_j\}.$$

We use M^μ to denote the male vertices in $U(\mathcal{L}^M, \mu)$.

Let $\Psi_\mu : M \cup W \rightarrow M^\mu \cup W$ denote a function between the vertex sets of $G(\mathcal{L}^M, \mu)$ and $U(\mathcal{L}^M, \mu)$. For every $w_i \in W$, we define $\Psi_\mu(w_i) = w_i$, and for every $m_i \in M$, we define $\Psi_\mu(m_i) = m_j^i$, where $\mu(m_i) = w_j$. We will prove that the map Ψ_μ is an isomorphism.

We begin with the observation that both graphs are bipartite, with vertex set (M, W) and (M^μ, W) . Thus, to prove that Ψ_μ is an isomorphism, it is sufficient to prove that for every $w \in W, m \in M$, (w, m) is an arc in $G(\mathcal{L}^M, \mu)$ if and only if $(w, \Psi_\mu(m))$ is an arc in $U(\mathcal{L}^M, \mu)$, and similarly (m, w) is an arc in $G(\mathcal{L}^M, \mu)$ if and only if $(\Psi_\mu(m), w)$ is an arc in $U(\mathcal{L}^M, \mu)$.

Let (w_i, m_j) be an arc in $G(\mathcal{L}^M, \mu)$. Thus, we have $\mu(w_i) = m_j$, and so $\Psi_\mu(m_j) = m_j^i$. The construction of $U(\mathcal{L}^M, \mu)$ (that is pruning w.r.t. μ) ensures that (w_i, m_j^i) is an arc in $U(\mathcal{L}^M, \mu)$. Conversely, if (w_i, m_j^i) is an arc in $U(\mathcal{L}^M, \mu)$ then since μ is a perfect matching, by Lemma [8.3.1](#), we can conclude that $\mu(w_i) = m_j$, and so (w_i, m_j) is an arc in $G(\mathcal{L}^M, \mu)$. This completes the proof of the if and only if statement about female to male arcs.

Let (m_i, w_k) be an arc in $G(\mathcal{L}^M, \mu)$ i.e., $\mu(m_i) = w_j$. Thus, $w_k >_{m_i} w_j$ (m_i

prefers w_k to w_j in $\mathcal{L}^M$). The vertex $m_i^j = \Psi_\mu(m_i)$ and the arc (m_i^j, w_k) exists in the universal suitor graph $U(\mathcal{L}^M)$. If we can show that m_i^j exists in $U(\mathcal{L}^M, \mu)$, then by the additional condition of Lemma 8.3.1, we know that the arc (m_i^j, w_k) exists in $U(\mathcal{L}^M, \mu)$. We note that m_i^j must survive the pruning of $U(\mathcal{L}^M)$ w.r.t. μ because (w_j, m_i) is an arc in $G(\mathcal{L}^M, \mu)$ and so from the earlier part we know that (w_j, m_i^j) is an arc in $U(\mathcal{L}^M, \mu)$. Hence, m_i^j must be a vertex in $U(\mathcal{L}^M, \mu)$, and so we conclude that $(\Psi_\mu(m_i), w_k)$ is an arc in $U(\mathcal{L}^M, \mu)$. Conversely, if (m_i^j, w_k) is an arc in $U(\mathcal{L}^M, \mu)$, then the presence of m_i^j in the graph allows us to invoke Lemma 8.3.1 to conclude that $\mu(m_i) = w_j$. This implies that $w_k >_{m_i} w_i$, hence (m_i, w_k) must also be an arc in $G(\mathcal{L}^M, \mu)$. This completes the proof of the if and only if statement about male to female arcs. Hence, our proof is complete.

◇

Thus, lemma is proved.

◇

8.4.2 Rooted Universal Suitor Graph and Valid Subgraphs

For a given universal suitor graph $U(\mathcal{L}^M, \mu')$ and a subset $S \subseteq W$, we define the corresponding **rooted universal suitor graph with sources in S** , as follows. For a vertex $w \in S$, if w is a source in $U(\mathcal{L}^M, \mu')$ (i.e. $N^-(w) = \emptyset$) then we add the arc (r, w) . Otherwise, we delete all the male vertices in $N^-(w)$, and add the arc (r, w) . The resulting graph is the rooted universal suitor graph with sources in S , and is denoted by $\bar{U}(\mathcal{L}^M, \mu', S)$. We refer the reader to Figure 8.2(b) for an example of a rooted universal suitor graph. The set of vertices marked as S is the set of source vertices that are connected to the root.

Recall that in a universal suitor graph for a partial matching there may be

multiple copies of a male vertex, and that brings us to the notion of a valid subgraph. A subgraph of $U(\mathcal{L}^M, \mu')$ is said to be a **valid subgraph** if it contains every female vertex, and exactly one copy of every male vertex, i.e. $|U(\mathcal{L}^M, \mu') \cap M_i| = 1$ for each i ($1 \leq i \leq n$). The definition can be extended to the rooted subgraphs of $\bar{U}(\mathcal{L}^M, \mu', S)$, where $S \subseteq W$, and a valid rooted subgraph contains the root, every female vertex and exactly one copy of every male vertex.

Consider a rooted tree, such that the root is considered to be in layer 0. A vertex v is said to be in layer i in the tree, if the (unique) path from the root to v contains i arcs. A rooted tree is called a **matching tree** if every vertex in an odd layer has a unique child in the tree. If a matching tree is a valid subgraph of $U(\mathcal{L}^M, \mu')$ then it is called a **valid matching tree**. We note that a matching tree is also an out-branching.

Given a matching tree T , we construct the **triangular matching tree** T^Δ , by adding two new vertices r^1 and r^2 to T and adding the arcs (r, r^1) , (r^1, r^2) and (r^2, r) . Similarly, for any given rooted universal suitor graph $\bar{U}(\mathcal{L}^M, \mu', S)$, we construct the **triangular rooted universal suitor graph**, $\bar{U}^\Delta(\mathcal{L}^M, \mu', S)$, by adding two new vertices r^1 and r^2 to T and adding the arcs (r, r^1) , (r^1, r^2) and (r^2, r) .

Finally, we define the special coloring χ_{sp} used to color the vertices of a triangular rooted universal suitor graph: χ_{sp} uses $2n + 3$ colors, giving distinct colors to $r, r^1, r^2, w_1, \dots, w_n$, and using the remaining n colors such that the subset of copies of the same male vertex gets a distinct color. That is, for each i ($1 \leq i \leq n$) the subset of $\{m_i^1, \dots, m_i^n\}$ that exists in the universal suitor graph gets the $n + 3 + i^{th}$ color.

8.4.3 $2^{\mathcal{O}(n)}$ Algorithm for SEOPM

In this section we combine all the results we have developed so far and design our algorithm.

Overview of Algorithm 8.4.1: Let $(\mathcal{L}^M, \mu')$ be an input instance of SEOPM. If μ' can be extended to μ , then (by Lemma 8.4.1), we know that $G(\mathcal{L}^M, \mu)$ is isomorphic to a subgraph in $U(\mathcal{L}^M, \mu')$. If μ' cannot be extended, then by Proposition 8.3.1 we know that for any perfect matching $\mu \supseteq \mu'$, the graph $\bar{G}(\mathcal{L}^M, \mu)$ does not contain an out-branching rooted at r . In other words, there exists a vertex v that is not reachable from r in the graph $\bar{G}(\mathcal{L}^M, \mu)$. Consequently, to “solve” SEOPM on $(\mathcal{L}^M, \mu')$, it is necessary and sufficient to look for a *valid out-branching or matching tree in the universal suitor graph* $U(\mathcal{L}^M, \mu')$. If the algorithm finds one, we can conclude that μ' can be extended, else it answers that μ' cannot be extended. We implement these ideas by constructing an appropriate instance of COL-SUB-ISO.

The algorithm works as follows. Assume that we have a stable matching μ that extends μ' . Then consider the graph $G(\mathcal{L}^M, \mu)$ and let S denote the subset of female vertices that are sources in the graph. Our algorithm implements this by enumerating all subsets S of W in the first loop. Furthermore, by Proposition 8.3.1 there is a matching tree, T , rooted at r in $\bar{G}(\mathcal{L}^M, \mu)$. To “guess” the tree T , we enumerate all non-isomorphic out-branchings on $2n + 1$ vertices and first check whether it is a matching tree. If the enumerated tree is a matching tree then we create an instance of COL-SUB-ISO, where the host graph is $\bar{U}^\Delta(\mathcal{L}^M, \mu', S)$, with its vertices colored by χ_{sp} , and the pattern graph is T^Δ . Finally, using an algorithm for COL-SUB-ISO described in Proposition 8.4.1, we test whether, or not $(\bar{U}^\Delta(\mathcal{L}^M, \mu', S), T^\Delta, \chi_{sp})$ is a YES-instance of COL-SUB-ISO. If the algorithm returns T^* , we can conclude that a stable matching μ extends μ' . If the outermost for-loop terminates without finding a YES-instance of COL-SUB-ISO, then we return that “no valid out-branching exists” (and hence no stable extension exists). This concludes the description of

Algorithm 8.4.1: Subroutine for solving SEOPM.

Input: A set of men and women vertices (M, W) , preferences of men $\mathcal{L}^M$,
and a partial matching μ'

```

1
2 Let  $\mathcal{F} \leftarrow \{\text{non-isomorphic out-branchings on } 2n + 1 \text{ vertices}\}$ 
3 forall  $S \subseteq W$  do
4   forall matching tree  $T \in \mathcal{F}$  do
5     Using Proposition 8.4.1 test whether  $(\bar{U}^\Delta(\mathcal{L}^M, \mu', S), T^\Delta, \chi_{sp})$  is a
       YES-instance of COL-SUB-ISO.
6     if the algorithm returns a subgraph  $T^*$  then
7       return  $T^*$ 
8     end
9   end
10 end
11 return "No valid out-branching exists"

```

the algorithm. We refer the reader to Algorithm 8.4.1 for further details. The next lemma argues the correctness of Algorithm 8.4.1.

Lemma 8.4.2. *Let $(\mathcal{L}^M, \mu')$ be an instance of SEOPM. Then $(\mathcal{L}^M, \mu')$ is a YES-instance of SEOPM if and only if Algorithm 8.4.1 returns a graph (valid triangular matching tree) on the input $(\mathcal{L}^M, \mu')$.*

Proof. Let $(\mathcal{L}^M, \mu')$ be a YES-instance, i.e., there exists $\mathcal{L}^W$ such that $\mu' \subseteq \mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$. Note that since the preference lists in $\mathcal{L}^M$ and $\mathcal{L}^W$ are complete, every stable matching w.r.t. $(\mathcal{L}^M, \mathcal{L}^W)$ (such as μ) is a perfect matching. By Lemma 8.4.1, we know that $G(\mathcal{L}^M, \mu)$ is isomorphic to a subgraph in $U(\mathcal{L}^M, \mu')$. Let $\tilde{T}$ denote the out-branching in $\bar{G}(\mathcal{L}^M, \mu)$ (rooted at r), as described by Proposition 8.3.1. By Lemma 8.4.1, we know that $G(\mathcal{L}^M, \mu)$ is isomorphic to a subgraph in $U(\mathcal{L}^M, \mu')$. Consequently, the rooted suitor graph $\bar{U}(\mathcal{L}^M, \mu', S^*)$, where S^* denotes the set of sources in $G(\mathcal{L}^M, \mu)$, must contain a valid matching tree T' that is isomorphic to $\tilde{T}$. If we delete the labels on the vertices in T' (or $\tilde{T}$), we get an out-branching (in fact, a matching tree) on $2n + 1$ vertices, denoted by T . Thus, $T \in \mathcal{F}$, and we conclude that Algorithm 8.4.1 will find T^* , a valid triangular matching tree of $\bar{U}^\Delta(\mathcal{L}^M, \mu', S^*)$ that is isomorphic to T^Δ . Hence, the algorithm will return T^* .

Suppose that Algorithm [8.4.1](#) outputs T^* . Then there exists a subset $S \subseteq W$, and an out-branching on $2n + 1$ vertices T , such that $\bar{U}^\Delta(\mathcal{L}^M, \mu', S)$ contains T^* as a (colored) subgraph such that T^* is isomorphic to the triangular matching tree T^Δ . Observe that $\bar{U}^\Delta(\mathcal{L}^M, \mu', S)$ has a unique triangle r, r^1, r^2 and thus due to the isomorphism, T^* contains the triangle r, r^1, r^2 . This implies that every vertex in T^* is reachable from the root of $\bar{U}(\mathcal{L}^M, \mu', S)$. Since male vertices are only reachable from a female vertex, this means that every male vertex has an in-coming female neighbor. Since, $T^* \setminus \{r^1, r^2\}$ is a valid matching tree of $\bar{U}(\mathcal{L}^M, \mu', S)$, there is exactly one copy of every male vertex and every female vertex has a unique out-neighbor. Thus, if (w_i, m_k^i) is a female to male arc in T^* , then T^* does not contain any other out-going arc from w_i . Thus, the female to male arcs in T^* yield a perfect matching. We denote it by μ . Note that $\mu' \subseteq \mu$ because $U(\mathcal{L}^M, \mu')$ contains a unique out-going arc for every matched woman in μ' , hence those arcs must also be part of T^* . Hence, we can conclude that $T^* \setminus \{r^1, r^2\}$ is an out-branching in the graph $\bar{G}(\mathcal{L}^M, \mu)$. By Proposition [8.3.1](#), this means that $(\mathcal{L}^M, \mu')$ is a YES-instance of SEOPM. This concludes the proof.

◇

The next lemma gives the running time of Algorithm [8.4.1](#).

Lemma 8.4.3. *Let $(\mathcal{L}^M, \mu')$ be an instance of SEOPM, where $|M| = n$. Then, Algorithm [8.4.1](#) decides whether $(\mathcal{L}^M, \mu')$ is a YES-instance to SEOPM in time $2^{\mathcal{O}(n)}$.*

Proof. The running time of the algorithm is upper bounded by the following formula

$$|\{S \subseteq W\}| \times |\mathcal{F}| \times \text{Time taken by COL-SUB-ISO algorithm}$$

By applying Proposition [8.4.2](#) we upper bound $|\mathcal{F}|$ by $2.956^{2n+1}n^{\mathcal{O}(1)}$. It is a well-known fact that the treewidth of a tree is one, from that it is easy to show that

the treewidth of a triangular matching tree is at most 3. (One can first find the tree-decomposition of the tree and then add the two vertices r^1, r^2 to every bag and thus increasing the treewidth by at most two. See [29, Chapter 7] for more details regarding treewidth.) Thus, when we apply Proposition 8.4.1, we have a host graph that has at most $n + n^2 + 3$ vertices, and a pattern graph that has size $2n + 3$ and treewidth at most 3. Therefore, the running time for using the subroutine for COL-SUB-ISO is $2^{2n+3}n^{\mathcal{O}(1)}$. Multiplying all these values together, gives the overall running time to be $2^n \times 2.956^{2n+1}n^{\mathcal{O}(1)} \times 2^{2n+3}n^{\mathcal{O}(1)} = 2^{\mathcal{O}(n)}$.

◇

Combining Lemmas 8.4.2 and 8.4.3 we get the following theorem.

Algorithm 8.4.2: Solves SEOPM

Input: A set of men and women vertices (M, W) , preferences of men $\mathcal{L}^M$, and a partial matching μ'

```

1
2 if Algorithm 8.4.1 on  $(\mathcal{L}^M, \mu')$  returns “No valid out-branching exists” then
3   | return “No stable extension exists”
4 end
5 if Algorithm 8.4.1 on  $(\mathcal{L}^M, \mu')$  returns a graph  $T^*$  then
6   |  $\triangleright T^*$  is a valid triangular matching tree in  $\bar{U}^\Delta(\mathcal{L}^M, \mu', S)$  for some subset
   |    $S \subseteq W$ , with triangle  $\{r, r^1, r^2\}$ .
7   |  $T \leftarrow T^* \setminus \{r^1, r^2\}$ 
8 end
9 Compute matching  $\mu$  by pairing every woman in  $T$  to her unique
   out-neighbor.
10 Invoke Algorithm Q1 in [100] on input  $(\mathcal{L}^M, \mu)$  to obtain a set of
    preferences  $\mathcal{L}^W$ .
11 return  $(\mathcal{L}^W, \mu)$ 

```

Theorem 24. Algorithm 8.4.2 solves SEOPM In time $2^{\mathcal{O}(n)}$.

Proof. Correctness and running time of Algorithm 8.4.2 follow from Lemmas 8.4.2 and 8.4.3.

◇

8.4.4 A Lower Bound under Exponential Time Hypothesis

In this section we show that Theorem 24 is asymptotically optimal. That is, barring an unlikely scenario occurring in complexity theory, there cannot be a better algorithm for SEOPM. To prove this we will invoke the Exponential Time Hypothesis (ETH) [81], and use the well-known NP-hardness reduction from SAT to SEOPM.

Exponential Time Hypothesis (ETH): Let τ denote the infimum of the set of constants c for which there exists an algorithm solving 3-SAT in time $\mathcal{O}(2^{cn}n^{\mathcal{O}(1)})$. Then, $\tau > 0$.

ETH introduced by Impagliazzo and Paturi [81] has been extensively used recently to obtain tight lower bounds for several problems. We use this here to get a lower bound on the running time possible for SEOPM. To this end we will use the following result stated in [29, Theorem 14.4].

Theorem 25 (Sparsification Lemma [29]). *Unless ETH fails, there exists a constant $c > 0$ such that no algorithm for 3-SAT can achieve running time $\mathcal{O}(2^{c(n+m)}n^{\mathcal{O}(1)})$. In particular, 3-SAT cannot be solved in time $2^{o(n+m)}$. Here, n and m denote the number of variables and clauses in the input formula to 3-SAT.*

Using Theorem 25 we show the next result.

Theorem 26. *Unless ETH fails, there is no algorithm for SEOPM running in time $2^{o(n)}$.*

Proof. Let us assume that we have an algorithm $\mathcal{A}$ that solves SEOPM in time $2^{o(n)}$ where n is the number of men/ women. In [100], Kobayashi and Matsui showed that SEOPM is NP-complete, by giving a reduction from SAT to SEOPM. In particular, given a SAT instance with n variables and m clauses, they reduce it to an instance of SEOPM with $2m + 3n$ men (and women). The reduction given by

Kobayashi and Matsui [100] to reduce SAT to SEOPM works identically to reduce 3-SAT to SEOPM. It gives rise to an instance with $2m + 3n$ men (and women).

Next, we show how to design an algorithm for 3-SAT running in time $2^{o(n+m)}$ using algorithm $\mathcal{A}$. Given an instance ϕ of 3-SAT, we start by applying the polynomial time reduction given in [100] and obtain an instance of SEOPM with $2m + 3n$ men and $2m + 3n$ women. Now we solve this instance of SEOPM using algorithm $\mathcal{A}$ in time $2^{o(m+n)}$. Using the solution to an instance of SEOPM we decide in polynomial time whether ϕ is satisfiable or not. Thus, we have given an algorithm for 3-SAT running in time $2^{o(n+m)}$, contradicting Theorem [25]. This concludes the proof.

◇

8.5 Stable extension when lists may not be complete

In this section we will consider an extension of SEOPM. As before, we are given a family of preference lists, $\mathcal{L}^M$ and $\mathcal{L}^W$, for the vertices in M and W respectively. However, unlike SEOPM, for each $m \in M$ the preference list $P(m)$ is a total order on a *subset* of W . Similarly, for each $w \in W$ the preference list $P(w)$ is a total order on a *subset* of M . We assume that woman w is in $P(m)$ for man m if and only if m is in $P(w)$. We will denote the subset of men who have woman w in their preference list by $\delta(w)$. This setup is commonly referred as STABLE MATCHING WITH INCOMPLETE LISTS or SMI.

STABLE EXTENSION OF PARTIAL MATCHING WITH INCOMPLETE LISTS (SEOPMI)

Input: A set of (incomplete) preference lists $\mathcal{L}^M$ of men M over women W , and a partial matching μ' on (M, W) .

Task: Does there exist a set of preference lists of women on men $\mathcal{L}^W$ such that $\mu' \subseteq \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$?

We assume that an input instance $(\mathcal{L}^M, \mu')$ of SEOPMI satisfies the criterion of being **individually rational**: if $(m, w) \in \mu'$ then w is in the preference list of m .

Note that the NP-complete problem SEOPM is a special case of SEOPMI. Hence, we can conclude that SEOPMI is NP-complete as well. We design an exact exponential algorithm for the problem STABLE EXTENSION OF PARTIAL MATCHING WITH INCOMPLETE LISTS (SEOPMI) using Algorithm 8.4.2 given for SEOPM. We wish to point out that the **Algorithm Q1** invoked therein on the input $(\mathcal{L}^M, \mu)$ implicitly assumes that μ is a perfect matching and the set of preference lists $\mathcal{L}^W$ constructed by it contain complete lists for every woman. The matching μ obtained in an earlier step of the algorithm is guaranteed to be perfect since it is inferred from a spanning subgraph of a suitable universal suitor graph, ensuring that all women have a matching partner. In our analysis of SEOPMI, we will begin by assuming that there exists a perfect matching μ such that $\mu' \subseteq \mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$. We can relax that condition on the preference lists in the following manner to produce a set of preference lists which satisfy the condition that m is in the preference lists of w if and only w is in the preference list of m . The preference lists $\{P(w) \mid w \in W\} = \mathcal{L}^W$ are constructed from an out-branching in the rooted suitor graph $\bar{G}(\mathcal{L}^M, \mu)$, where μ is a perfect matching. For every female vertex $w \in W$ we define $P(w)$ to be a strict ordering on the set $\delta(w)$, where the most preferred man is $\mu(w)$. In the out-branching, w has a unique in-neighbor, denoted by $\text{prt}(w)$. If $\text{prt}(w) \in M$, then the second most preferred man in $P(w)$ is $\text{prt}(w)$, followed by an arbitrary

total ordering of the set $\delta(w) \setminus \{\mu(m), \text{prt}(w)\}$. If $\text{prt}(w)$ is the root, then $\mu(w)$ is followed by an arbitrary total ordering of the set $\delta(w) \setminus \{\mu(m)\}$. This completes the description of the construction of the preference lists for women with respect to a perfect matching.

We begin our analysis with the following observation.

Observation 8.5.1. *Let $(\mathcal{L}^M, \mu')$ be an instance of SEOPMI on M and W . Then the suitor graph $G(\mathcal{L}^M, \mu')$ and the universal suitor graph $U(\mathcal{L}^M, \mu')$ can be constructed as usual. Furthermore, let μ be a perfect matching between (M, W) such that $(\mathcal{L}^M, \mu)$ satisfies the individually rational criterion. Then, Proposition 8.3.1 and the conclusion of Lemma 8.4.1 hold as follows:*

- *There exists $\mathcal{L}^W$, a set of preference lists for W such that $\text{GS}(\mathcal{L}^M, \mathcal{L}^W) = \mu$ if and only if the rooted suitor graph $\bar{G}(\mathcal{L}^M, \mu)$ has an out-branching.*
- *If $\mu' \subseteq \mu$, then $U(\mathcal{L}^M, \mu)$ is a subgraph of $U(\mathcal{L}^M, \mu')$, and $U(\mathcal{L}^M, \mu)$ is isomorphic to the suitor graph $G(\mathcal{L}^M, \mu)$.*

Proof. The proof of both the items are identical to the ones given for Proposition 8.3.1 in [100, Theorem 2] and Lemma 8.4.1 for complete preference lists, under the following properties: Both $(\mathcal{L}^M, \mu')$ and $(\mathcal{L}^M, \mu)$ satisfy the individually rational criterion; and for any $m \in M$ and $w \in W$, m is in the preference lists of w if and only if m is in the preference list of w . We skip the details of the proofs.

◇

The following result is the incomplete-list version of Lemma 8.4.2.

Lemma 8.5.1. *Let $(\mathcal{L}^M, \mu')$ be an instance of SEOPMI on M and W . There exists a set of preference lists $\mathcal{L}^W$ such that $\mu' \subseteq \mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$ and μ is a perfect matching between (M, W) if and only if Algorithm 8.4.1 returns a graph on the input $(\mathcal{L}^M, \mu')$.*

Proof. Suppose that there exists a set of preference lists $\mathcal{L}^W$ such that $\mu' \subseteq \mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$ and μ is a perfect matching between (M, W) . Therefore, Observation [8.5.1](#) can be invoked. The rest of the proof for the forward direction is identical to the one presented for Lemma [8.4.2](#) which uses Proposition [8.3.1](#) and Lemma [8.4.1](#).

The proof of the converse is identical to the one presented for Lemma [8.4.2](#), where instead of invoking Proposition [8.3.1](#) we invoke Observation [8.5.1](#). The rest of the details are the same.

◇

Corollary 8.5.1. *Let $(\mathcal{L}^M, \mu')$ be an instance of SEOPMI on M and W . If Algorithm [8.4.1](#) returns a graph on the input $(\mathcal{L}^M, \mu')$, then Algorithm [8.4.2](#) on the input $(\mathcal{L}^M, \mu')$ returns $(\mathcal{L}^W, \mu)$ where $\mu' \subseteq \mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$ and μ is a perfect matching.*

Proof. This follows directly from Theorem [8.4.2](#) and our initial discussion on the construction of (possibly incomplete) preference lists in from a perfect matching between (M, W) .

◇

8.5.1 Overview of Algorithm [8.5.1](#)

We design the algorithm following the lead from Observation [8.5.1](#) and Lemma [8.5.1](#). We intend to utilize the tools that we used for developing the algorithm for SEOPM in Section [8.4](#). The main aspect in which SMI differs from SM with complete lists is that some men and women may not get matched by any stable matching. Given an instance $(\mathcal{L}^M, \mu')$ of SEOPMI, we do not know which men and women will be matched by a stable extension of the partial matching μ' . So we enumerate over all possible subsets of men and women who will be matched by a stable extension of

μ' . In the Algorithm [8.5.1](#), we guess the subsets $M' \subseteq M$ and $W' \subseteq W$ who will be matched by the stable extension. Next, we truncate the preference list for a man in M' to contain women from W' who are preferred more by m than any woman in $W \setminus W'$. This set of truncated list is denoted by $\tilde{\mathcal{L}}^{M'}$. The choice of $M' \cup W'$ ensures that the solution (if the guess is correct) is a perfect matching between M' and W' that extends μ' . To find such a solution we invoke Algorithm [8.4.2](#). If $(\tilde{\mathcal{L}}^{M'}, \mu')$ is a YES-instance of SEOPMI, then we obtain a matching $\mu \supseteq \mu'$ and set of preference lists for every woman in $V(\mu) \cap W$. These preference lists are strict ordering of men in $V(\mu) \cap M$. In the next step we modify the preference list of each $w \in V(\mu) \cap W$ by adding the set of men in $\delta(w) \setminus V(\mu)$ to the end of the list in an arbitrary strict order. Next, we define the preference list for each $w \in W \setminus V(\mu)$ to be some arbitrary ordering of the men in $\delta(w)$. This gives the set of preference lists $\mathcal{L}^W$. Finally, we check if the matching μ is stable w.r.t. $(\mathcal{L}^M, \mathcal{L}^W)$. If $(\mathcal{L}^M, \mu)$ is a NO-instance of SEOPMI, then none of our guesses will yield a stable extension; so at the end of our exhaustive search of all possible subsets of M and W we can return that no stable extension exists.

Lemma [8.5.2](#) proves the correctness of Algorithm [8.5.1](#). But before we present the proof, we must discuss a very well known result by the name of Rural Hospital Theorem, attributed to Gale and Sotomayor [\[59\]](#) which states that for any instance of the stable matching problem with strict preference lists that are possibly incomplete, every stable matching matches the exact same set of vertices between (M, W) . We will invoke this property of stable matchings in the following proof.

Lemma 8.5.2. *Let $(\mathcal{L}^M, \mu')$ be the input to SEOPMI. Then $(\mathcal{L}^M, \mu')$ is a YES-instance of SEOPMI if and only if Algorithm [8.5.1](#) returns $(\mathcal{L}^W, \mu)$.*

Proof. ($\Rightarrow$) Suppose $(\mathcal{L}^M, \mu')$ is a YES-instance of SEOPMI. That is, there exists $\mathcal{L}^W$ such that $\mu' \subseteq \mu'' = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$. We will prove that Algorithm [8.5.1](#) will output a pair $(\mathcal{L}^W, \mu)$ such that $\mu' \subseteq \mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$.

Consider the iteration of Algorithm [8.5.1](#) when $M' = M \cap V(\mu'')$ and $W' = W \cap V(\mu'')$. In other words, μ'' is a perfect matching between (M', W') .

Claim 3. Algorithm [8.4.1](#) on the input $(\tilde{\mathcal{L}}^{M'}, \mu')$ will return a graph.

Proof. We will prove that there exists $\tilde{\mathcal{K}}^{W'}$ such that $\mu' \subseteq \text{GS}(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{K}}^{W'})$. That will allow us to invoke Lemma [8.5.1](#) on the instance $(\tilde{\mathcal{L}}^{M'}, \mu')$ to conclude that Algorithm [8.4.1](#) on the input $(\tilde{\mathcal{L}}^{M'}, \mu')$ will return a graph.

We begin the proof by observing that μ'' matches every vertex in M' . From the preference lists $\mathcal{K}^W = \{Q(w) \mid w \in W\}$, we construct the set $\tilde{\mathcal{K}}^{W'} = \{Q(w) \cap M' \mid w \in W'\}$. We define $\mu^* = \text{GS}(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{K}}^{W'})$, i.e, μ^* is the man-optimal stable matching w.r.t. $(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{K}}^{W'})$. We will prove that $\mu' \subseteq \mu^*$.

It is easy to see that μ'' is stable w.r.t. $(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{K}}^{W'})$. By definition, μ'' is perfect matching between (M', W') . Therefore, (by Rural Hospital Theorem) every matching stable w.r.t. $(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{K}}^{W'})$ must also be perfect. In particular, $\mu^* = \text{GS}(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{K}}^{W'})$ is a perfect matching between (M', W') . Next we will prove that μ^* is stable w.r.t. $(\mathcal{L}^M, \mathcal{K}^W)$. Once that is proved, we can deduce that for all $m \in M'$, $\mu''(m) \geq_m \mu^*(m)$ in preferences in $\mathcal{L}^M$ because μ'' is the man-optimal stable matching w.r.t. $(\mathcal{L}^M, \mathcal{K}^W)$. Additionally, for all $m \in M'$, we know that $\mu^*(m) \geq_m \mu''(m)$ in preferences in $\tilde{\mathcal{L}}^{M'}$ because μ^* is the man-optimal stable matching w.r.t. $(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{K}}^{W'})$, and μ'' is stable w.r.t. $(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{K}}^{W'})$. Together these two conditions imply that $\mu''(m) = \mu^*(m)$ for all $m \in M'$; and so $\mu'' = \mu^*$ follows as a result.

Now we will prove that μ^* is stable w.r.t. $(\mathcal{L}^M, \mathcal{K}^W)$. Suppose that for the sake of contradiction, μ^* is not stable w.r.t. $(\mathcal{L}^M, \mathcal{K}^W)$, and (m_b, w_b) denotes a blocking pair. Then, $w_b >_{m_b} \mu^*(m_b)$ and $m_b >_{w_b} \mu^*(w_b)$ in the preference list in $\mathcal{L}^M$ and $\mathcal{K}^W$, respectively. By definition, μ^* is the man-optimal stable matching w.r.t. $(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{K}}^{W'})$, and μ'' is stable w.r.t. $(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{K}}^{W'})$, thus $\mu^*(m_b) \geq_{m_b} \mu''(m_b)$ in the preference list in $\tilde{\mathcal{L}}^{M'}$. Note that m_b 's preference lists in $\tilde{\mathcal{L}}^{M'}$ is identical to that in $\mathcal{L}^M$ at least up to

the appearance of $\mu''(m_b)$. Thus, we can conclude that $w_b >_{m_b} \mu^*(m_b) \geq_{m_b} \mu''(m_b)$ in both $\mathcal{L}^M$ and $\tilde{\mathcal{L}}^{M'}$. But then $\mu''(w_b) >_{w_b} m_b >_{w_b} \mu^*(w_b)$ in the preference list in $\mathcal{K}^W$, otherwise (m_b, w_b) is a blocking pair in μ'' w.r.t. $(\mathcal{L}^M, \mathcal{K}^W)$, a contradiction. Since the preference lists in $\mathcal{K}^W$ and $\tilde{\mathcal{K}}^{W'}$ (by our definition) have the same relative order for the men in M' , we can conclude that $\mu''(w_b) >_{w_b} m_b >_{w_b} \mu^*(w_b)$ holds for w_b 's preference list $\tilde{\mathcal{K}}^{W'}$, as well. Recall that earlier we had proved that $w_b >_{m_b} \mu^*(m_b) \geq_{m_b} \mu''(m_b)$ in $\tilde{\mathcal{L}}^{M'}$. But then, (m_b, w_b) is a blocking pair of μ^* w.r.t. $(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{K}}^{W'})$, a contradiction. Therefore, we can conclude that μ^* is stable w.r.t. $(\mathcal{L}^M, \mathcal{K}^W)$.

As discussed earlier, this completes the proof that $\mu'' = \mu^*$. Therefore we have shown that $\mu' \subseteq \mu^* = \text{GS}(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{K}}^{W'})$, where μ^* is a perfect matching between (M', W') . Lemma 8.5.1 applied to the instance $(\tilde{\mathcal{L}}^{M'}, \mu')$ completes the proof of the claim.

◇

By Corollary 8.5.1, Algorithm 8.4.2 on the input $(\tilde{\mathcal{L}}^{M'}, \mu')$ will return a set of preference lists, and a stable extension of μ' that is a perfect matching between (M', W') . We denote it by $(\tilde{\mathcal{L}}^{W'}, \mu)$, and note that $\mu = \text{GS}(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{L}}^{W'})$. In the next claim we show that μ is stable w.r.t. $(\mathcal{L}^M, \mathcal{L}^W)$, where $\mathcal{L}^W$ is constructed from $\tilde{\mathcal{L}}^{W'}$ as described in (the last forall-loop in) Algorithm 8.5.1.

Before we present the proof of the next claim, we wish to emphasize an important property about the preference lists in $\tilde{\mathcal{L}}^{W'}$ and $\mathcal{L}^W$ that is used in the upcoming proof. For any $w \in W'$, man $\mu(w)$ appears at the front of w 's preference list in $\tilde{\mathcal{L}}^{W'}$. This is due to the property of **Algorithm Q1** in [100], used by our Algorithm 8.4.2 as a subroutine to compute the necessary preference lists. Considering the construction of the preference lists in $\mathcal{L}^W$ (by our algorithm), it holds that for every $w \in W'$, the same property holds for w 's preference lists in $\mathcal{L}^W$. Now we are ready to present the proof.

Claim 4. Matching $\mu = \text{GS}(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{L}}^{W'})$ is stable w.r.t. $(\mathcal{L}^M, \mathcal{L}^W)$.

Proof. Suppose that (m_b, w_b) is a blocking pair of μ w.r.t. $(\mathcal{L}^M, \mathcal{L}^W)$. If $w_b \in W'$, then $\mu(w_b)$ appears at the first position in w_b 's preference list in $\mathcal{L}^W$. Therefore, if $w_b \in W'$, then (m_b, w_b) cannot block μ^* w.r.t. $(\mathcal{L}^M, \mathcal{L}^W)$. Suppose that $w_b \in W \setminus W'$ and $m_b \in M'$, implying that w_b is unmatched in μ and $w_b >_{m_b} \mu(m_b)$ in m_b 's preference list in $\mathcal{L}^M$. The preference list of m_b in $\tilde{\mathcal{L}}^{M'}$ is the prefix of the preference list of m_b in $\mathcal{L}^M$ up to (but not including) the first woman from $W \setminus W'$. Therefore, $\mu(m_b)$ does not even exist in m_b 's preference list in $\tilde{\mathcal{L}}^{M'}$. This contradicts definition that $\mu = \text{GS}(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{L}}^{W'})$. This leaves the case that $m_b \in M \setminus M'$ and $w_b \in W \setminus W'$, yet, w_b is in m_b 's preference list in $\mathcal{L}^M$ and so $m_b \in \delta(w_b)$. This is a contradiction to the choice of M' and W' , in our algorithm. Hence, we can conclude that (m_b, w_b) cannot block μ w.r.t. $(\mathcal{L}^M, \mathcal{L}^W)$. This completes the proof that μ is stable w.r.t. $(\mathcal{L}^M, \mathcal{L}^W)$.

◇

Thus, we can conclude that Algorithm 8.5.1 on the input $(\mathcal{L}^M, \mu')$ returns the output $(\mathcal{L}^W, \mu)$ such that $\mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$; ($\Rightarrow$) is proved.

($\Leftarrow$) Suppose that $(\mathcal{L}^W, \mu)$ is the output of Algorithm 8.5.1. Then there exists $M' \supseteq V(\mu') \cap M$ such that Algorithm 8.4.2 on the input $(\tilde{\mathcal{L}}^{M'}, \mu')$ returns $(\tilde{\mathcal{L}}^{W'}, \mu)$. Thus, implying that $\mu' \subseteq \mu = \text{GS}(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{L}}^{W'})$, and μ is also stable w.r.t. $(\mathcal{L}^M, \mathcal{L}^W)$. We will prove that in fact, μ is the man-optimal stable matching w.r.t. $(\mathcal{L}^M, \mathcal{L}^W)$, and thereby direction ($\Leftarrow$) will be proved.

Claim 5. Matching $\mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$

Proof. Note that $V(\mu) = M' \cup W'$. Suppose that $\mu^* = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$ and $\mu^* \neq \mu$. Since both μ and μ^* are stable w.r.t. $(\mathcal{L}^M, \mathcal{L}^W)$, (by Rural Hospital Theorem) μ^* must be a perfect matching between (M', W') , i.e. $V(\mu^*) = M' \cup W'$.

Now, since $\mu \neq \mu^*$, there exists $m \in M'$ such that $\mu(m) \neq \mu^*(m)$. Since μ^*

is the men-optimal matching, hence $\mu^*(m) >_m \mu(m)$ in m 's preference list in $\mathcal{L}^M$. Every preference list in $\tilde{\mathcal{L}}^{M'}$ is a restriction of the corresponding man's list in $\mathcal{L}^M$ up to the appearance of the first woman in $W \setminus W'$. Specifically, this implies that $\mu^*(m) >_m \mu(m)$ in $\tilde{\mathcal{L}}^{M'}$ as well. But this contradicts the property that μ is the man-optimal stable matching w.r.t. $(\tilde{\mathcal{L}}^{M'}, \tilde{\mathcal{L}}^{W'})$, and so $\mu(m) >_m \mu^*(m)$ in m 's preference list in $\tilde{\mathcal{L}}^{M'}$. Thus, we can conclude that $\mu = \mu^* = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$.

◇

This concludes the proof of the lemma.

◇

Lemma 8.5.3. *Algorithm [8.5.1](#) solves SEOPMI in $2^{\mathcal{O}(n)}$ time.*

Proof. The correctness of Algorithm [8.5.1](#) is given by Lemma [8.5.2](#). We invoke the Algorithm [8.4.2](#) for each subset of M and W . From Theorem [24](#), we know that each invocation of Algorithm [8.4.2](#) takes $2^{\mathcal{O}(n)}$ time. All other computations can be done in time $\mathcal{O}(n^2)$. So in total Algorithm [8.5.1](#) takes $2^{\mathcal{O}(n)} \times 2^n \times 2^n = 2^{\mathcal{O}(n)}$ time.

◇

8.6 Concluding thoughts

In this chapter we designed an exact algorithm for STABLE EXTENSION OF PARTIAL MATCHING running in time $2^{\mathcal{O}(n)}$. We complemented this result by showing that unless ETH fails the running time bound is asymptotically optimal. There are several problems in the stable matching model that are NP-complete and have been studied from the perspective of approximation algorithms. However, there is almost no study about these problems either from the view point of moderately exponential time

algorithms or parameterized complexity. The area needs a thorough study in these algorithmic paradigms and is waiting to be explored.

Algorithm 8.5.1: Solves SEOPMI

Input: A set of men and women vertices (M, W) , preferences of men $\mathcal{L}^M$, and a partial matching μ' between (M, W)

```
forall  $w \in W$  do
  |  $\delta(w) = \{m \mid m \text{ has } w \text{ in his preference list in } \mathcal{L}^M\}$ 
end
forall  $M' \subseteq M$  and  $W' \subseteq W$  such that  $V(\mu') \cap M \subset M'$  and
 $V(\mu') \cap W \subset W'$  do
  ▷ Guessing  $(M', W')$  s.t.  $(\tilde{\mathcal{L}}^{M'}, \mu')$  is an instance of SEOPM, where  $\tilde{\mathcal{L}}^{M'}$ 
  is an appropriate restriction of  $\mathcal{L}^M$  on  $(M', W')$ .
  if  $m \in M \setminus M'$  and  $w \in W \setminus W'$  such that  $m \in \delta(w)$  then
    | break
    ▷ Guess the subsets again
  end
  Let  $\tilde{\mathcal{L}}^{M'} \leftarrow \emptyset$  and  $\mathcal{L}^W \leftarrow \emptyset$ 
  ▷ Initializing empty sets
  forall  $m \in M'$  do
    |  $P(m) \leftarrow$ 
    |    $[m\text{'s preference list up to (not including) the first } w \in W \setminus W']$ .
    |  $\tilde{\mathcal{L}}^{M'} \leftarrow \tilde{\mathcal{L}}^{M'} \cup \{P(m)\}$ 
    |   ▷ Lists are restricted to women in  $W'$ .
  end
  if Algorithm 8.4.2 on input  $(\tilde{\mathcal{L}}^{M'}, \mu')$  returns  $(\tilde{\mathcal{L}}^{W'}, \mu)$  then
    forall  $w \in W$  do
      if  $w \in W'$  then
        |  $P(w) \leftarrow [\text{Preference list of } w \text{ in } \tilde{\mathcal{L}}^{W'}], [\delta(w) \setminus M']$ 
      end
      else
        |  $P(w) \leftarrow [\delta(w) \text{ in any arbitrary order}]$ 
      end
       $\mathcal{L}^W \leftarrow \mathcal{L}^W \cup \{P(w)\}$ 
    end
  end
  if  $\mu$  is stable w.r.t.  $(\mathcal{L}^M, \mathcal{L}^W)$  then
    | return  $(\mathcal{L}^W, \mu)$ 
  end
end
return "No stable extension exists"
```

Chapter 9

Tournament Fixing Problem

In the previous chapter we gave an algorithm to manipulate an instance of a STABLE MATCHING. We continue our exploration in manipulation problems in the realm on computational social choice. Specifically, we give algorithms for rigging a knockout tournament.

A knockout tournament is a standard format of competition, consisting of several rounds. In each round, all players that have not yet been eliminated are paired up into matches. Losers are eliminated, and winners are raised to the next round, until only a single winner exists. For an illustrative example, consider Wimbledon Men's tennis tournament (having 128 players). In fact, knockout tournament is the most widely-used format of competition in sports [75, 27, 67]. Apart from sports and popular culture, knockout tournaments are also prevalent in elimination-based election and decision making schemes. Specifically, they received notable attention from the viewpoints of artificial intelligence [151, 155, 147, 146, 145, 9, 96, 97, 135] as well as economics and operation research [138, 150, 104].

Having a favorite player v^* in mind, is there a way to conduct the competition so that v^* wins? To formulate this question formally, note that the execution of a knockout tournament with a set N of N players is governed by a complete (unordered)

binary tree T with n leaves and a mapping $\varphi : N \rightarrow \text{leaves}(T)$, called a *seeding*. In the first round, every two players mapped to leaves with the same parent compete against each other, and the winner is mapped to the common parent. The leaves are then deleted from the tree, and the next round is conducted similarly. The execution stops when the tree contains a single vertex, mapped to a player who is declared the winner. The question of making v^* a winner is sensible when we have predictive information about outcomes of matches. Specifically, we assume we have a tournament $D = (V, A)$ (not to be confused with the competition itself that is also termed a tournament), which is a digraph where either $(u, v) \in A$ or $(v, u) \in A$ for all $u, v \in V$. This encodes predictive information as follows : $V = N$, and for every $u, v \in N$, we predict that in a match between u and v , u would beat v if and only if $(u, v) \in A$.

Now, our question is formalized as follows.

TOURNAMENT FIXING PROBLEM (TFP)

Input: A tournament $D = (V, A)$, and a vertex $v^* \in V$.

Parameter: The feedback arc set number k^* of D .

Task: Is there a seeding of the $n = |V|$ players such that v^* wins the resulting knockout tournament?

The problem of rigging a knockout tournament was introduced by Vu et al. [151]. This work led to a flurry of research of structural properties of D that guarantee that v^* wins [96, 97, 147, 146, 145, 9, 96, 155]. Additional information can be found in surveys such as [156]. The question of whether TFP is NP-hard was posed in several works, including [151, 155, 140, 146, 147, 145, 103]. It was finally resolved in the affirmative by Aziz et al. [9], showing that it can be solved in time $\mathcal{O}(2.83^n)$. Later, Kim and Williams [97] gave an $\mathcal{O}(2^n)$ time and space algorithm. Even for a small number of players, such a running time is prohibitive. Moreover, approximation of

TFP does not make sense.

In light of the discussion above, a most natural framework to study TFP is parameterized complexity. Specifically, Aziz et al. [9] considered the feedback arc set number of D as the parameter k^* . This parameter has a natural interpretation. It is likely that in real world scenarios, there is an approximate ranking of players' strengths (e.g., Wimbledon), so that a player of a certain rank is expected to beat players of a lower rank. The number of matches where we guess (before the competition begins) that a player of a certain rank will beat a player of higher rank, which upper bounds k^* , is significantly smaller than n .

Aziz et al. [9] showed that TFP is XP, which means solvability in time $n^{f(k^*)}$ for some function f of k^* . They gave a clever dynamic programming algorithm that runs in time $n^{\mathcal{O}(k^*)}$. Assuming that k^* is not fixed (i.e., independent of n), such a running time is again prohibitive. The question of whether TFP is fixed-parameter tractable with respect to k^* , that is, solvable in time $f(k^*)n^{\mathcal{O}(1)}$ for some function f , was later resolved by Ramanujan and Szeider [135]. They showed that TFP is solvable in time $2^{\mathcal{O}(k^{*2} \log k^*)} n^{\mathcal{O}(1)}$. This means that whenever $k^{*2} \log k^* = \mathcal{O}(\log n)$, TFP is solvable in polynomial time.

In addition, TFP received significant attention from a structural point of view—an array of works exhibited structural properties of D that guarantee that v^* wins [96, 97, 147, 146, 155]. Let us review some of the results that are closer to our study (with emphasis on non-probabilistic structural statements), and refer the reader to the survey [156] for more information. Vassilevska Williams [155] studied the case where v^* is a *king* (resp. *super king*), that is, every player is either beaten by v^* or beaten by a player that is beaten by v^* (resp. $\log n$ players that are beaten by v^*). In particular, she showed that for a king v^* that beats at least half the players, there always exists a fixing that makes v^* the winner, while for a king v^* that beats less than half the players, there *might* not exist a fixing that makes v^* the winner. In

addition, she showed that if v^* is a *super king*, then there exists a fixing that makes v^* win.

Later, Stanton and Williams [146] strengthened the work by Vassilevska Williams [155] with respect to her results for a probabilistic model, and by extending her condition that ensures that a king can always win. In a different work, Stanton and Williams [147] exhibited sufficient and necessary conditions that ensure that for *any* player v^* among the K players that have the largest out-degrees in D (for certain choices of K), there exists a fixing that makes v^* the winner. More recently, Kim and Williams [97] proved that TFP is NP-hard even if v^* is a king that beats $n/4$ players, or if v^* is a so called *3-king* (that is not as strong as a king) that beats half the players. They also provided a sufficient condition to ensure that a 3-king can win the tournament. Later, Kim et al. [96] further extended and generalized previous structural results, including the study of the power of 3-kings. We remark that a certain argument in one of our proofs may be viewed in the terms of a king (see Section 9.6), but apart from this curiosity, the notions we introduce are incomparable to those of kings and of the K best players by [147] (see Section 9.5).

In addition, several studies present empirical results. For example, Mattei and Walsh [120] showed that although TFP is NP-hard, real world soccer and tennis instances (English Premier League, the German Bundesliga, and the ATP World Tour) are easily solvable. Indeed, real world instances are generally structured, hinting at the relevance of appropriate parameterizations to capture these structures. Another empirical study on TFP was conducted by Russell and Beek [140], which focused on restrictions on the space of possible seedings. Here, the motivation is that real cup competitions often place restrictions to ensure fairness and wide geographic interest, which make the problem harder to solve in practice.

Kim and Williams [97] introduced a more general question, called BRIBERY TF (BTF). Given N , D , $v^* \in N$ and $\ell \in \mathbb{N} \cup \{0\}$, BTF asks if it is possible to reverse at

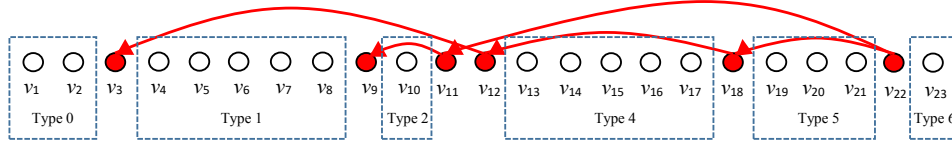


Figure 9.1: Partition of $V(D) \setminus V(K)$ into types. The arcs in K and the vertices in $V(K)$ are colored red. For all v_i, v_j with $i < j$ such that $(v_j, v_i) \notin K$, we suppose that $(v_i, v_j) \in A(D)$ (not displayed).

most ℓ arcs in D (that is, change the outcome of at most ℓ matches) so that afterwards there is a way to fix the tournament that makes v^* wins. TFP is the special case of BTF where $\ell = 0$, which implies that BTF is NP-hard. Kim and Williams [97] proved that for any fixed $\epsilon > 0$, BTF is NP-hard even if $\ell < (1 - \epsilon) \log_2 n$. On the other hand, they observed that if $\ell \geq \log_2 n$, then the input is a YES-instance. The problem is formally defined as follows.

BRIBERY TF(BTF)

Input: A tournament $D = (V, A)$, a vertex $v^* \in V$, and $\ell \in \mathbb{N} \cup \{0\}$.

Parameter: The feedback arc set number k^* of D .

Task: Is it possible to reverse at most ℓ arcs in D so that afterwards there is a seeding of the $n = |V|$ players such that v^* wins the resulting knockout tournament?

Prior to [97], bribery has already been studied with respect to tournaments whose seeding is fixed in advance. The computational complexity of this problem is significantly simpler than that of TFP. In particular, Russell and Walsh [141] showed that this problem is solvable in polynomial time. In fact, this result was proved in a more general setting where we have a coalition (a set of players) such that all manipulations must involve a lose of one of its members. Additionally, Mattei et al. [119] showed that even in a probabilistic model, where we want to make v^* have a nonzero probability of winning, the problem is still solvable in polynomial time.

9.1 Our Contribution

The main component of the algorithm by Ramanujan and Szeider [135] is the translation of TFP into an algebraic system of equations, solved in a black box fashion by an ILP solver. In particular, the algorithm (and not just the proof of correctness) by Ramanujan and Szeider [135] is thus both complicated and not self-contained. We present a fresh greedy solution whose correctness relies on new insights into TFP itself. The advantages of our algorithm are as follows.

First, our algorithm is *purely combinatorial* (unlike the algebraic algorithm of [135]). While the analysis of our algorithm is intricate, the algorithm itself is *simple* and easily implementable. Essentially, after we “guess” a template tree (similarly to [135]), our algorithm merely iterates over the paths and subtrees that are not already determined by the guess, and fills them up in a greedy manner. We find it both surprising and pleasing that such a strategy just works. From our proof of correctness, the reasons for the validity of this strategy are revealed, but at first glance, we do not find them apparent.

Second, our solution is completely *self-contained*. We do not invoke any black-box ([135] invokes a big hammer, i.e., an ILP solver).

Third, our algorithm is substantially faster than that of [135]. Specifically, our running time bound is $2^{\mathcal{O}(k^* \log k^*)} n^{\mathcal{O}(1)}$ rather than $2^{\mathcal{O}(k^{*2} \log k^*)} n^{\mathcal{O}(1)}$. The difference is clear already for values as small as $k^* = 8$, where $2^{k^* \log_2 k^*} = 2^{24}$ while $2^{k^{*2} \log_2 k^*} = 2^{192}$. Even for $k^* = 4$, $2^{k^* \log_2 k^*} = 2^{32} \gg 2^{24}$. Thus, it is very plausible that our improvement constitutes the difference between being practically infeasible and feasible for social choice applications, where n typically ranges between a few tens to a few hundreds, if k^* is presumed to be, say, 5% of n . In a different view, our result also proves that whenever $k^* \log k^* = \mathcal{O}(\log n)$, TFP is solvable in polynomial time.

Our contribution in the study of BTF is fourfold. First, we present two

“obfuscation operations”, whose input is a solution, i.e. a set of at most ℓ arcs to reverse and a fixing of the tournament (with these arcs reversed). If the operation is applicable, it returns (in polynomial time) a different solution. These operations are relevant when: (i) the deceit in the current solution is too conspicuous; (ii) the current solution could not be realized (e.g., some player refused to be bribed). Moreover, we use these two operations as building blocks in our proofs.

Secondly, we present a combinatorial result, which states that there always exists a solution with all reversals incident to v^* and “elite players”. Roughly speaking, given a feedback arc set K of D with $k = |V(K)|$, we partition $V(D) \setminus V(K)$ into $k + 1$ modules. Each of these modules induces a DAG, and an *elite club* roughly refers to a set of players of “highest rank” in the topological order of each of these DAGs (the formal definition is more general and robust, see Definition [9.5.1](#)). Our definition of an elite club is incomparable to the notions discussed earlier (see the explanation in Section [9.5](#)).

Third, we give a closed formula (based on our combinatorial result) that resolves the special case of BTF where D is a DAG. More precisely, the formula is satisfied if and only if the input instance of BTF is a YES-instance. The formula can be verified in polynomial time. Moreover, its proof is constructive (the satisfaction of the formula indicates that a certain candidate to be a solution is indeed a solution). In addition, we generalize our formula to characterize all YES-instances also for the general case. Of course, the general characterization cannot be verified in polynomial time.

Finally, we present exact algorithms for the general case of BTF. (Recall that in its full generality, BTF is NP-hard.) Specifically, we develop an algorithm that runs in time $2^n n^{\mathcal{O}(1)}$ and has a polynomial space complexity (which improves upon the algorithm by Kim and Williams [\[97\]](#) in the special case of TF), as well as a parameterized algorithm that runs in time $2^{\mathcal{O}(k^* \log k^*)} n^{\mathcal{O}(1)}$, where k^* is the feedback

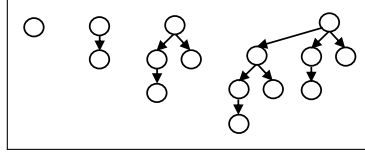


Figure 9.2: Binomial arborescences of sizes $2^0, 2^1, 2^2$ and 2^3 .

arc set number of D . The latter algorithm implies that BTF is FPT parameterized by k^* alone.

9.2 Preliminaries

Given a tournament D , let $V(D)$ and $A(D)$ denote its vertex set and arc set, respectively. For a pair of vertices u, v , by (u, v) we denote an arc from u to v , and say that u is an in-neighbor of v and v is an out-neighbor of u . A uv -path is a path from u to v . Given $X \subseteq V(D)$, by $D[X]$ we denote the subtournament of D induced by X and $N(X)$ is the set of neighbors of the vertices in X outside X . For a rooted tree T and a vertex $v \in V(T)$, by T_v we denote the subtree of T rooted at v . An *arborescence* is a rooted directed tree such that all arcs are directed away from the root.

We now discuss the notion of a binomial arborescence and its relevance to our work (see Fig. 9.2.).

Definition 9.2.1. An unlabeled binomial arborescence (**usba**) T rooted at $v \in V(T)$ is defined recursively as follows:

- A single node v is a binomial arborescence rooted at v .
- Given two vertex disjoint binomial arborescences of equal size, T_v rooted at v and T_u rooted at u , adding an arc from v to u results in a binomial arborescence T rooted at v .

Let D be a directed graph. If T is a subgraph of D with $V(T) = V(D)$, then T is a

labeled spanning binomial arborescence (**sba**) of D .

Given an integer $n = 2^j$, there exists a unique **usba**, denoted by B_n , on n vertices. For our purpose, we *fix a labelling* of the vertices of B_n from $[n]$, that is, a function $\gamma : V(B_n) \rightarrow [n]$. Whenever we refer to an **usba** on n vertices, we always mean B_n , where the vertices are labeled by γ . Our interest in **sba** is due to the following connection between **sba** and finding a seeding of the vertices in D that results in a specific vertex being the winner.

Proposition 9.2.1 ([155]). *Let D be a tournament with $v^* \in V(D)$. There is a seeding of the vertices in D such that the resulting knockout tournament is won by v^* if and only if D has a **sba** rooted at v^* .*

Proposition 9.2.1 reduces TFP to the problem of finding a **sba** rooted at v^* . BTF is equivalent to the following question: Is there $R \subseteq A(D)$ of size at most ℓ such that D^R has a spanning binomial arborescence rooted at v^* ? Unless written otherwise, we take this point of view. In this context, the following lemma will be useful.

Lemma 9.2.1. *Let D be a tournament, $X \subseteq V(D)$ such that $D[X]$ is acyclic, and T be arborescence on $|X|$ vertices. Then, $D[X]$ contains (as a subgraph) an arborescence isomorphic to T that can be computed in polynomial time.*

Proof. Observe that since $D[X]$ is acyclic tournament, there is an unique topological ordering σ on X . The first vertex v in σ beats every other vertex in X . The second vertex in σ beats every other vertex other than v , and so on. We can construct an arborescence isomorphic to T in top down fashion. Assign the first vertex v in σ as the root of T , and delete v from σ . We recursively continue the process for each child of v in T . For the correctness of the first step, note that since v beats every vertex in σ , v beats the vertices that are assigned as it's children in the following

step. This invariant is maintained in each step of the assignment. Hence, we produce an arborescence isomorphic to T . $\diamond$

Next, we give some simple properties of **sba**.

Observation 9.2.1 ([135]). *Let T be a binomial arborescence on $n \geq 2$ vertices. Then, for every vertex $v \in V(T)$, it holds that T_v is a spanning binomial arborescence of $D[V(T_v)]$ and in particular $|V(T_v)|$ is a power of 2.*

*Let $\mathbf{B}$ be an **sba** on n vertices. Then, the height of $\mathbf{B}$ (the maximum length of a path from root to leaf) is $\log n$.*

Moreover, the root of T has exactly $\log n$ children, $v_1, v_2, \dots, v_{\log n}$, with $|V(T_{v_i})| = 2^{i-1}$ for all $i \in \{1, 2, \dots, \log n\}$.

9.3 Greedy Algorithm for TFP

As stated earlier, due to Proposition 9.2.1, we study the following equivalent question: *Does there exist an **sba** rooted at v^* in D ?* We start by computing a minimum sized feedback arc set, F , of D . It is well known that a tournament has a directed cycle if and only if it has a directed triangle. Furthermore, a subset F of $A(D)$ is a minimal *feedback arc set* if and only if the tournament D^{rev} obtained after reversing the arcs in F is acyclic. Using these two facts, one can design a trivial branching algorithm for finding F running in time $3^{k^*} n^{\mathcal{O}(1)}$. That is, while there exists a triangle in the given tournament, find one and recursively try to find a smaller sized solution by reversing an arc of the triangle. See [29] for further details. In fact one can find a minimum sized feedback arc set, F , of D in time $2^{\mathcal{O}(\sqrt{k^*})} n^{\mathcal{O}(1)}$ [47, 93]. From now onwards, we assume that we have a feedback arc set F at hand.

Let σ be the *topological ordering* of D after reversing the arcs in the feedback arc set F . Throughout the chapter, we always work with the topological ordering σ .

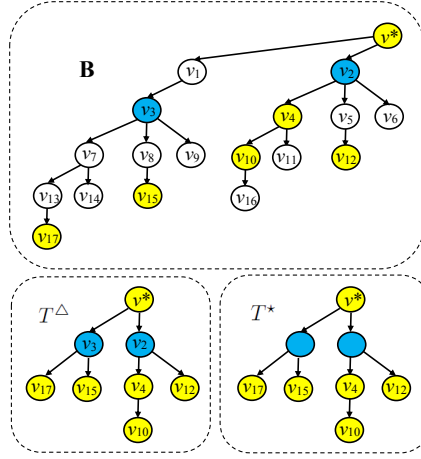


Figure 9.3: The topology T^Δ of $\{v^*, v_4, v_{10}, v_{12}, v_{15}, v_{17}\}$ in B (left), and the template T^* (right). LCA-vertices are colored blue.

If a vertex v precedes u in σ , then we write $v \prec_\sigma u$. Given $V' \subseteq V(D)$, the *highest vertex in V'* is the vertex that precedes all other vertices of V' in the ordering σ . Furthermore, for any arc (u, v) , we say that u *beats* v (equivalently, v is *beaten* by u). We now give a few basic definitions central to our algorithm.

Definition 9.3.1. (LCA closure) For a rooted tree T and a vertex subset $M \subseteq V(T)$, the least common ancestor-closure (LCA-closure) $\text{LCA}(M)$ is obtained by the following process. Initially, set $M' = M$. Then, as long as there are vertices x and y in M' whose least common ancestor w in T is not in M' , add w to M' . When the process terminates, output M' as the LCA-closure of M .

The following folklore lemma (see, e.g., [29]) summarizes two basic properties of LCA closures.

Lemma 9.3.1. Let T be a rooted tree, $M \subseteq V(T)$ and $M' = \text{LCA}(M)$. Then $|M'| \leq 2|M|$ and for every connected component C of $T \setminus M'$, $|N(C)| \leq 2$.

Our algorithm will “guess” some information about a solution (if one exists), and then greedily complete the parts unknown. To discuss these guesses (Phase 1), we require the following definition (see Fig. 9.3).

Definition 9.3.2. (Topology and Template) Let $\mathbf{B}$ be an **sba** rooted at v^* and let $X \subseteq V(\mathbf{B})$. We construct a tree T^Δ from $\mathbf{B}$ as follows. First delete all vertices that do not lie on any v^*u -path in $\mathbf{B}$ for any $u \in \text{LCA}(X)$. Now, as long as there is a vertex $u \in V(\mathbf{B}) \setminus \text{LCA}(X)$ with exactly one in-neighbor u^- and exactly one out-neighbor u^+ , delete the vertex u and add the arc (u^-, u^+) . Then T^Δ is a tree obtained at the end of this process, it is called the *topology of X in $\mathbf{B}$* . The *template tree of X in $\mathbf{B}$* is a tree obtained from T^Δ by deleting the labels of all vertices in $\text{LCA}(X) \setminus X$. A tree T^* is a *template of X* if it is a template of X in at least one **sba** $\mathbf{B}$ rooted at v^* .

Before going further, we recall some notation.

Let F denote the feedback arc set and V_F denote the set of vertices that are adjacent to arcs in F . We denote $k = |V_F|$. Observe that $k \leq 2k^*$. Furthermore, σ denote the *topological ordering* of D after reversing the arcs in F .

For the phases where we complete missing information, the following definition will be crucial.

Definition 9.3.3. (Fas-vertex, Fas-parent and Fas-child) Let T^* be a template of $V_F \cup \{v^*\}$. Vertices in $V_F \cup \{v^*\}$ are called *fas-vertex*. Let $v \in V(D)$. The *fas-parent* of v is the last *fas-vertex* w not equal to v on the v^*v -path in T^* . An *fas-child* of v is an *fas-vertex* w , that is the first *fas-vertex* not equal to v on the vw path in T^* . The set of *fas-children* of v in T^* is denoted by $\text{faschild}(v)$.

As a first step, our algorithm would need to enumerate all templates of $V_F \cup \{v^*\}$. Due to Lemma 9.3.1, a template of $V_F \cup \{v^*\}$ in any **sba** is a rooted tree T on at most $2k + 1$ vertices with some of its vertices labeled with the vertices in $V_F \cup \{v^*\}$ and the others unlabeled, beings *placeholder for LCA vertices*. We can therefore enumerate a superset of templates by enumerating every rooted tree on at most $2k + 1$ vertices with a surjective mapping of *some* of its vertices to vertices in $V_F \cup \{v^*\}$

such that the root is mapped to v^* , and all the leaves are mapped as well (to vertices in $V_F \cup \{v^*\}$):

Observation 9.3.1. *A superset of the set of templates of $V_F \cup \{v^*\}$ of size $k^{\mathcal{O}(k)}$ can be enumerated in time $k^{\mathcal{O}(k)}$.*

Intuition. Our algorithm proceeds in four steps. First, we guess a partial structure of the **sba**. Second, we verify that the guesses made in the previous step are “realizable”. These two steps are straightforward, and are also present in a similar form in [135]. Then, we turn to the heart of our algorithm. Here, we design a “greedy” procedure to complete the partial **sba** in two steps. The first fills up paths corresponding to edges of the template at hand. Roughly speaking, here we always select a yet “unhandled” fas-vertex v that is highest in $V(D)$ with respect to σ . Then, we greedily fill the path between v and its fas-parent using the highest “available” vertices that are “between” v and its fas-parent (i.e., beat v and beaten by the parent). This is a slight simplification since when we look at the path between v and its fas-parent, a part of it (closer to the parent) is already filled. However, by considering lca vertices this technicality is easily handled. The second phase fills-up the subtrees hanging from the paths we have just filled. Roughly speaking, while we still have an unfilled subtree, we pick one that hangs from a vertex v that is highest. We greedily fill the subtree with “available” vertices that are the highest ones beaten by v . If the greedy procedure ever gets stuck, we move to the next partial **sba**. While the procedure itself is very simple, correctness is surprising, and indeed the proof is quite intricate.

9.3.1 Phase I: Guessing

We first intuitively explain what we intend to achieve by guessing. Suppose $I = (D, v^*, k)$ is a YES-instance of TFP. Then, there exists an **sba** $\mathbf{B}$ rooted at v^* . We

first guess the template $T^\star$ of $V_F \cup \{v^\star\}$ in $\mathbf{B}$. Given $T^\star$, for each arc (u, v) , we guess a length, denoted by $\text{len}(u, v)$, that is *the number of internal vertices on the unique uv -path in $\mathbf{B}$* . Then, for each vertex $v \in V_F$, we guess the size of the subtree rooted at v in $\mathbf{B}$, denoted by $\text{siz}(v)$. Due to Observation 9.2.1, $\text{len}(u, v) \in \{0, \dots, \log n\}$ and $\text{siz}(v) \in \{2^0, 2^1, \dots, 2^{\log n}\}$. Formally, in this phase we guess the following.

1. Using Proposition 9.3.1, we iterate over all templates of $V_F \cup \{v^\star\}$. Let $T^\star$ denote one such template.
2. For each arc (u, v) in $A(T^\star)$, guess an integer in $\{0, \dots, \log n\}$. Formally, we enumerate over all possible “length functions” $\text{len} : A(T^\star) \rightarrow \{0, \dots, \log n\}$.
3. Next, we guess the “size function”. That is, we enumerate all possible functions $\text{siz} : V_F \rightarrow \{2^0, \dots, 2^{\log n}\}$.

Definition 9.3.4. *An sba $\mathbf{B}$ complies with $(T^\star, \text{len}, \text{siz})$ if (i) the template of $V_F \cup \{v^\star\}$ in $\mathbf{B}$ is $T^\star$, (ii) for each arc $(u, v) \in A(T^\Delta)$ where T^Δ is the topology of $V_F \cup \{v^\star\}$ in $\mathbf{B}$, the number of internal vertices on the unique uv -path in $\mathbf{B}$ is $\text{len}(u, v)$, and (iii) for every $v \in V_F \cup \{v^\star\}$, the size of the subtree rooted at v in $\mathbf{B}$ is $\text{siz}(v)$.*

As we do exhaustive guessing, the following is immediate.

Lemma 9.3.2. *If $(D, v^\star, k)$ is a YES-instance of TFP, there exists an sba $\mathbf{B}$ that complies with some tuple $(T^\star, \text{len}, \text{siz})$ we enumerate.*

9.3.2 Phase II: Verification of Guesses

In this section, we give an algorithm to check whether a guessed tuple $(T^\star, \text{len}, \text{siz})$ is “realizable”. We first define the verification operations we perform on a tuple $(T^\star, \text{len}, \text{siz})$.

Definition 9.3.5 (Realizability). A tuple $(T^*, \text{len}, \text{siz})$ is realizable if there is a **usba** B on n vertices with an injective function $\phi : V(T^*) \rightarrow [n]$ satisfying the properties below.¹

1. v^* is the root of B , that is, ϕ maps v^* to the root of B .
2. T^* is the template of $V_F \cup \{v^*\}$ in B .
3. For each $(u, v) \in A(T^*)$, $\text{len}(uv)$ is equal to the number of internal vertices on the uv -path in B .
4. For each $v \in V_F \cup \{v^*\}$, $\text{siz}(v)$ is equal to the size of the subtree rooted at v in B .

We say that $(T^*, \text{len}, \text{siz})$ is realizable by the pair (B, ϕ) .

To check whether a tuple $(T^*, \text{len}, \text{siz})$ is realizable by at least one pair (B, ϕ) , we give a dynamic programming (DP) algorithm. Let B be a **usba** on n vertices. For a vertex $u \in V(T^*)$, let $\ell(u)$ denote the number of children of u in T^* , and let $x(1, u), \dots, x(\ell(u), u)$ denote the children themselves. Here, we refer to $x(i, u)$ as the i^{th} child of u . Furthermore, by $T_{u, x(i, u)}^*$ we denote the subtree of T_u from which we have deleted all the subtrees rooted at $x(i+1, u), \dots, x(\ell(u), u)$. Whenever it is clear from the context, we use x_i to denote $x(i, u)$. We define $x_0 = \text{nil}$, and $T_{u, \text{nil}}^*$ to be the subtree of T_u with a single vertex, u . For a vertex $v \in V(B)$, by C_v we denote the set of children of v . For a vertex $v \in V(B)$ and $C \subseteq C_v$, by $B_{v, C}$ we denote the subtree of B_v rooted at v from which we have deleted all the subtrees rooted at vertices in $C_v \setminus C$.

We use a multi-dimensional array M with an entry $M[u, v, x_i, C]$ for all $u \in V(T^*)$, $v \in V(B) = [n]$, x_i that is either $x_0 = \text{nil}$ or the i^{th} child of u , and $C \subseteq C_v$. The entry $M[u, v, x_i, C]$ is assigned 1 if and only if T_{u, x_i}^* is realizable by $B_{v, C}$. In

¹ Here, we abuse notation: if a vertex v in T^* is mapped (by ϕ) to some vertex $i \in [n]$ in B , we refer to i as v .

particular, note that $(T^*, \text{len}, \text{siz})$ is realizable by some pair (B, ϕ) if and only if $M[v^*, r, x(\ell(v^*), v^*), C_r]$ is 1. Here, r is the root of B .

Lemma 9.3.3. *There is a polynomial-time algorithm that checks whether a given tuple $(T^*, \text{len}, \text{siz})$ is realizable by at least one pair (B, ϕ) , and if the answer is positive, outputs such a pair.*

Proof. As we discussed above, it suffices to show that M can be filled-up correctly in polynomial time. To derive a pair (B, ϕ) as required (if the answer is positive), simply use standard backtracking on the computation that follows. The number of choices of u, v, x_i is clearly polynomial in n . Here, the crucial point is that by Observation [9.2.1](#), the number of choices for C is at most $2^{\log n} = n$. Thus, the size of the array is polynomial in n , and it remains to show how to compute each of its entries in polynomial time. As the following approach is quite standard, we only briefly sketch the computation itself. The entries are filled up in post-order traversal of T^* . Furthermore, when we consider an entry corresponding to a vertex $u \in V(T^*)$, we first fill all entries corresponding to $M[u, \star_1, x_1, \star_2]$, then $M[u, \star_1, x_2, \star_2]$ and so on. Here, $\star_1$ corresponds to a vertex in $V(T^*)$ and $\star_2$ corresponds to a subset of $C_{\star_1}$.

For a rooted tree T and two vertices $a, b \in V(T)$, by $\text{dist}_T(a, b)$ we denote the number of internal vertices on the ab -path in T . For example if $(a, b) \in A(T)$ then $\text{dist}(a, b) = 0$.

In the base case, where $x_i = \text{nil}$ (that is, $i = 0$), $M[u, v, x_0, C]$ is set to 1 if and only if the size of the subtree $B_{v,C}$ is $\text{siz}(u)$. Observe that, here we are mapping u to v and since u is a leaf of T^* , the only way T_{u,x_0}^* can be realized by $B_{v,C}$, if the number of vertices in the subtree $B_{v,C}$ is $\text{siz}(u)$.

In the recursive step, we apply the first step that is applicable:

$M[u, v, x_i, C]$ is set to 0 if u is a fvs-vertex and $|V(B_{v,C})|$ is not equal to $\text{siz}(v)$. The

correctness of this step is evident.

$$M[u, v, x_i, C] = \bigvee_{\substack{c \in C \\ w \in V(B_{v,C}) \\ \text{dist}_{B_{v,C}}(v, w) = \text{len}(u, x_i)}} M[u, v, x_{i-1}, C \setminus \{c\}] \\ \bigwedge M[u, v, x(\ell(x_i), x_i), C_w]$$

The idea of this recursive formula is the following: T_{u, x_i}^* is realizable by $B_{v, C}$ if and only if there exists a child $c \in C$ and a vertex $w \in B_c$ such that $T_{u, x_{i-1}}^*$ is realizable by $B_{v, C \setminus \{c\}}$, $T_{x_i, x(\ell(x_i), x_i)}^*$ is realizable by B_{w, C_w} and the $\text{dist}_{B_{v,C}}(v, w) = \text{len}(u, x_i)$. In this recursive step we implement this idea. When we are processing the vertex u and its i^{th} child, x_i , we have already processed the first $i - 1$ child of u . We also know that either u is not a fvs-vertex or $|V(B_{v,C})| = \text{siz}(v)$. In this case, idea is that we map u to v and want to test whether T_{u, x_i}^* is realizable by $B_{v, C}$. In order to do this we need to find the place where x_i can be mapped to. Towards this, we first guess in which subtree of $B_{v, C}$, rooted at a vertex in C , does x_i is mapped to. Note that every child of u is mapped in a different subtree of $B_{v, C}$, rooted at a vertex in C . Next, we guess on which vertex of $V(B_{v,C})$ does x_i is mapped to. Finally, we check whether $\text{dist}_{B_{v,C}}(v, w) = \text{len}(u, x_i)$. That is, whether the number of internal vertices on the vw -path is equal to the integer given by $\text{len}(u, x_i)$. we check whether $T_{u, x_i}^* \diamond$

9.3.3 Phase III: Greedy Choice for Path Resolution

Given a tuple $(T^*, \text{len}, \text{siz})$, we use a greedy strategy to construct an **sba** $\mathbf{B}$ such that $(T^*, \text{len}, \text{siz})$ is realizable by $\mathbf{B}$ (if one exists). As a preprocessing step, for every arc $(u, v) \in A(T^*)$ with $u, v \in V_F \cup \{v^*\}$ and $\text{len}(u, v) = 0$, we check that indeed $(u, v) \in A(D)$. If this is not the case, then we conclude that there is no **sba** that complies with $(T^*, \text{len}, \text{siz})$.

Now, recall that our greedy algorithm proceeds in two steps. First, we substitute

PathGreedy:

1. Let v be the highest unresolved fas-vertex in $T^\star$ and w be its fas-parent.
2. Consider the wv -path (P_{wv}) in T (not in $T^\star$). Note that in each step we always resolve all the arcs of a path simultaneously. So the set of unresolved arcs on the path P_{wv} form a subpath of P_{wv} that ends at v . Let x be the first vertex on the path P_{wv} such that the no arc on the unique xv -path (P_{xv}) in T is resolved.
3. Let H be the set of available vertices between x and v in the ordering σ . If $|H| \geq \eta = \sum_{(a,b) \in A(P_{xv})} (\text{len}(ab) + 1) - 1$, then let H' denote the subset of H consisting of highest η vertices in it. Then, we replace the path P_{xv} by the path P'_{xv} , which consists of vertices in H' : in the path P'_{xv} , the vertices in H' are ordered from highest to lowest rank.
4. If $|H| < \eta$, return that there is no **sba** that complies with $(T^\star, \text{len}, \text{siz})$.

each arc (v, u) in $T^\star$ by a path, which brings us closer to uncovering some **sba** **B**. In this section, we only consider the procedure to fill-up these paths. Specifically, given the template $T^\star$, we will construct a tree T that is a template of a superset of $V_F \cup \{v^\star\}$. In particular, it will be a template of $V(T)$ itself.

The tree we construct in the process is referred to as T . Initially, $T = T^\star$. For an arc $(u, v) \in A(T^\star)$, we call (u, v) *resolved* if it has been replaced by a path in T that has $\text{len}(uv)$ internal vertices. An fas-vertex v is called *resolved* if either it is the root ($v^\star$) or all the arcs on the path from its fas-parent, w , to v are resolved. We call a vertex $v \in V(D) \setminus V_F$ *available* if $v \in V \setminus V(T)$. That is, as vertices are added to T to fill the paths corresponding to arcs in $V(T^\star)$, they become unavailable. A vertex is the *highest available* vertex in $X \subseteq V(D)$ if it precedes all other available vertices in X according to σ .

With this notation, we repeat procedure **PathGreedy** until all arcs are resolved. Note that in one step (that is, one execution of the procedure), several arcs of $A(T^\star)$ can be resolved.

Let T^i denote the tree T that we had at hand immediately after we resolved the i^{th} vertex. The root is automatically resolved, and hence $T^1 = T^\star$. Note that

$T^{k+1} = T$. We say that **PathGreedy** *fails in step i* , if it concludes in i^{th} step (step in which it resolves the i^{th} vertex) that no **sba** complies with $(T^*, \text{len}, \text{siz})$, and else we say that it *succeeds* in i^{th} step. We say that **PathGreedy** *fails* if it concludes that no **sba** complies with $(T^*, \text{len}, \text{siz})$ in some step, and else we say that it *succeeds*. To argue about the correctness of our greedy choices, we need to extend Definition [9.3.4](#).

Definition 9.3.6. *An **sba** $\mathbf{B}$ complies with T^i if it complies with $(T^*, \text{len}, \text{siz})$ and the template of $V(T^i) \cap V(D)$ in $\mathbf{B}$ is T^i .*

The main components of proof are Lemmata [9.3.4](#) and [9.3.5](#).

Lemma 9.3.4. *Suppose that **PathGreedy** succeeds up to step i for some $i \in [k]$. Let $\mathbf{B}$ be an **sba** that complies with T^i . Then, **PathGreedy** succeeds at step $i + 1$.*

Proof. Let v be the highest unresolved fas-vertex in T^* that we consider in step $i + 1$ and let w be its fas-parent. Furthermore, let x be the first vertex on the path P_{wv} in T^i ($= T$) such that the no arc on the unique xv -path (P_{xv}) in T^i is resolved. In order to prove the lemma we need to show that when **PathGreedy** resolves v in step $i + 1$ then there is sufficient number of available vertices. In particular, let H be the set of available vertices between x and v in the ordering σ . Let $\eta = \sum_{(a,b) \in A(P_{xv})} (\text{len}(ab) + 1) - 1$. Then we need to show that there is at least η available vertices between x and v .

By the premise of the lemma, we know that $\mathbf{B}$ is an **sba** that complies with T^i . Thus, it complies with $(T^*, \text{len}, \text{siz})$ and the template of $V(T^i) \cap V(D)$ in $\mathbf{B}$ is T^i . Since, it complies with $(T^*, \text{len}, \text{siz})$ we have that for each arc $(u, v) \in A(T^\Delta)$, the number of internal vertices on the unique uv -path in $\mathbf{B}$ is $\text{len}(u, v)$. For every arc (a, b) on the path P_{xv} , we have the corresponding arc (a^Δ, b^Δ) in $A(T^\Delta)$ such that the number of internal vertices on the unique uv -path in $\mathbf{B}$ is $\text{len}(u, v)$. Since the template of $V(T^i) \cap V(D)$ in $\mathbf{B}$ is T^i , we have that for any two arcs (a_1, b_1) and (a_2, b_2) in $A(P_{xv})$, the corresponding $a_1^\Delta b_1^\Delta$ -path in $\mathbf{B}$ and $a_2^\Delta b_2^\Delta$ -path in $\mathbf{B}$ are

internally pairwise vertex disjoint. Let $P_{xv} = x \rightarrow v_1 \rightarrow \cdots \rightarrow v_\eta \rightarrow v$. Then consider the path in $\mathbf{B}$ corresponding to arcs on $P_{xv}^\Delta = x \rightarrow v_1^\Delta \rightarrow \cdots \rightarrow v_\eta^\Delta \rightarrow v$. Observe that, in $\mathbf{B}$, we can concatenate the paths corresponding to each arc on P_{xv}^Δ and obtain a xv -path in $\mathbf{B}$ that has $\sum_{(a,b) \in A(P_{xv})} (\text{len}(ab) + 1) - 1$ number of internal vertices. Since $\mathbf{B}$ complies with T^i we have that each vertex on xv -path in $\mathbf{B}$ is available. Furthermore, since there is no internal fas-vertex on the path P_{xv} we have that there is no fas-vertex on xv -path in $\mathbf{B}$. Thus, every vertex on xv -path in $\mathbf{B}$ is between x and v with respect to σ . Thus, since every vertex on xv -path in $\mathbf{B}$ is available, we have that there is at least η available vertices between x and v when PathGreedy performs step $i + 1$. This concludes the proof. $\diamond$

For the sake of clarity, before we present Lemma 9.3.5, we define a procedure used only for analysis. Suppose that PathGreedy succeeds up to step $i + 1$. Here, suppose we have some **sba** $\mathbf{B}$ that complies with T^i for some $i \in [k]$. As PathGreedy succeeds at step $i + 1$, T^{i+1} is well defined. Let v, x, H', P_{xv} and P'_{xv} be as defined in step $i + 1$ of PathGreedy. Now, supposing that $\mathbf{B}$ does not comply with T^{i+1} , there exists a vertex p on the unique xv -path in $\mathbf{B}$ that does not belong to H' , and a vertex $q \in H'$ that does not belong to the unique xv -path in $\mathbf{B}$. Since the length of these paths are same we have that the only way the last assertion can be violated if both these paths use the same set of vertices but in different order. However, this is not possible since $D[H']$ is a transitive tournament and thus there is a unique directed path spanning all the vertices in H' . Having these (hypothetical) elements at hand, we consider procedure PathAnalysis.

Lemma 9.3.5. *Assume PathGreedy succeeds up to step $i + 1$, and let $\mathbf{B}, v, x, H', P_{xv}, P'_{xv}, p$ and q be defined as in the procedure PATHANALYSIS. If $\mathbf{B}$ does not comply with T^{i+1} , then if these elements are the input to PathAnalysis, it returns $\mathbf{B}^*$ that is an **sba** with the following properties: (i) it complies with T^i ; (ii) the number of vertices on the unique xv -path in $\mathbf{B}^*$ from H' is larger by 1 than the number of*

PathAnalysis:

1. Let $Q = x \rightarrow u_1 \rightarrow \dots \rightarrow u_\eta \rightarrow v$ be the unique xv -path in $\mathbf{B}$. (This is well defined as $\mathbf{B}$ complies with T^i .)
2. For all $i \in [\eta]$, let u'_i be the i^{th} highest vertex in $(\{u_1, \dots, u_\eta\} \setminus \{p\}) \cup \{q\}$ according to σ .
3. For all $i \in [\eta]$, put u'_i in place of u_i in $\mathbf{B}$, and put p in (the original) place of q in $\mathbf{B}$. Denote the result by $\mathbf{B}'$.
4. As long as p has a child r' in $\mathbf{B}'$ such that $(r', p) \in A(D)$, let r be the highest such child according to σ , and swap p and r . Denote the result by $\mathbf{B}^*$.
5. Return $\mathbf{B}^*$.

vertices on the unique xv -path in $\mathbf{B}$ from H' .

Proof. Let $Q = x \rightarrow u_1 \rightarrow \dots \rightarrow u_\eta \rightarrow v$ be the unique xv -path in $\mathbf{B}$ and let Q' denote the unique xv -path in $\mathbf{B}'$ that we obtained in the third step of the procedure PathAnalysis. An arc (u, v) of $\mathbf{B}$ or $\mathbf{B}^*$ is called a *bad arc*, if u is a child of v (that is, if we forget the direction then v is more closer to the root than u). Observe that if we orient all the arcs in $\mathbf{B}^*$ away from v^* then it is a **sba**. Thus, the only reason $\mathbf{B}^*$ is not an **sba** is because there exist a bad arc. Through several claims we first show that there is no bad arc in $\mathbf{B}^*$. By our choice of H' the next claim follows.

Claim 9.3.1. *q is higher than p in σ .*

Another simple claim is the following.

Claim 9.3.2. *Let a be a non fas-vertex. Then a beats b if and only if a is higher than b in σ .*

In the next five claims we prove some invariants of the algorithm that will finally help us prove the lemma.

Claim 9.3.3. *In $\mathbf{B}'$ the only bad arcs that are possible are between p and some of its children.*

Proof. An arc that could become a bad arc in $\mathbf{B}'$ is of following types: (a) an arc

on the path Q' ; (b) an arc between a vertex u'_i on Q' and its child w_j ; (c) an arc between the parent of p and p and (d) an arc between p and some of its children. We show that the first three types of bad arcs can not exist. Observe that all the vertices in H' as well as on the path Q lies between x and v on σ . Furthermore, between x and w there are no fas-vertex and hence $D[H' \cup V[Q]]$ is a transitive tournament. The path Q' is obtained from $(\{u_1, \dots, u_\eta\} \setminus \{p\}) \cup \{q\}$ by following the ordering given by σ on $D[H' \cup V[Q]]$ and thus there is no bad arc on Q' . For the second type, observe that there is an arc from u_i to w_j , since $\mathbf{B}$ is an **sba**. Furthermore, since u_i is not a fas-vertex we have that w_j is lower than u_i . However, u'_i is higher than u_i and u'_i is not a fas-vertex and thus there is an arc (u'_i, w_j) . This implies that there is no bad arc of second type. For the non-existence of bad arc of third type, we argue as follows. By Claim [9.3.1](#), we know that q is higher than p in σ . Furthermore, since q is not a fas-vertex we have that parent of q in $\mathbf{B}$ is higher than q in σ . However, the parent of q in $\mathbf{B}$ is same as the parent of p in $\mathbf{B}'$ and thus it beats p . This proves that there is no bad arc of third type. This proves the claim. $\diamond$

Claim 9.3.4. *Let $\text{faschild}_{\mathbf{B}}(q)$ be the set of fas-children of q in $\mathbf{B}$. Then no vertex in $\text{faschild}(p)$ is resolved upto step $i + 1$.*

Proof. Let w' be the fas-parent of q in $\mathbf{B}$. Assume that there is a vertex v' in $\text{faschild}(q)$ that has been resolved earlier. Then this implies that the path resolving $w'v'$ -path in T^j for some $j \leq i$ is present in T^i . However, since $\mathbf{B}$ complies with T^i we have that this path is also present in $\mathbf{B}$. But the $w'v'$ -path in $\mathbf{B}$ contains q . This implies that $q \in V(T^i)$. But, we know that the vertices in H' are available at step $i + 1$. Since, $q \in H'$ we have that $q \notin V(T^i)$. This contradicts the existence of v' in $\text{faschild}(q)$ that has been resolved earlier. This concludes the proof. $\diamond$

Claim 9.3.5. *Let $\text{faschild}_{\mathbf{B}'}(p)$ be the set of fas-children of p in $\mathbf{B}'$. Then, p beats all of them.*

Proof. By Claim 9.3.4 we know that no vertex in $\text{faschild}(q)$ is resolved until step $i + 1$. Since, $\text{faschild}_{\mathbf{B}'}(p)$ is same as $\text{faschild}_{\mathbf{B}}(q)$ we know that no vertex in $\text{faschild}(p)$ is resolved until step $i + 1$. However, in step $i + 1$ PathGreedy chooses v to resolve and thus, we know that every vertex in $\text{faschild}(p)$ is lower than v . However, p beats v and thus p is higher than v . Now since p is a non-fas vertex by Claim 9.3.2, we have that p beats every vertex in $\text{faschild}(p)$. $\diamond$

Claim 9.3.6. *During the entire procedure of PathAnalysis, p is neither swapped with an fas-vertex, nor any other vertex in $V(T^i) \cap V(D)$.*

Proof. Look at the step 4 of the procedure PathAnalysis. In this step as long as p has a child r' in $\mathbf{B}'$ such that $(r', p) \in A(D)$, let r be the highest such child according to σ , and swap p and r . By, Claim 9.3.5 we know that p beats all vertices in $\text{faschild}_{\mathbf{B}'}(p)$ and thus it will never swap with any vertex in $\text{faschild}_{\mathbf{B}'}(p)$. Since, before encountering any other fas-vertex, p will encounter a vertex in $\text{faschild}_{\mathbf{B}'}(p)$ we have that p is never swapped with an fas-vertex. The proof that p is not swapped with any vertex in $V(T^i) \cap V(D)$ follows by the execution of PathGreedy. That is, the way PathGreedy resolves vertices. This concludes the proof. $\diamond$

Recall that because of Claim 9.3.3 we know that all the bad arcs in $\mathbf{B}'$ are confined to p and some of its children. In the next claim we will show that this property remains *invariant* even after each swap operation in fourth step of procedure PathAnalysis. Suppose that swap operation occurs s times in PathAnalysis, starting from $\mathbf{B}'$. Let $\mathbf{B}'_0 = \mathbf{B}'$ and $\mathbf{B}'_j$, $j \in [s]$, denote the “tree” obtained after the j^{th} swap operation applied on $\mathbf{B}'_{j-1}$.

Claim 9.3.7. *Suppose that swap operation happens s times in PathAnalysis. For each integer $j \in \{0, \dots, s\}$, the only bad arcs that may occur in $\mathbf{B}'_j$ are between p and some of its children.*

Proof. We prove this by induction on j . The base case follows from Claim [9.3.3](#). For the inductive step assume that this holds for some $0 \leq j < s$. By induction hypothesis we know that the only bad arcs that may occur in $\mathbf{B}'_j$ are between p and some of its children. Now suppose we swap p with child r' in $\mathbf{B}'_j$ such that $(r', p) \in A(D)$. Observe that in $\mathbf{B}'_{j+1}$ the only bad arcs that potentially can occur are either between p and some of its children or between the parent of p , $\text{parent}(p)$ and r . Consider the path, P^* , between $\text{parent}(q)$ and r in $\mathbf{B}$. Observe that this does not contain any fas-vertex and thus $D[V(P^*)]$ is transitive tournament and hence has a unique path spanning all the vertices. In particular, this implies that every vertex appearing on the path beats every other vertex that appears after it. Observe that $\text{parent}(p)$ appears before r on this path and thus beats it. This implies that there is no bad arc between $\text{parent}(p)$ and r in $\mathbf{B}'_{j+1}$. This concludes the proof. $\diamond$

By Claim [9.3.7](#), we know that in each swap operation the only bad arcs that may occur are confined to p and its children. Thus, when the swapping operation finishes either p becomes a leaf vertex or there are no bad arcs incident to p . This proves that indeed $\mathbf{B}^*$ is indeed an **sba**. By Claim [9.3.6](#) it follows that $\mathbf{B}^*$ complies with T^i . To see the last claim observe that the only way it can get violated is that p is swapped back to the path Q' . However, for this to happen p must be swapped with an fas-vertex. But, due to Claim [9.3.6](#) we know that p is never swapped with an fas-vertex. This proves the third part of the lemma and concludes the proof. $\diamond$

From Lemma [9.3.5](#), we derive the following consequence.

Corollary 9.3.1. *Suppose that PathGreedy succeeds up to step $i + 1$. If there is an **sba** $\mathbf{B}$ that complies with T^i , then there is an **sba** $\mathbf{B}^*$ that complies with T^{i+1} .*

Proof. We prove by contradiction. Among all **sba** that comply with T^i , let $\mathbf{B}$ be an **sba** that has the largest number of vertices on the unique xv -path in $\mathbf{B}$ from H' . If $\mathbf{B}$ does not comply with T^{i+1} , then by Lemma [9.3.5](#) we have that there exists an

sba $\mathbf{B}^*$ that complies with T^i ; and the number of vertices on the unique xv -path in $\mathbf{B}^*$ from H' is larger by 1 than the number of vertices on the unique xv -path in $\mathbf{B}$ from H' . This contradicts our choice of $\mathbf{B}$. $\diamond$

Having the lemma above at hand, our proof is done by induction. For the sake of clarity, let us extract the claim.

Lemma 9.3.6. *For all $i \in [k + 1]$, if there is an **sba** $\mathbf{B}$ complying with $(T^*, \text{len}, \text{siz})$, then (i) **PathGreedy** succeeds up until step i , and (ii) there is an **sba** that complies with T^i .*

Proof. We prove this case by induction on i . At the base case, where $i = 1$, our greedy procedure has not yet begun, and hence it could not have yet failed. Moreover, $T^1 = T^*$, and thus an **sba** complying with $(T^*, \text{len}, \text{siz})$ also complies with T^1 . Thus, the base case is trivially correct. In what follows, suppose that the lemma is correct for some $i \in [k]$, and let us prove it for $i + 1$.

By induction hypothesis we know that **PathGreedy** succeeds up until step i , and (ii) there exists an **sba** that complies with T^i . Now by Lemma 9.3.4 **PathGreedy** succeeds in step $i + 1$. Now by Corollary 9.3.1 the proof of the lemma follows. $\diamond$

As a corollary to Lemma 9.3.6, we obtain the following.

Corollary 9.3.2. *If there is an **sba** $\mathbf{B}$ complying with $(T^*, \text{len}, \text{siz})$, then (i) **PathGreedy** succeeds, and (ii) there exists an **sba** that complies with T .*

Let us also explicitly state an observation that directly follows from our construction. (Here, correctness also relies on our preprocessing step.)

Observation 9.3.2. *If **PathGreedy** succeeds, then T is a subtree of D , and in particular $A(T) \subseteq A(D)$.*

9.3.4 Phase IV: Greedy Choices for Subtree Resolution

Because we managed to reach this phase, we have at hand (i) a pair (B, ϕ) that realizes $(T^*, \text{len}, \text{siz})$ (from Phase II), and (ii) the tree T outputted by **GreedyPaths** (from Phase III). Before we proceed to our greedy choices, let us first update (B, ϕ) to (B^*, ϕ^*) as follows. First, initialize $B^* = B$. Second, define $\phi^* : V(T) \rightarrow [n]$ as the injective function such that for all $v \in V(T^*)$, we have $\phi^*(v) = \phi(v)$, and for all $v \in V(T) \setminus V(T^*)$, we have $\phi^*(v) = i$ where i is the j^{th} vertex on the unique $\phi(v^*)\phi(w)$ -path in B^* for w being the fas-child of v in T^* of highest rank and j being the position of v on the unique v^*w -path of T . We remark that this notation is well defined because (B, ϕ) realizes $(T^*, \text{len}, \text{siz})$ and by our construction of T . The choice of w as the fas-child of highest rank does not matter in the sense that any choice of an fas-child w of v would have resulted in the same $\phi^*(v)$.

A central notion in our greedy choices is of *private vertices*, defined as follows.

Definition 9.3.7. Let $\widehat{T}$ be a rooted tree where $V_F \cup \{v^*\} \subseteq V(\widehat{T})$. The set of private vertices of a vertex $v \in V_F \cup \{v^*\}$ in $\widehat{T}$, denoted by $\text{priv}_{\widehat{T}}(v)$, is the set of vertices that belong to $\widehat{T}_v$, but not to $\widehat{T}_u$ rooted at a descendant $u \in V_F \setminus \{v\}$ of v . In case $\widehat{T} = T$, denote $\text{priv}(v) = \text{priv}_T(v)$.

We are now ready to describe our second phase of greedy choices. In this phase, each fas-vertex v has a “bucket”, which is a set W_v that is initially empty. Moreover, for each fas-vertex v , denote $\alpha_v = \text{siz}(v) - \sum_{u \in \text{faschild}_{T^*}(v) \cap V_F} \text{siz}(u)$. In addition, denote $\beta_v = \alpha_v - |\text{priv}(v)|$. Here, we say that an fas-vertex v is *unresolved* if its set W_v has not yet been updated. Moreover, we say that a vertex $u \in V(D)$ is *available* if $u \notin V(T) \cup (\bigcup_{v \in V_F \cup \{v^*\}} W_v)$. As long as there is an unresolved vertex, we apply the steps of procedure **SubtreeGreedy**.

We say that **SubtreeGreedy** *fails* if it concludes that no **sba** complies with $(T^*, \text{len}, \text{siz})$, and else we say that it *succeeds*. For $i \in [k + 1]$, let v^i be the fas-

SubtreeGreedy:

1. Let v be the highest unresolved fas-vertex.
2. Let H be the set of the β_v highest available vertices beaten by v . If not enough such vertices are found, return that no **sba** complies with $(T^*, \text{len}, \text{siz})$.
3. Update W_v to be equal to H .

vertex considered at step i . To argue about the correctness of our greedy choices, we need to further extend Definition [9.3.6](#).

Definition 9.3.8. *An sba $\mathbf{B}$ complies with v^i if it complies with T and $\text{priv}_{\mathbf{B}}(v) = W_{v^i} \cup \text{priv}(v^i)$. Moreover, $\mathbf{B}$ complies with $v^{\leq i}$ if it complies with all the vertices $v^1, \dots, v^i$.*

The main argument in our proof are summarized in Lemmata [9.3.7](#), [9.3.9](#) and [9.3.10](#).

Lemma 9.3.7. *Suppose that SubtreeGreedy succeeds up to step i . Let $\mathbf{B}$ be an sba that complies with $v^{\leq i}$ for some $i \in [k+1]$. Then, SubtreeGreedy succeeds at step $i+1$.*

Proof. Denote $v = v^{i+1}$. To show that SubtreeGreedy succeeds at step $i+1$, it suffices to show that in the beginning of that step, there exist β_v available vertices beaten by v . Let $X = \text{priv}_{\mathbf{B}}(v) \setminus V(T)$. Then, by the definition of priv , for every vertex $x \in X$ there is a directed path from v to x in $\mathbf{B}$ whose only fas-vertex is B . In particular, this means that v beats x . In addition, since $\mathbf{B}$ complies with $v^{\leq i}$, and for all $j \leq i$, we have $\text{priv}_{\mathbf{B}}(v) \cap \text{priv}_{\mathbf{B}}(v^j) = \emptyset$, we have that all the vertices in $X \setminus \{v\}$ are available in the beginning of step i . As $|X|$ is precisely β_v , this completes the proof. $\diamond$

Towards Lemma [9.3.9](#), we need yet another lemma.

Lemma 9.3.8. *For all $i \in [k+1]$ and $u \in \text{priv}(v^i)$, the vertex u beats all the vertices in W_{v^i} .*

Proof. Choose $i \in [k + 1]$ and $u \in \text{priv}(v^i)$ arbitrarily. Let w be some vertex in W_{v^i} . Then, because $u \in \text{priv}(v^i)$, we have that the vertex u became unavailable in some iteration of PathGreedy. In that iteration u was chosen to a set H' , which contained the highest available vertices beaten by v at that point of time. As all the vertices in W_{v^i} are beaten by v and remained available at the end of that iteration, we conclude that u beats all of them. $\diamond$

We are now ready to state Lemma [9.3.9](#).

Lemma 9.3.9. *Suppose that PathGreedy succeeds up to step $i + 1$ for some $i \in [k + 1]$. Let $\mathbf{B}$ be an **sba** that complies with $v^{\leq i}$. Then, there is an **sba** that complies with $v^{\leq i+1}$.*

Proof. Let $\mathbf{B}$ be an **sba** that complies with $v^{\leq i}$ and which maximized $W_{v^{i+1}} \cap \text{priv}_{\mathbf{B}}(v^{i+1})$. Suppose that $\mathbf{B}$ does not comply with $v^{\leq i+1}$, else we are done. Thus, there exists a vertex $p \in \text{priv}_{\mathbf{B}}(v^{i+1})$ that does not belong to $W_{v^{i+1}}$, and a vertex $q \in W_{v^{i+1}}$ that does not belong to $\text{priv}_{\mathbf{B}}(v^{i+1})$. Let u be the fas-vertex such that $q \in \text{priv}_{\mathbf{B}}(u)$. By Lemma [9.3.8](#), we have that all vertices in $\text{priv}(v^{i+1})$ beat q .

Note that, by the specification of our algorithm, $W_{v^j} \cap W_{v^{i+1}} = \emptyset$. Since $u \neq v^{i+1}$, $q \in \text{priv}_{\mathbf{B}}(u)$, $\mathbf{B}$ complies with $v^{\leq i}$, this means that $u = v^j$ for some $j > i + 1$. Therefore, v^{i+1} is ranked higher than u . This means that v^{i+1} beats p (because u beats p). Then, by the definition of $W_{v^{i+1}}$, we deduce that every vertex in $W_{v^{i+1}}$, and in particular q , beats p . Thus, because q beats p while u beats q (because $q \in \text{priv}_{\mathbf{B}}(u)$), we have that u beats p . Using the arguments in the proof of Lemma [9.3.8](#), we further conclude every vertex in $\text{priv}(u)$ beats p . Let v' be the vertex in $\text{priv}(v)$ of maximum distance from the root in $\mathbf{B}$ such that p belongs to the subtree of v' in $\mathbf{B}$, and let u' be the vertex in $\text{priv}(u)$ of maximum distance from the root in $\mathbf{B}$ such that q belongs to the subtree of u' in $\mathbf{B}$. Now, let $U_{v'}$ be the set of all the vertices in the subtree of v' that do not belong to the subtree of any other vertex in

$V(T)$. Similarly, define $U_{u'}$.

Note that both $D[\{v'\} \cup (U_{v'} \setminus \{p\}) \cup \{q\}]$ and $D[\{u'\} \cup (U_{u'} \setminus \{q\}) \cup \{p\}]$ are acyclic, and by the arguments above and Lemma 9.3.8, we deduce that in the first one, v' is the highest vertex (according to σ), and in the second one, u' is the highest vertex. By Lemma 9.2.1, $D[\{v'\} \cup (U_{v'} \setminus \{p\}) \cup \{q\}]$ contains (as a subgraph) an $\hat{A}_{v'}$ arborescence isomorphic to $\mathbf{B}[\{v'\} \cup U_{v'}]$. Similarly, $D[\{u'\} \cup (U_{u'} \setminus \{q\}) \cup \{p\}]$ contains (as a subgraph) an $\hat{A}_{u'}$ arborescence isomorphic to $\mathbf{B}[\{u'\} \cup U_{u'}]$. Thus, by replacing $\mathbf{B}[\{v'\} \cup U_{v'}]$ by $\hat{A}_{v'}$ and $\mathbf{B}[\{u'\} \cup U_{u'}]$ by $\hat{A}_{u'}$ in $\mathbf{B}$, we obtain an **sba** $\mathbf{B}'$ that complies with $v^{\leq i}$ and such that $|W^{v^{i+1}} \cap \text{priv}_{\mathbf{B}'}(v^{i+1})| > |W^{v^{i+1}} \cap \text{priv}_{\mathbf{B}}(v^{i+1})|$, which is a contradiction. $\diamond$

Towards the presentation of Lemma 9.3.10, we need the **SubtreeCompletion** procedure. Lemmata 9.3.7 and 9.3.9 are not sufficient to complete the proof, since they are based on the supposition that there exists an **sba** that complies with $(T^*, \text{len}, \text{siz})$, which is precisely the information we need to determine. From these lemmata alone, we would only be able to conclude that if there is an **sba** $\mathbf{B}$ complying with $(T^*, \text{len}, \text{siz})$, then **SubtreeGreedy** succeeds, but *not* that if **SubtreeGreedy** succeeds, then there is an **sba** $\mathbf{B}$ complying with $(T^*, \text{len}, \text{siz})$. The reverse direction will be ensured using Lemma 9.3.10. In case we do not only want to determine whether there exists an **sba** that makes v^* the winner, but also compute one such **sba**, the procedure below should be performed (that is, in that case this procedure is not given only for the sake of analysis). Here, it will also become apparent why we had to have at hand a pair (B, ϕ) that realizes our initial guess. A surprising implication of the proof below is that, although there can be many pairs that realize our initial guess, for the sake of exhibiting an **sba** that complies with our initial guess, taking any one of them would work. Initialize $(B^0, \phi^0) = (B^*, \phi^*)$. For $i = 1, 2, \dots, k+1$, $B^i = B^{i-1}$, and the procedure executes **SubtreeCompletion**.

The arguments given in the presentation of the procedure above directly imply

SubtreeCompletion:

1. Initialize ϕ^i to be ϕ^{i-1} , and U to be W_{v^i} .
2. For all $u \in \text{priv}(v^i)$, execute the following operations:
 - (a) Let $\widehat{B}_u$ be the subtree of B_u^i from which we remove the subtree of t for all $t \in [n]$ for which there is a vertex $w \in V(T) \setminus \{u\}$ such that $\phi^i(w) = t$.
 - (b) Let U_u be some subset of $|V(\widehat{B}_u)| - 1$ vertices from U , and update U to be $U \setminus U_u$.
 - (c) Note that $D[\{u\} \cup U_u]$ is acyclic. By Lemma 9.2.1, $D[\{u\} \cup U_u]$ contains (as a subgraph) an $\widehat{A}_u$ arborescence isomorphic to $\widehat{B}_u$ that is computable in polynomial time. Observe that u is the root of $\widehat{A}_u$, else u were been beaten by at least one vertex in U_u , contradicting Lemma 9.3.8.
 - (d) We update ϕ^i to label the vertices of $\widehat{B}_u$ according to their “copies” in $\widehat{A}_u$. (More precisely, let f be an isomorphism from $\widehat{B}_u$ to $\widehat{A}_u$. Then, for all $t \in V(\widehat{B}_u)$, define $\phi^i(t) = f(t)$.)

the correctness of the following lemma.

Lemma 9.3.10. *Suppose that SubtreeGreedy succeeds up to step i . Then, (1) for all $(u, v) \in A(B^{i+1})$ such that both u, v are in the image of ϕ^{i+1} and are private vertices of v^i in B^i , we have $(\phi^i(x), \phi^i(y)) \in A(D)$, and (2) ϕ^i extends ϕ^{i-1} , and it injectively maps all the vertices in W_{v^i} .*

Proof. The correctness of condition (1) is immediate since for every $u \in \text{priv}(v^i)$, we argued that $\widehat{A}_u$ is arborescence of $D[\{u\} \cup U_u]$ isomorphic to $\widehat{B}_u$ and rooted at u . Condition (2) also directly follows from the definition of the procedure, and because $\bigcup_{u \in \text{priv}(v^i)} U_u = W_{v^i}$ and $U_u \cap U_{u'} = \emptyset$ for all distinct $u, u' \in \text{priv}(v^i)$. $\diamond$

From Lemmata 9.3.7, 9.3.9 and 9.3.10, we have the following lemma.

Lemma 9.3.11. *For all $i \in [k+1] \cup \{0\}$, if there is an **sba** $\mathbf{B}$ complying with $(T^*, \text{len}, \text{siz})$, then (i) SubtreeGreedy succeeds up until step i and there exists an **sba** complying with $v^{\leq i}$, (ii) for all $(u, v) \in A(B^i)$ such that both u, v are in the image of ϕ^i , we have $(x, y) \in A(D)$ where $\phi^i(x) = u$ and $\phi^i(y) = v$, and (iii) ϕ^i extends ϕ^j for all $j < i$, and it injectively maps all the vertices in $V(T) \cup W_{v^1} \cup \dots \cup W_{v^i}$.*

Proof. The proof is by induction on the step i . In the base case, $i = 0$. Then, condition (i) vacuously holds. Condition (ii) is satisfied by Observation [9.3.2](#), and condition (iii) trivially holds.

Now, suppose that the claim is true for i , and let us prove it for $i + 1$. Then, by the inductive hypothesis there exists an **sba** complying with $v^{\leq i}$, and hence Lemma [9.3.7](#), **SubtreeGreedy** succeeds at step $i + 1$. Combined with Lemma [9.3.9](#), this further means that there exists an **sba** complying with $v^{\leq i+1}$, and therefore condition (i) is satisfied. Moreover, the satisfaction of conditions (ii) and (iii) directly follows from the inductive hypothesis and Lemma [9.3.10](#). $\diamond$

From Lemma [9.3.11](#), we derive the following corollary.

Corollary 9.3.3. *If there is an **sba** $\mathbf{B}$ complying with $(T^*, \text{len}, \text{siz})$, then **SubtreeGreedy** succeeds and $\mathbf{B}^{k+1}$ (where we abuse notation such that each vertex i is labelled by $\phi^{k+1-1}(i)$) is an **sba** rooted at v^* .*

Proof. Suppose there is an **sba** $\mathbf{B}$ complying with $(T^*, \text{len}, \text{siz})$. Then, by condition (i) of Lemma [9.3.11](#), **SubtreeGreedy** succeeds. Note that $V(T) \cap W_{v^i} = \emptyset$ for all $i \in [k+1]$, and $W_{v^i} \cap W_{v^j} = \emptyset$ for all $i \in [k+1]$. Since $|V(T)| + \cup v \in V_D \cup \{v^*\} \beta_v = n$, we thus conclude from condition (iii) that ϕ^{k+1} injectively maps all vertices in $V(D)$ and the root of $\mathbf{B}^{k+1}$ is v^* . By condition (ii), we further have that $\mathbf{B}^{k+1}$ is a subtree of D . Thus, as we started from a **usba**, we conclude that $\mathbf{B}^{k+1}$ is an **sba** rooted at v^* . $\diamond$

Finally, the correctness of our main result, summarized in the theorem below, directly follows from Lemma [9.3.2](#) and Corollary [9.3.3](#). We remark that here we rely on the straightforward observation that given an **sba** where v^* is the root, a seeding to make v^* win can be computed in polynomial time. The running time of the algorithm is dominated by the guesses $(k^{O(k)})$ in the first phase, as all other phases run in polynomial time.

Theorem 27. *TFP is solvable in time $2^{\mathcal{O}(k^* \log k^*)} n^{\mathcal{O}(1)}$. Moreover, for a YES-instance (D, v^*, k^*) , a seeding to make v^* win can be constructed within this time bound.*

Next, we develop an algorithm for BTF that runs in time $2^n n^{\mathcal{O}(1)}$ and has a polynomial space complexity, as well as a parameterized algorithm that runs in time $2^{\mathcal{O}(k^* \log k^*)} n^{\mathcal{O}(1)}$.

9.4 Obfuscation Operations

We give two operations to turn one solution into another. In each operation, we have a tournament D , a subset $R \subseteq A(D)$ of size at most ℓ , and a spanning binomial arborescence T of D^R rooted at v^* . Without loss of generality, suppose $R^{\text{rev}} = A(T) \setminus A(D)$. We say that an operation is *valid* if it returns a subset $R' \subseteq A(D)$ of size at most ℓ , and a spanning binomial arborescence T' of $D^{R'}$ rooted at v^* .

Subtree clean-up. Our first operation is called “subtree clean-up” (see Fig. 9.4). We say that this operation is *applicable* if there exists a vertex $v \in V(T) \setminus \{v^*\}$ such that $A(T_v) \cap R^{\text{rev}} \neq \emptyset$. Given such a vertex $v \in V(T) \setminus \{v^*\}$, the operation is performed as follows.

First, since $|V(T_v)|$ is a power of 2 (see Observation 9.2.1), we have that $D[V(T_v)]$ admits a spanning binomial arborescence T'_u , rooted at some vertex u . To see this, arbitrarily fix a tournament on $D[V(T_v)]$, conduct the competition, and let T'_u be the spanning binomial arborescence obtained by taking all arcs of $D[V(T_v)]$ that correspond to the matches that took place. Note that the root u of T'_u might be v itself. However, $T'_u \neq T_v$ because T'_u is a subtree of D while $A(T_v) \cap R^{\text{rev}} \neq \emptyset$ (which means that T_v is not a subtree of D). Let p be the parent of v in T (since $v \neq v^*$, this parent exists). Now, if $(p, u) \in A(D)$ then define $R' = R \setminus A(T_v)^{\text{rev}}$, and otherwise (if

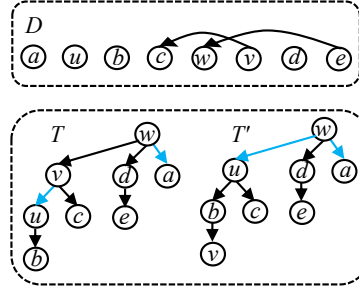


Figure 9.4: An application of the subtree clean-up operation. In D , displayed arcs are backward arcs; all other arcs are forward arcs. The sets R^{rev} and R'^{rev} are the sets of blue arcs in T and T' , respectively. In T' , the operation is not applicable.

$(u, p) \in A(D)$) define $R' = (R \setminus A(T_v)^{\text{rev}}) \cup \{(u, p)\}$. We define T' from T as follows: remove the tree T_v and the arc (p, v) , and then add the tree T'_u and the arc (p, u) .

Lemma 9.4.1. *The subtree clean-up operation is valid.*

Proof. Since T_v is a spanning binomial arborescence of $D^R[V(T_v)]$ (see Observation 9.2.1), we have that T'_u is isomorphic to T_v . This observation implies that T' is a spanning binomial arborescence of $D^{R'}$ rooted at v^* . Moreover, $|R'| \leq |(R \setminus A(T_v)^{\text{rev}}) \cup \{(u, p)\}| \leq |R| - |R \cap A(T_v)^{\text{rev}}| + 1 \leq |R|$. Thus, the validity of our operation follows. $\diamond$

Cyclic shift. Our second operation is called “cyclic shift” (see Fig. 9.5), where we assume that we are given a feedback arc set K of D with $|V(K)| = k$, and the topological order $<$ of D^K . We say that a directed subpath of T , denoted by $P = p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_q$, enables a cyclic shift if $p_1, p_2, \dots, p_q \notin V(K) \cup \{v^*\}$

1. are vertices of the same type $t \in \{0, 1, \dots, k\}$,
2. which are consecutive (that is, p_i is the predecessor of p_{i+1} for all $i \in \{1, 2, \dots, q-1\}$), and
3. the predecessor of p_1 exists, it is not the parent of p_1 in T , and it is also of type t .

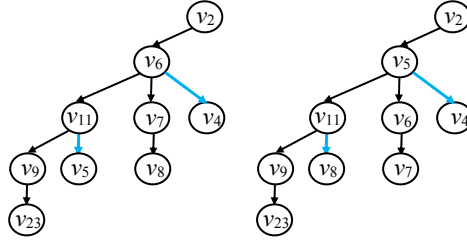


Figure 9.5: An application of the cyclic shift clean-up operation with D being the graph in Fig. 9.1 (add vertices to ensure its size is a power of 2, whose display is irrelevant here). Only parts of the trees T and T' are displayed. Reversed arcs are colored blue. Here, $P = v_6 \rightarrow v_7 \rightarrow v_8$, $p_0 = v_5$, $p_1 = v_6$, $p_2 = v_7$ and $p_3 = v_8$.

We say that the cyclic shift operation is *applicable* if we have a directed subpath of T that enables a shift. Given such a subpath $P = p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_q$, the operation is performed as follows. Let p_0 denote the predecessor of p_1 . Then, as the name of the operation suggests, we define T' from T as follows: for all $i \in \{0, 1, \dots, q\}$, replace the vertex p_i by the vertex $p_{(i-1) \bmod (q+1)}$. Denote $R' = (A(T') \setminus A(D))^{\text{rev}}$. We now argue why this operation is valid.

Lemma 9.4.2. *The cyclic shift operation is valid.*

Proof. First, note that T' is obtained from T by “permuting” vertices, all of which belong to $\{p_0, p_1, \dots, p_q\}$ and none of which is v^* . Hence, T' is a spanning binomial arborescence of $D^{R'}$ rooted at v^* . To prove that $|R'| \leq |R|$, it suffices to show that the number of arcs incident to $\{p_0, p_1, \dots, p_q\}$ in T' that do not belong to $A(D)$ is upper bounded by the number of arcs incident to $\{p_0, p_1, \dots, p_q\}$ in T that do not belong to $A(D)$. To this end, we claim that for all $i \in \{0, 1, \dots, q\}$, the number of arcs incident to p_i in T' that do not belong to $A(D)$ is upper bounded by the number of arcs incident to $p_{(i+1) \bmod (q+1)}$ in T that do not belong to $A(D)$. To complete the proof, we will verify this claim for p_0 . The claim for the other vertices can be verified similarly. Let x denote the parent of p_1 in T . Because p_0 is not the parent of p_1 in T , we have that $x \notin \{p_0, p_1, \dots, p_q\}$. In particular, this means that x is also the parent of p_0 in T' . Since p_0 is the predecessor of p_1 and both vertices do not belong to $V(K)$, we have that x beats p_0 if and only if x beats p_1 . Every child of p_0 in T' is

either p_1 or a child of p_1 in T . Since p_0 is the predecessor of p_1 and both vertices do not belong to $V(K)$, we have that p_0 beats p_1 , and every other child of p_0 in T' is beaten by p_0 if and only if it is beaten by p_1 . $\diamond$

9.5 Combinatorial Result

In this section, we show that if the input instance of BTF is a YES-instance, then it admits a solution with two very specific properties. Apart from being a combinatorial result of independent interest, it is also useful practically as it significantly shrinks our search space. Only the second property assumes the presence of a feedback arc set K . If one is interested only in the first property, s/he can set $K = A(D)$, and otherwise K can be obtained by either a polynomial-time approximation scheme (PTAS) [118], or an $2^{\mathcal{O}(\sqrt{|K|})}n^{\mathcal{O}(1)}$ -time parameterized algorithm [93, 47].

To formulate the second property, we introduce the following definition (examples are given below).

Definition 9.5.1. *Let D be a tournament with a feedback arc set K . Let $<$ be the topological order of D^K . A set $C \subseteq V(D)$ is an elite club if for all types $i \in \{0, \dots, |V(K)|\}$, there do not exist two vertices u, v of type i such that $u < v$, $u \notin C$ and $v \in C$. That is, if a vertex v of type i belongs to C , then all vertices u of type i that beat v also belong to C .*

For $v^ \in V(D)$, an elite club C is a v^* -elite club if $(v, v^*) \in A(D)$ for all $v \in C$. That is, all members of C beat w .*

For example, in Fig. 9.1, $C_1 = \{v_1, v_3, v_4, v_5, v_{13}, v_{14}, v_{15}, v_{18}\}$ is an elite club, while $C_2 = \{v_1, v_3, v_4, v_6, v_{13}, v_{14}, v_{15}, v_{18}\}$ is not since it contains v_6 but not its predecessor v_5 that belongs to the same type. It is easy to see that our notion of an elite club is incomparable to both the notion of kings and the best players defined by [147].

For example, a vertex v *last* of its type is a king if it beats, say, a vertex that beats any other vertex, and it is “likely to be” a best player if it belongs to one of the first types, but it belongs to an elite club C only if *all* vertices of the same type as v belong to C . Similarly, a vertex in an elite club (even if it is first of its type) can be neither a king nor a best player.

Our main combinatorial result says that if there is a way to make v^* win, then there is also such a way where all matches necessary to alter involve v^* itself. Moreover, it further says that all of the other players that are involved in these matches belong to a group of “top players” (among those of each type). We think these two findings are of philosophical interest as well. Formally, this is stated as follows.

Theorem 28. *Let (D, v^*, ℓ) be a YES-instance of BTF. Let K be a feedback arc set of D with $|V(K)| = k$, and let $<$ be the topological order of D^K . Then, there exists $R \subseteq A(D)$ with $|R| \leq \ell$ and a spanning binomial arborscence T of D^R rooted at v^* , which satisfy both properties below.*

1. *Every arc in R has v^* as an endpoint.*
2. *There exists a v^* -elite club C of size at most ℓ such that every arc in R has an endpoint in C .*

Proof. Let $R \subseteq A(D)$ be a subset of size at most ℓ such that D^R has a spanning binomial arborscence T rooted at v^* . We first show how to transform (R, T) into a solution that satisfies property [1](#). As long as v^* has a child v in T with $A(T_v) \cap R^{\text{rev}} \neq \emptyset$, apply the subtree clean-up operation (from Section [9.4](#)) to obtain another solution, and update (R', T') to be that solution. By Lemma [9.4.1](#) and the specification of this operation, it can be easily verified that this process terminates after at most $\log n$ steps (since v^* has $\log n$ children by Observation [9.4.1](#)), and when it does, no subtree rooted at a child of v^* contains a reversed arc, implying that property [1](#) is satisfied.

Now, we proceed to show how to transform (R, T) into a solution that also satisfies property [2](#). As long as this property is not satisfied (and because property [1](#) is satisfied), v^* has a child $v \notin V(K)$ such that the predecessor u of v is of the same type as v , but $(v, v^*) \in A(D)$ while u is not incident to an arc in R . In particular, $u \neq v^*$, and hence u is not the parent of v in T . Then, we perform the cyclic shift operation (from Section [9.4](#)) with $P = p_1 = v$ (that is, our path consists of a single vertex) and $p_0 = u$. Observe that this operation maintains property [1](#). Moreover, by Lemma [9.4.2](#) and from the specification of this operation, we have that this process terminates after less than n^2 steps, and when it does, property [2](#) is satisfied. $\diamond$

In case we have power to alter the outcome of at least $\log n$ matches, the following result states that we can always make our favorite player win. While this theorem is subsumed by Theorem [32](#) (given in Section [9.6](#)), we think it merits its own statement, and present a simplified proof for this special case.

Theorem 29. *Let (D, v^*, ℓ) be an instance of BTF. If $\ell \geq \log n$, then (D, v^*, ℓ) is a YES-instance. Moreover, a solution can be found in polynomial time.*

Proof. Fix an arbitrary tournament (competition), and let v be its winner. Let T'' be the spanning binomial arborescence of D , rooted at v , that corresponds to this competition. Now, swap v with v^* to obtain a spanning binomial arborescence T' of $D^{R'}$, rooted at v^* , for some $R' \subseteq A(D)$. By performing the subtree clean-up operation for every child of v^* in T' whose subtree contains a reversed arc (like it is done in the proof of Theorem [28](#)), we obtain a binomial arborescence whose reversed arcs are all incident to v^* . As v^* has at most $\log n$ children, this is a spanning binomial arborescence of D^R for some $R \subseteq A(D)$ of size at most $\log n \leq \ell$. $\diamond$

9.6 Characterization of YES-Instances

In this section, we present two (related) characterizations of YES-instances, along with one implication. First, based on Theorem 28 and Observation 9.2.1, we present a closed formula that resolves the special case of BTF where D is a DAG. To this end, we need the following definition.

Definition 9.6.1. *Let D be a tournament. The victory count of a vertex $v \in V(D)$, denoted by $\text{victory}(v)$, is defined by*

$$\text{victory}(v) = |\{u \in V(D) : (v, u) \in A(D)\}|.$$

That is, $\text{victory}(v)$ is the number of players beaten by v .

If D is acyclic, then $\text{victory}(v) = |\{u \in V(D) : v < u\}|$.

We are now ready to state our formula. While the statement is existential, the proof is constructive and implies how a solution, if one exists, can be found.

Theorem 30. *Let (D, v^*, ℓ) be an instance of BTF where D is a DAG. Then, (D, v^*, ℓ) is a YES-instance if and only if the expression $[\text{victory}(v^*) \geq \frac{n}{2^\ell} - 1]$ is true. In case the expression is true, a solution can be found in polynomial time.*

Proof. In one direction, suppose that $\text{victory}(v^*) \geq \frac{n}{2^\ell} - 1$ is true. Let S be a set of exactly $\frac{n}{2^\ell} - 1$ vertices beaten by v^* . Arbitrarily partition S into $\log n - \ell$ sets, $U_1, \dots, U_{\log n - \ell}$, such that $|U_i| = 2^{i-1}$ for all $i \in \{1, \dots, \log n - \ell\}$. Since $\sum_{i=1}^{\log n - \ell} 2^{i-1} = \frac{n}{2^\ell} - 1$, this is possible. Moreover, arbitrarily partition $V(D) \setminus (U \cup \{v^*\})$ into ℓ sets, $U_{\log n - \ell + 1}, \dots, U_{\log n}$, such that $|U_i| = 2^{i-1}$ for all $i \in \{\log n - \ell + 1, \dots, \log n\}$. Now, for all $i \in \{1, \dots, \log n\}$, fix a tournament (competition) for $D[U_i]$, and let T_i denote the corresponding spanning binomial arborescence of $D[U_i]$ rooted at some vertex w_i . Define T by $V(T) = V(T_1) \cup \dots \cup V(T_{\log n}) \cup \{v^*\}$ and

$A(T) = A(T_1) \cup \dots \cup A(T_{\log n}) \cup \{(v^*, w_i) : i \in \{1, \dots, \log n\}\}$. Moreover, define $R = \{(w_i, v^*) \in A(D) : i \in \{\log n - \ell + 1, \dots, \log n\}\}$. Then, $|R| \leq \ell$. From the definition of $A(T)$, T is rooted at v^* . For each $i \in \{1, \dots, \log n\}$, T_i is a spanning binomial arborescence. Therefore, T is a spanning binomial arborescence of D^R rooted at v^* . The proof also indicates how to construct a solution in polynomial time.

In the other direction, suppose that (D, v^*, ℓ) is a YES-instance. Let (R, T) be a solution that satisfies property [1](#) in Theorem [28](#). Let X be the set of children v of v^* in T such that $(v^*, v) \in A(D)$, and denote $U = \bigcup_{v \in X} V(T_v)$. Since $|R| \leq \ell$, we have that $|X| \geq \log n - \ell$. Thus, by Observation [9.2.1](#), $|U| = \sum_{v \in X} |V(T_v)| \geq \sum_{i=1}^{\log n - \ell} 2^{i-1} = \frac{n}{2^\ell} - 1$. Moreover, since (R, T) satisfies property [1](#), we have that $A(T_v) \cap R^{\text{rev}} = \emptyset$ for all $v \in X$. Thus, since $(v^*, v) \in A(D)$ for all $v \in X$ and D is a DAG, we have that v^* beats all the vertices in U . $\diamond$

In our proof of the forward direction above, it can be shown that v^* is made a king by reversing ℓ arcs. The spirit of the proof of Theorem [30](#) is not limited to the special case of DAGs. Indeed, we have a characterization of YES-instances also for general tournaments. However, this characterization is not nice in the sense that it cannot be verified in polynomial time. Nevertheless, we find it of theoretical interest, and it will be used later. In this context, recall that BTF is NP-hard even when $\ell = 0$, and hence we do not expect it to admit an easy to verify characterization.

Theorem 31. *Let (D, v^*, ℓ) be an instance of BTF. Then, (D, v^*, ℓ) is a YES-instance if and only if there exists $U \subseteq V(D)$ of size exactly $\frac{n}{2^\ell} - 1$ so that there is a fixing of $D[U \cup \{v^*\}]$ that makes v^* win. Given such U with its fixing, a solution can be found in polynomial time.*

Proof. In one direction, suppose that there exists $U \subseteq V(D)$ of size exactly $\frac{n}{2^\ell} - 1$ so that there is a fixing of $D[U \cup \{v^*\}]$ that makes v^* win, and let T' be the

corresponding binomial arborescence of $D[U \cup \{v^*\}]$ rooted at v^* . Arbitrarily partition $V(D) \setminus (U \cup \{v^*\})$ into ℓ sets, $U_{\log n - \ell + 1}, \dots, U_{\log n}$, such that $|U_i| = 2^{i-1}$ for all $i \in \{\log n - \ell + 1, \dots, \log n\}$. For all $i \in \{\log n - \ell + 1, \dots, \log n\}$, fix a tournament (competition) for $D[U_i]$, and let T_i denote the corresponding spanning binomial arborescence of $D[U_i]$ rooted at some vertex w_i . Define T by $V(T) = V(T_{\log n - \ell + 1}) \cup \dots \cup V(T_{\log n}) \cup V(T')$ and $A(T) = A(T_{\log n - \ell + 1}) \cup \dots \cup A(T_{\log n}) \cup \{(v^*, w_i) : i \in \{\log n - \ell + 1, \dots, \log n\}\} \cup A(T')$. Moreover, define $R = \{(w_i, v^*) \in A(D) : i \in \{\log n - \ell + 1, \dots, \log n\}\}$. Similarly to the proof of Theorem 30, we have that T is rooted at v^* (from the definition of $A(T)$). For each $i \in \{1, \dots, \log n\}$, T_i is a spanning binomial arborescence. Therefore, the forward direction follows. This proof also indicates how to construct a solution in polynomial time.

In the other direction, suppose that (D, v^*, ℓ) is a YES-instance. Let (R, T) be a solution that satisfies property 1 in Theorem 28. Let X' be the set of children v of v^* in T such that $(v^*, v) \in A(D)$. Since $|R| \leq \ell$, we have that $|X'| \geq \log n - \ell$, and we denote by X some subset of X' of size exactly $\log n - \ell$. Order $X = \{x_1, \dots, x_{\log n - \ell}\}$ such that $|V(T_{x_i})| \leq |V(T_{x_{i+1}})|$ for all $i \in \{1, \dots, \log n - \ell - 1\}$. Since (R, T) satisfies property 1, we know that T_{x_i} is a spanning binomial arborescence of $D[V(T_{x_i})]$ for all $i \in \{1, \dots, \log n - \ell\}$. By Observation 9.2.1, it can be verified that there exists a subtree T'_{x_i} of T_{x_i} on exactly 2^{i-1} vertices that is a spanning binomial arborescence of $D[V(T'_{x_i})]$ rooted at x_i (the idea is to truncate the subtrees of the children of x_i that are largest). Now, denote $U = V(T'_{x_1}) \cup \dots \cup V(T'_{x_{\log n - \ell}})$. Observe that $|U| = \sum_{i=1}^{\log n - \ell} |V(T'_{x_i})| = \sum_{i=1}^{\log n - \ell} 2^{i-1} = \frac{n}{2^\ell} - 1$. Thus, by taking all the arborescences T'_{x_i} and adding the arcs (v^*, x_i) (which belong to $A(D)$ by our choice of X), we exhibit a spanning binomial arborescence of $D[U \cup \{v^*\}]$ that is rooted at v^* . $\diamond$

In our proof of the forward direction above, it can be shown that v^* is not necessarily made a king by reversing ℓ arcs. As a corollary of Theorem 31, we have the following one-way statement (which is constructive).

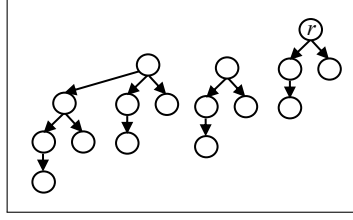


Figure 9.6: A 2-removed binomial arborescence on 2^4 vertices.

Theorem 32. *Let (D, v^*, ℓ) be an instance of BTF. If $\text{victory}(v^*) \geq \frac{n}{2^\ell} - 1$, then (D, v^*, ℓ) is a YES-instance. In this case, a solution can be found in polynomial time.*

Proof. Suppose that $\text{victory}(v^*) \geq \frac{n}{2^\ell} - 1$, and let U be a set of exactly $\frac{n}{2^\ell} - 1$ vertices beaten by v^* . We fix a tournament (competition) for $D[U \cup \{v^*\}]$, and let T denote the corresponding spanning binomial arborescence of $D[U \cup \{v^*\}]$. Because v^* beats every vertex in U , it must be the root of T . By the reverse direction of Theorem [31](#), the proof is complete. $\diamond$

9.7 Exact Algorithms

Our exact algorithms are based on translating the characterization in Theorem [31](#) into the language of spanning binomial arborescences. To this end, we introduce a new definition (see Fig. [9.6](#)). The validity of the sizes mentioned in this definition follows from Observation [9.2.1](#).

Definition 9.7.1. *Let $\ell \in \mathbb{N}$. Let T be a binomial arborescence of size n with $\log n \geq \ell$. Let the children of its root r be v_i , $i \in [\log n]$, where $|V(T_{v_i})| = 2^{i-1}$ for all $i \in \{1, \dots, \log n\}$. An ℓ -removed binomial arborescence T' of size n is the digraph obtained from T by deleting the arcs (r, v_i) for all $i \in \{\log n - \ell + 1, \dots, \log n\}$. The vertex r is the root of T' .^{[2](#)}*

²Although T' contains $\ell + 1$ vertices of in-degree 0, there is only one vertex that we define as the root.

Note that up to isomorphism, an ℓ -removed binomial arborescence of size n is unique. We now give the promised translation of Theorem [31](#).

Lemma 9.7.1. *Let (D, w, ℓ) be an instance of BTF. Then, (D, w, ℓ) is a YES-instance if and only if D contains a spanning ℓ -removed binomial arborescence T rooted at w .*

Proof. In one direction, suppose that D contains a spanning ℓ -removed binomial arborescence T rooted at w . Let T' denote the tree of T that is rooted at w . Then, $|V(T') \setminus \{w\}| = n - 1 - \sum_{i=\log n - \ell + 1}^{\log n} 2^{i-1} = \frac{n}{2^\ell} - 1$ (by Observation [9.2.1](#)) and T' is a spanning binomial arborescence of $D[V(T')]$ rooted at w . Denoting $U = V(T') \setminus \{w\}$, the claim follows from the reverse direction of Theorem [31](#).

In the other direction, suppose that (D, w, ℓ) is a YES-instance. By the forward direction of Theorem [31](#), there exists $U \subseteq V(D)$ of size exactly $\frac{n}{2^\ell} - 1$ so that there is a fixing of $D[U \cup \{w\}]$ that makes w win. Let T' be a spanning binomial arborescence of $D[U \cup \{w\}]$ rooted at w . Arbitrarily partition $V(D) \setminus (U \cup \{w\})$ into ℓ sets, $U_{\log n - \ell + 1}, \dots, U_{\log n}$, such that $|U_i| = 2^{i-1}$ for all $i \in \{\log n - \ell + 1, \dots, \log n\}$. For all $i \in \{\log n - \ell + 1, \dots, \log n\}$, fix a tournament (competition) for $D[U_i]$, and let T_i denote the corresponding spanning binomial arborescence of $D[U_i]$ rooted at some vertex w_i . Define T by $V(T) = V(T_{\log n - \ell + 1}) \cup \dots \cup V(T_{\log n}) \cup V(T')$ and $A(T) = A(T_{\log n - \ell + 1}) \cup \dots \cup A(T_{\log n}) \cup A(T')$, and let w be its root. Then, it is easy to verify that T is a spanning ℓ -removed binomial arborescence of D rooted at w . $\diamond$

Towards the presentation of our first algorithm, we show how to detect ℓ -removed binomial arborescences.

Lemma 9.7.2. *Let D be a digraph, $w \in V(D)$, and $\ell \in \mathbb{N}$. We can decide in time $2^n n^{\mathcal{O}(1)}$ and polynomial space if D contains a spanning ℓ -removed binomial arborescence T rooted at w . If the answer is positive, we output T .*

Proof. Amini et al. [6] proved the following result. Suppose that we have two digraphs, D on n vertices and H on k vertices, such that the treewidth of the underlying undirected graph of H is t . Moreover, suppose that each vertex in D has a color from a set of k colors. Then, it can be decided in time $2^k n^{t+\mathcal{O}(1)}$ and polynomial space if D has a colorful subgraph isomorphic to H . This algorithm can be immediately adapted to also solve the case where each digraph among D and H has a distinguished vertex, say, d and h , so that the isomorphism must match d and h . In our setting, H is the (unique) ℓ -removed binomial arborescence T on n vertices (that is, $k = n$), whose underlying undirected graph is a forest and hence has treewidth 1. The coloring is simply the identity function (that is, every vertex in D gets a unique color), $d = w$ and h is the root of T . As the algorithm in [6] also outputs a copy of H in D (if one exists) in the same running time bound, the correctness of our lemma follows. $\diamond$

Having Lemmata [9.7.1] and [9.7.2], we are ready to present our exponential-time algorithm.

Theorem 33. *BTF can be solved in time $2^n n^{\mathcal{O}(1)}$ and polynomial space. In the case of a YES-instance, a solution can be found in the same running time bound.*

Proof. Given an instance (D, w, ℓ) of BTF, call the algorithm in Lemma [9.7.2] to decide if D contains an ℓ -removed binomial arborescence T' of size n rooted at w . If the answer is negative, return NO. Else, let T' be the output. Define T as the binomial arborescence obtained from T' by adding arcs from w to all other vertices in T' of in-degree 0. Moreover, define $R = (A(T) \setminus A(D))^{\text{rev}}$, and observe that $|R| \leq \ell$. The correctness of the algorithm then follows from Lemma [9.7.1]. $\diamond$

Replacing Lemma [9.7.2] by Theorem [27] proves the following.

Theorem 34. *BTF can be solved in time $2^{\mathcal{O}(k^* \log k^*)} n^{\mathcal{O}(1)}$ and polynomial space,*

where k^* is the feedback arc set number of the input tournament. In case of a YES-instance, a solution can be found in the same running time bound.

9.8 Conclusion

In this chapter, we studied BTF and gave combinatorial results, polynomial-time algorithms for special cases, and exact exponential-time algorithms for the general case. One of our results imply that the problem is FPT parameterized by k^* , the size of a feedback arc set of the input tournaments. A natural question, therefore, is whether the problem admits a polynomial kernel. Another open question is whether BTF (or even TOURNAMENT FIXING) is FPT when parameterized by the size of a feedback vertex set of the input tournament.

Bibliography

- [1] D. J. Abraham, P. Biró, and D. F. Manlove. ““Almost stable” matchings in the roommates problem”. In: *WOAO*. 2006, pp. 1–14.
- [2] D. J. Abraham, A. Blum, and T. Sandholm. “Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges”. In: *EC*. 2007, pp. 295–304.
- [3] F. N. Abu-Khzam. “An improved kernelization algorithm for r -set packing”. In: *Information Processing Letters* 110 (2010), pp. 621–624.
- [4] R. Aharoni and E. Berger. “Rainbow matchings in r -partite r -graphs”. In: *The Electronic Journal of Combinatorics* 16.1 (2009), p. 119.
- [5] N. Alon. “Multicolored matchings in hypergraphs”. In: *Moscow Journal of Combinatorics and Number Theory* 1 (2011), pp. 3–10.
- [6] O. Amini, F. V. Fomin, and S. Saurabh. “Counting subgraphs via homomorphisms”. In: *SIAM Journal on Discrete Mathematics* 26.2 (2012), pp. 695–717.
- [7] H. Aziz, E. Elkind, P. Faliszewski, M. Lackner, and P. Skowron. “The condorcet principle for multiwinner elections: from shortlisting to proportionality”. In: *IJCAI*. 2017, pp. 84–90.
- [8] H. Aziz, E. Elkind, P. Faliszewski, M. Lackner, and P. Skowron. “The condorcet principle for multiwinner elections: from shortlisting to proportionality”. In: *CoRR* abs/1701.08023 (2017).

- [9] H. Aziz, S. Gaspers, S. Mackenzie, N. Mattei, P. Stursberg, and T. Walsh. “Fixing a balanced knockout tournament”. In: *AAAI (Accepted to Artificial Intelligence Journal (AIJ).)* 2014, pp. 552–558.
- [10] H. Aziz., R. Savani, and H. Moulin. “Hedonic games. chapter 15 in: handbook of computational social choice”. In: *Handbook of Computational Social Choice*. Ed. by F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D Procaccia. Cambridge University Press, 2016, pp. 356–376.
- [11] R. Bar-Yehuda and S. Even. “A linear-time approximation algorithm for the weighted vertex cover problem”. In: *Journal of Algorithms* 2.2 (1981), pp. 198–203.
- [12] N. Betzler. “A multivariate complexity analysis of voting problems”. PhD thesis. Friedrich-Schiller-Universität Jena, 2010.
- [13] N. Betzler, R. Brederbeck, J. Chen, and R. Niedermeier. “Studies in computational aspects of voting - A parameterized complexity perspective”. In: *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*. 2012, pp. 318–363.
- [14] T. Beyer and S. M. Hedetniemi. “Constant time generation of rooted trees”. In: *SIAM Journal on Computing* 9.4 (1980), pp. 706–712.
- [15] P. Biró, D. F. Manlove, and E. J. McDermid. ““Almost stable” matchings in the roommates problem with bounded preference list”. In: *Theoretical Computer Science* 432 (2012), pp. 10–20.
- [16] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. “Narrow sieves for parameterized paths and packings”. In: *Journal of Computer and System Sciences* 87 (2017), pp. 119–139.
- [17] B. Bliem, R. Brederbeck, and R. Niedermeier. “Complexity of efficient and envy-free resource allocation: few agents, resources, or utility levels”. In: *IJCAI*. New York, New York, USA, 2016, pp. 102–108. ISBN: 978-1-57735-770-4.

- [18] H. L. Bodlaender. “A linear-time algorithm for finding tree-decompositions of small treewidth”. In: *SIAM Journal on Computing* 25.6 (1996), pp. 1305–1317.
- [19] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. “A $c^k n^5$ -approximation algorithm for treewidth”. In: *SIAM Journal on Computing* 45.2 (2016), pp. 317–378.
- [20] F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia. *Handbook of computational social choice*. English. London: Cambridge University Press, 2016. ISBN: 9781107060432.
- [21] R. Bredereck, J. Chen, P. Faliszewski, J. Guo, R. Niedermeier, and G. Woeginger. “Parameterized algorithmics for computational social choice: nine research challenges”. In: *Tsinghua Science and Technology* 19.4, 358 (2014), pp. 358–373.
- [22] G. Chalkiadakis, G. Greco, and E. Markakis. “Characteristic function games with restricted agent interactions: core-stability and coalition structures”. In: *Artificial Intelligence* 232 (2016), pp. 76–113.
- [23] J. Chen, D. Hermelin, M. Sorge, and H. Yedidsion. “How hard is it to satisfy (almost) all roommates?” In: *ICALP*. 2018, 35:1–35:15.
- [24] J. Chen, I. A. Kanj, and G. Xia. “Improved upper bounds for vertex cover”. In: *Theoretical Computer Science* 411.40-42 (2010), pp. 3736–3756.
- [25] D. Coelho. *Understanding, evaluating and selecting voting rules through games and axioms*. Universitat Autònoma de Barcelona, 2005.
- [26] M. D. Condorcet. “Essai sur l’application de l’analyse, a la probabillite des decisions rendues a la pluralite des voix”. In: (1785).
- [27] R. A. Connolly and R. J. Rendleman. “Tournament qualification, seeding and selection efficiency”. In: *Technical report 2011-96, tuck school of business* (2011).

- [28] A. Cseh and D. F. Manlove. “Stable marriage and roommates problems with restricted edges : complexity and approximability”. In: *SAGT*. Vol. 9347. LNCS. 2015, pp. 15–26.
- [29] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Springer, 2015.
- [30] M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. “On multiway cut parameterized above lower bounds”. In: *ACM Transactions on Algorithms* 5.1 (2013), 3:1–3:11.
- [31] A. Darmann. “How hard is it to tell which is a condorcet committee?” In: *Mathematical Social Sciences* 66.3 (2013), pp. 282–292.
- [32] A. Darmann, E. Elkind, S. Kurz, J. Lang, J. Schauer, and G. Woeginger. “Group activity selection problem”. In: *WINE*. Springer, 2012, pp. 156–169.
- [33] H. Dell and D. Marx. “Kernelization of packing problems”. In: *SODA*. 2012.
- [34] J. Diemunsch, M. Ferrara, C. Moffatt, F. Pfender, and P. S. Wenger. “Rainbow matchings of size $\delta(G)$ in properly edge-colored graphs”. In: *arXiv preprint arXiv:1108.2521* (2011).
- [35] R. Diestel. *Graph theory, 4th edition*. Vol. 173. Graduate texts in mathematics. Springer, 2012.
- [36] R. G. Downey and M. R. Fellows. “Fixed-parameter tractability and completeness II: on completeness for $W[1]$ ”. In: *Theoretical Computer Science* 141.1&2 (1995), pp. 109–131.
- [37] R. G. Downey and M. R. Fellows. *Fundamentals of parameterized complexity*. Texts in Computer Science. Springer, 2013.
- [38] E. A. Durant. *Hearing aids and methods and apparatus for audio fitting thereof*. US Patent 7,650,004. 19 2010.

- [39] J. Edmonds. “Paths, trees, and flowers”. In: *Canadian Journal of Mathematics* 17.3 (1965), pp. 449–467.
- [40] E. Elkind. “Coalitional games on sparse social networks”. In: *WINE*. 2014, pp. 308–321.
- [41] E. Elkind, P. Faliszewski, P. Skowron, and A. Slinko. “Properties of multi-winner voting rules”. In: *Social Choice and Welfare* 48.3 (2017), pp. 599–632.
- [42] E. Elkind, J. Lang, and A. Saffidine. “Condorcet winning sets”. In: *Social Choice and Welfare* 44.3 (2015), pp. 493–517.
- [43] P. Erdős and L. Moser. “On the representation of directed graphs as unions of orderings”. In: *Mathematical Institute of the Hungarian Academy of Sciences* 9 (1964), pp. 125–132.
- [44] P. Faliszewski and R. Niedermeier. “Parameterization in computational social choice”. In: *Encyclopedia of Algorithms*. 2016, pp. 1516–1520.
- [45] P. Faliszewski, P. Skowron, A. Slinko, and N. Talmon. “Multiwinner voting: a new challenge for social choice theory”. In: *Trends in Computational Social Choice* 74 (2017).
- [46] T. Feder. “Stable networks and product graphs”. PhD thesis. Stanford University, 1990.
- [47] U. Feige. “Faster FAST (Feedback Arc Set in Tournaments)”. In: *Corr abs/0911.5094* (2009).
- [48] P. C. Fishburn. “An analysis of simple voting systems for electing committees”. In: *SIAM Journal on Applied Mathematics* 41.3 (1981), pp. 499–502.
- [49] F. V. Fomin, F. Grandoni, and D. Kratsch. “A measure & conquer approach for the analysis of exact algorithms”. In: *Journal of the ACM* 56.5 (2009), 25:1–25:32.

- [50] F. V. Fomin and D. Kratsch. *Exact exponential algorithms*. Springer Science & Business Media, 2010.
- [51] F. V. Fomin, D. Lokshtanov, F. Panolan, and S. Saurabh. “Efficient computation of representative families with applications in parameterized and exact algorithms”. In: *Journal of the ACM* 63.4 (2016), 29:1–29:60.
- [52] F. V. Fomin, D. Lokshtanov, F. Panolan, and S. Saurabh. “Representative sets of product families”. In: *ESA*. Vol. 8737. Lecture Notes in Computer Science. Springer, 2014, pp. 443–454.
- [53] F. V. Fomin, D. Lokshtanov, V. Raman, S. Saurabh, and B. V. Raghavendra Rao. “Faster algorithms for finding and counting subgraphs”. In: *Journal of Computer and System Sciences* 78.3 (2012), pp. 698–706.
- [54] F. V. Fomin, D. Lokshtanov, S. Saurabh, and M. Zehavi. *Kernelization: Theory of parameterized preprocessing*. Cambridge University Press, 2018.
- [55] S. Fujita, A. Kaneko, I. Schiermeyer, and K. Suzuki. “A rainbow k -matching in the complete graph with r colors”. In: *The Electronic Journal of Combinatorics* 16.1 (2009), p. 51.
- [56] M. Gairing and R. Savani. “Computing stable outcomes in hedonic games”. In: *SAGT*. Vol. 6386. LNCS. Springer, 2010, pp. 174–185.
- [57] M. Gairing and R. Savani. “Computing stable outcomes in hedonic games with voting-based deviations”. In: *AAMAS*. 2011, pp. 559–566.
- [58] D. Gale and L. S. Shapley. “College admissions and the stability of marriage”. In: *The American Mathematical Monthly* 69.1 (1962), pp. 9–15.
- [59] D. Gale and M. Sotomayor. “Ms. machiavelli and the gale-shapley algorithm”. In: *American Mathematical Monthly* 92.4 (1985), pp. 261–268.
- [60] D. Gale and M. Sotomayor. “Some remarks on the stable matching problem”. In: *Discrete Applied Mathematics* 11.4 (1985), pp. 223–232.

- [61] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman, 1979.
- [62] M. R. Garey and D. S. Johnson. “The rectilinear steiner tree problem is NP-complete”. In: *SIAM Journal on Applied Mathematics* 32 (1977), pp. 826–834.
- [63] S. Garg and G. Philip. “Raising the bar for vertex cover: fixed-parameter tractability above a higher guarantee”. In: *SODA*. 2016, pp. 1152–1166.
- [64] W. V. Gehrlein. “The condorcet criterion and committee selection”. In: *Mathematical Social Sciences* 10.3 (1985), pp. 199–209.
- [65] I. P. Gent and P. Prosser. “An empirical study of the stable marriage problem with ties and incomplete lists”. In: *ECAI*. IOS Press. 2002, pp. 141–145.
- [66] R. Glebov, B. Sudakov, and T. Szabó. “How many colors guarantee a rainbow matching?” In: *arXiv preprint arXiv:1211.0793* (2012).
- [67] C. Groh, B. Moldovanu, A. Sela, and U. Sunde. “Optimal seedings in elimination tournaments”. In: *Economic theory* 49.1 (2012), pp. 59–80.
- [68] S. Gupta, S. Roy, S. Saurabh, and M. Zehavi. “Group activity selection on graphs: parameterized analysis”. In: *SAGT*. 2017, pp. 106–118.
- [69] S. Gupta, S. Saurabh, and M. Zehavi. “On treewidth and stable marriage”. In: *CoRR* abs/1707.05404 (2017).
- [70] D. Gusfield. “Three fast algorithms for four problems in stable marriage”. In: *SIAM Journal on Computing* 16.1 (1987).
- [71] D. Gusfield and R. W. Irving. *The stable marriage problem - structure and algorithms*. Foundations of computing series. MIT Press, 1989. ISBN: 978-0-262-07118-5.
- [72] K. Hamada, K. Iwama, and S. Miyazaki. “The hospitals/residents problem with quota lower bounds”. In: *ESA*. 2011, pp. 180–191.

- [73] P. Hell and M. Rosenfeld. “The complexity of finding generalized paths in tournaments”. In: *Journal of Algorithms* 4.4 (1983), pp. 303–309.
- [74] J. E. Hopcroft and R. M. Karp. “An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs”. In: *SIAM Journal on Computing* 2 (1973), pp. 225–231.
- [75] J. Horen and R. Riezman. “Comparing draws for single elimination tournaments”. In: *Operations Research* 33.2 (1985), pp. 249–262.
- [76] J. D. Horton and K. Kilakos. “Minimum edge dominating sets”. In: *SIAM Journal of Discrete Mathematics* 6.3 (1993), pp. 375–387.
- [77] A. Igarashi, R. Brederick, and E. Elkind. “On parameterized complexity of group activity selection problems on social networks”. In: *AAMAS*. 2017, pp. 1575–1577.
- [78] A. Igarashi and E. Elkind. “Hedonic games with graph-restricted communication”. In: *AAMAS*. 2016, pp. 242–250.
- [79] A. Igarashi, D. Peters, and E. Elkind. “Group activity selection on social networks”. In: *arXiv preprint arXiv:1611.04524[Appeared in AAAI, 2017]* (2016).
- [80] A. Igarashi, D. Peters, and E. Elkind. “Group activity selection on social networks”. In: *AAAI*. 2017, pp. 565–571.
- [81] R. Impagliazzo and R. Paturi. “On the complexity of k-sat”. In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 367–375.
- [82] T. Inoshita, R. W. Irving, K. Iwama, S. Miyazaki, and T. Nagase. “Improving man-optimal stable matchings by minimum changes to preference lists”. In: *Algorithms* 6.2 (2013), pp. 371–382.
- [83] R. W. Irving. “An efficient algorithm for the “stable roommates” problem”. In: *Journal of Algorithms* 6.4 (1985), pp. 577–595. ISSN: 0196-6774.

- [84] R. W. Irving. “Stable marriage and indifference”. In: *Discrete Applied Mathematics* 48.3 (1994), pp. 261–272.
- [85] R. W. Irving, K. Iwama, D. F. Manlove, S. Miyazaki, and Y. Morita. “Hard variants of stable marriage”. In: *Theoretical Computer Science* 276.1-2 (2002), pp. 261–279.
- [86] R. W. Irving, P. Leather, and D. Gusfield. “An efficient algorithm for the “optimal” stable marriage”. In: *Journal of the ACM* 34.3 (1987), pp. 532–543.
- [87] R. W. Irving, D. F. Manlove, and G. O’Malley. “Stable marriage with ties and bounded length preference lists”. In: *Journal of Discrete Algorithms* 7.2 (2009), pp. 213–219.
- [88] A. Itai, M. Rodeh, and S. L. Tanimoto. “Some matching problems for bipartite graphs”. In: *Journal of the ACM* 25.4 (1978), pp. 517–525.
- [89] B. M. P. Jansen. “Polynomial kernels for hard problems on disk graphs”. In: *SWAT*. Vol. 6139. LNCS. Springer, 2010, pp. 310–321.
- [90] J. B. Jensen and P. Hell. “Fast algorithms for finding hamiltonian paths and cycles in in-tournament digraphs”. In: *Discrete Applied Mathematics* 41.1 (1993), pp. 75–79.
- [91] E. Kamwa. “On stable rules for selecting committees”. In: *Journal of Mathematical Economics* 70 (2017), pp. 36–44.
- [92] M. Kano and X. Li. “Monochromatic and heterochromatic subgraphs in edge-colored graphs-a survey”. In: *Graphs and Combinatorics* 24.4 (2008), pp. 237–263.
- [93] M. Karpinski and W. Schudy. “Faster algorithms for feedback arc set tournament, Kemeny rank aggregation and betweenness tournament”. In: *ISAAC*. 2010, pp. 3–14.

- [94] A. Kato. “Complexity of the sex-equal stable marriage problem”. In: *Japan Journal of Industrial and Applied Mathematics* 10.1 (1993), p. 1. ISSN: 1868-937X.
- [95] B. Kaymak and M. R. Sanver. “Sets of alternatives as condorcet winners”. In: *Social Choice and Welfare* 20.3 (2003), pp. 477–494.
- [96] M. P. Kim, W. Suksompong, and V. V. Williams. “Who can win a single-elimination tournament?” In: *AAAI*. 2016, pp. 516–522.
- [97] M. P. Kim and V. V. Williams. “Fixing tournaments for kings, chokers, and more”. In: *IJCAI*. 2015, pp. 561–567.
- [98] J. M. Kleinberg and É. Tardos. *Algorithm design*. Addison-Wesley, 2006. ISBN: 978-0-321-37291-8.
- [99] D. E. Knuth. *Stable marriage and its relation to other combinatorial problems : an introduction to the mathematical analysis of algorithms*. CRM proceedings & lecture notes. Providence, R.I. American Mathematical Society, 1997. ISBN: 0-8218-0603-3.
- [100] H. Kobayashi and T. Matsui. “Cheating strategies for the gale-shapley algorithm with complete preference lists”. In: *Algorithmica* 58 (2010), pp. 151–169.
- [101] H. Kobayashi and T. Matsui. “Successful manipulation in stable marriage model with complete preference”. In: *IEICE TRANSACTIONS on Information and Systems* E92-D.2 (2009), pp. 116–119.
- [102] A. Kostochka and M. Yancey. “Large rainbow matchings in edge-coloured graphs”. In: *Combinatorics, Probability and Computing* 21.1-2 (2012), pp. 255–263.
- [103] J. Lang, M. S. Pini, F. Rossi, D. Salvagnin, K. B. Venable, and T. Walsh. “Winner determination in voting trees with incomplete preferences and weighted votes”. In: *AAMAS* 25.1 (2012), pp. 130–157.

- [104] J.F. Laslier. *Tournament solutions and majority voting*. Studies in Economic Theory. Springer Berlin Heidelberg, 2011. ISBN: 9783642645617.
- [105] V. B. Le and F. Pfender. “Complexity results for rainbow matchings”. In: *Theoretical Computer Science* 524 (2014), pp. 27–33.
- [106] H. Lee and V. Williams. “Complexity of the stable invitations problem”. In: *AAAI*. 2017, pp. 579–585.
- [107] H. Lee and V. Williams. “Parameterized complexity of group activity selection”. In: *AAMAS*. 2017, pp. 353–361.
- [108] T. D. LeSaulnier, C. Stocker, P. S. Wenger, and D. B. West. “Rainbow matching in edge-colored graphs”. In: *The Electronic Journal of Combinatorics* 17.1 (2010), p. 26.
- [109] R. J. Lipton and R. E. Tarjan. “A separator theorem for planar graphs”. In: *SIAM Journal on Applied Mathematics* 36.2 (1979), pp. 177–189.
- [110] D. S. Lita. *Method and apparatus for managing billiard tournaments*. US Patent App. 11/789,667. 30 2008.
- [111] A. Lo. “Existences of rainbow matchings and rainbow matching covers”. In: *Discrete Mathematics* 338.11 (2015), pp. 2119–2124.
- [112] D. Lokshtanov, N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh. “Faster parameterized algorithms using linear programming”. In: *TALG* 11.2 (2014), 15:1–15:31.
- [113] L. Lovász and M. D. Plummer. *Matching theory*. Vol. 367. American Mathematical Society, 2009.
- [114] D. F. Manlove. *Algorithmics of matching under preferences*. Vol. 2. Theoretical Computer Science. World Scientific, 2013.
- [115] D. F. Manlove. “The structure of stable marriage with indifference”. In: *Discrete Applied Mathematics* 122.1-3 (2002), pp. 167–181.

- [116] D. Marx and I. Schlotter. “Parameterized complexity and local search approaches for the stable marriage problem with ties”. In: *Algorithmica* 58.1 (2010), pp. 170–187.
- [117] D. Marx and I. Schlotter. “Stable assignment with couples: parameterized complexity and local search”. In: *Discrete Optimization* 8.1 (2011), pp. 25–40.
- [118] C. K. Mathieu and W. Schudy. “How to rank with few errors”. In: *STOC*. 2007, pp. 95–103.
- [119] N. Mattei, J. Goldsmith, A. Klapper, and M. Mundhenk. “On the complexity of bribery and manipulation in tournaments with uncertain information”. In: *Journal of Applied Logic* 13.4 (2015), pp. 557–581.
- [120] N. Mattei and T. Walsh. “Empirical evaluation of real world tournaments”. In: *Corr abs/1608.01039* (2016).
- [121] E. McDermid. “A $3/2$ -approximation algorithm for general stable marriage”. In: *ICALP*. Springer. 2009, pp. 689–700.
- [122] E. McDermid. “Personal communications between mcdermid and manlove”. In: 2010.
- [123] E. McDermid and R. W. Irving. “Sex-equal stable matchings: complexity and exact algorithms”. In: *Algorithmica* 68.3 (2014), pp. 545–570.
- [124] K. Meeks and B. Rastegari. “Stable marriage with groups of similar agents”. In: *WINE*. 2018, pp. 312–326.
- [125] S. Micali and V. V. Vazirani. “An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs”. In: *FOCS*. 1980, pp. 17–27.
- [126] M. Mnich and I. Schlotter. “Stable marriage with covering constraints—a complete computational trichotomy”. In: *SAGT*. 2017, pp. 320–332.

- [127] D. Munera, D. Diaz, S. Abreu, F. Rossi, V. Saraswat, and P. Codognet. “Solving hard stable matching problems via local search and cooperative parallelization”. In: *AAAI*. 2015.
- [128] R. B. Myerson. “Graphs and cooperation games”. In: *Mathematics of Operations Research* (1977).
- [129] G. O’Malley. “Algorithmic aspects of stable matching problems”. PhD thesis. University of Glasgow, 2007.
- [130] R. Otter. “The number of trees”. In: *Annals of Mathematics* (1948), pp. 583–599.
- [131] D. Peters. “Graphical hedonic games of bounded treewidth”. In: *AAAI*. 2016, pp. 586–593.
- [132] M. S. Pini, F. Rossi, B. Venable, and T. Walsh. “Stable marriage problems with quantitative preferences”. In: *arXiv preprint arXiv:1007.5120* (2010).
- [133] A. Pokrovskiy. “An approximate version of a conjecture of aharoni and berger”. In: *Advances in Mathematics* 333 (2018), pp. 1197–1241.
- [134] V. Raman, M. S. Ramanujan, and S. Saurabh. “Paths, flowers and vertex cover”. In: *ESA*. 2011, pp. 382–393.
- [135] M. S. Ramanujan and S. Szeider. “Rigging nearly acyclic tournaments is fixed-parameter tractable”. In: *AAAI*. 2017, pp. 3929–3935.
- [136] N. Robertson, P. D. Seymour, and R. Thomas. “Quickly excluding a planar graph”. In: *Journal of Combinatorial Theory, Series B* 62.2 (1994), pp. 323–348.
- [137] E. Ronn. “NP-complete stable matching problem”. In: *Journal of Algorithms* 11 (1990), pp. 285–304.
- [138] S. Rosen. “Prizes and incentives in elimination tournaments”. In: *The American Economic Review* 76.4 (1986), pp. 701–715.

- [139] A. E. Roth and M. A. O. Sotomayor. *Two-sided matching: a study in game-theoretic modeling and analysis*. Business & Economics 18. Cambridge University Press, 1992.
- [140] T. Russell and P. V. Beek. “An empirical study of seeding manipulations and their prevention”. In: *IJCAI*. 2011, pp. 350–356.
- [141] T. Russell and T. Walsh. “Manipulating tournaments in cup and round robin competitions”. In: *ADT*. 2009, pp. 26–37.
- [142] H. J. Ryser. “Neuere probleme der kombinatorik”. In: *Vorträge über Kombinatorik, Oberwolfach* (1967), pp. 69–91.
- [143] D. Ryvkin. “The selection efficiency of tournaments”. In: *European Journal of Operational Research* 206.3 (2010), pp. 667–675.
- [144] S. Salvador and D. Coelho. “How to choose a non-controversial list with k names”. In: *Social Choice and Welfare* 31.1 (2008), pp. 79–96.
- [145] I. Stanton and V. V. Williams. “Manipulating single-elimination tournaments in the braverman-mossel model”. In: *IJCAI Workshop on Social Choice and Artificial Intelligence*. 2011.
- [146] I. Stanton and V. V. Williams. “Manipulating stochastically generated single-elimination tournaments for nearly all players”. In: *WINE*. 2011, pp. 326–337.
- [147] I. Stanton and V. V. Williams. “Rigging tournament brackets for weaker players”. In: *IJCAI*. 2011, pp. 357–364.
- [148] L. J. Stockmeyer and V. V. Vazirani. “NP-completeness of some generalizations of the maximum matching problem”. In: *Information Processing Letters* 15.1 (1982), pp. 14–19.
- [149] *Trends in computational social choice*. AI Access, 2017.

- [150] G. Tullock. “Toward a theory of the rent-seeking society”. In: *Texas A&M University Press* (1980).
- [151] T. Vu, A. Altman, and Y. Shoham. “On the complexity of schedule control problems for knockout tournaments”. In: *AAMAS*. 2009, pp. 225–232.
- [152] K. Wagner. “Uber eine eigenschaft der ebenen komplexe”. In: *Annals of Mathematics* 114 (1937), pp. 570–590.
- [153] G. Wang. “Rainbow matchings in properly edge colored graphs”. In: *The Electronic Journal of Combinatorics* 18.1 (2011), p. 162.
- [154] G. Wang and H. Li. “Heterochromatic matchings in edge-colored graphs”. In: *The Electronic Journal of Combinatorics* 15.1 (2008), p. 138.
- [155] V. V. Williams. “Fixing a tournament”. In: *AAAI*. 2010.
- [156] V. V. Williams. “Knockout tournaments”. In: *Handbook of computational social choice*. Cambridge University Press, 2016, pp. 453–474.
- [157] M. Yannakakis and F. Gavril. “Edge dominating sets in graphs”. In: *SIAM Journal on Applied Mathematics* 38.3 (1980), pp. 364–372.
- [158] M. Zehavi. “Mixing color coding-related techniques”. In: *ESA*. Vol. 9294. Lecture Notes in Computer Science. 2015, pp. 1037–1049.