

Dynamic Programming using Representative Families

By

Fahad P

MATH10201005007

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



July, 2015

Homi Bhabha National Institute

Recommendations of the Viva Voce Board

As members of the Viva Voce Board, we certify that we have read the dissertation prepared by **Fahad P** entitled “**Dynamic Programming using Representative Families**” and recommend that it may be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

_____ Date:
Chair - V. Arvind

_____ Date:
Guide/Convener - Saket Saurabh

_____ Date:
Co-Guide/Member 1 - Venkatesh Raman

_____ Date:
Member 2 - Sourav Chakraborty

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to HBNI.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it may be accepted as fulfilling the dissertation requirement.

Date:

Place:

Guide

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.

Fahad P

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.

Fahad P

DEDICATIONS

This thesis is dedicated to my brother Faisal, my parents, my wife Jeshma and my son Ehan.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Prof. Saket Saurabh for being there through thick and thin for the completion of my PhD. His timely advice and suggestions on both research as well as my career was very insightful. I was extremely lucky to have him as a driving force supporting me both academically and emotionally throughout the completion of this thesis. I appreciate his willingness to have discussions and meetings at short notice every time and going through several drafts of my thesis as well as conference papers. He has always been an inspiration to me. I am indebted to my co-advisor Prof. Venkatesh Raman for his advice and support towards the completion of this thesis. I thank you for your comments and suggestions.

It would have been impossible to finish my dissertation without the support and guidance from my committee members. Their support and guidance has been the stepping stones for the perfect completion of this thesis. Besides my advisor and co-advisor a special thanks goes to Prof. V. Arvind for his encouragement in my work.

It is an honor for me to have visited Prof. Fedor V Fomin at the University of Bergen. I thank him for giving me this delightful opportunity. The stay at Bergen was one of the most memorable incidents in my life. It is a pleasure to have worked with him and his group at University of Bergen. The brainstorming discussions and the insightful meetings with him have enlightened my research.

I thank all the faculties at Theoretical Computer Science Group, IMSc for laying the foundation of my research carrier. I thank them for the courses they offered which helped me improve my knowledge in the area. I thank Prof. K. Muralikrishnan for his guidance during my MCA and encouraging me to do PhD.

I thank Neeldhara Misra who was an amazing source of support and energy. Her unconditional support from early period of my PhD onwards helped me a lot in completion of this thesis. I was fortunate enough to have met Prof. Petteri Kaski and Daniel Lokshtanov. The discussions and sessions I had with them was inspirational and enlightening. I also thank Geevarghese Philip for his support.

I am deeply obliged to Prof. Gregory Gutin at Royal Holloway London for giving me an opportunity to visit him and work with his group. I thank him for this wonderful experience. I thank Prof. Kurt Mehlhorn for giving me an opportunity to visit the parameterized complexity group at MPI, Saarbrücken.

I acknowledge and thank my co-authors Manu Basavaraju, Esha Ghosh, Petr Golovach, Prachi Goyal, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Amer E. Mouawad, Ashutosh Rai, M. S. Ramanujan and Meirav zehavi.

I also thank all my friends at IMSc and CMI for supporting me during this amaz-

ing journey of five years. I thank my friends and faculties in algorithm group at University of Bergen for making my stay at Bergen eventful.

There are no words to express my feelings I have for my beloved parents. Their unconditional love and support was the driving force for me throughout my life. I would like to thank my very dear brother Faisal for extending his whole hearted support for the continuation of my education. Without his financial aid, I could not have completed my education and reached till here. I do not know how to repay you for all the sacrifice and pain you went through. When I needed help, you were the only one to stretch out a helping hand for me to hold on to.

I thank my wife, Jeshma for being there with me throughout this entire journey of ups and downs. Thank You for being there for me. Thank You for being a wonderful mother to our son Ehan.

Contents

Synopsis	vii
List of Publications from the Thesis	ix
List of Figures	xi
List of Tables	xiii
I Introduction	1
1 Computational Framework	3
1.1 Fixed Parameter Tractability	4
1.2 Exact Exponential Time Algorithms	7
1.3 Randomized Algorithms	7
2 Organization of the thesis	9
3 Preliminaries	11
3.1 Sets, Families and Functions	11
3.2 Graphs and Directed Graphs	12
3.3 Polynomials	13

4	Matroids	15
4.1	Linear Matroids and Representable Matroids	16
4.2	Direct Sum of Matroids.	17
4.3	Truncation and elongation of a Matroid.	17
4.4	Examples of Matroids	18
4.4.1	Uniform and Partition Matroids	18
4.4.2	Graphic and Co-graphic Matroids	19
4.4.3	Transversal matroids	19
4.4.4	Gammoid and Strict Gammoid	20
5	Pseudo Random Objects	21
5.1	Definitions	21
5.2	FKS Hashing	22
5.3	Splitters	24
5.3.1	k -restriction problem	25
5.3.2	Solving k -restriction problem	25
5.3.3	(n, k, k) -family of perfect hash functions	27
5.3.4	(n, k) -universal sets	29
5.4	Lopsided universal sets and generalized universal sets	29
5.4.1	Efficient Construction of (n, k, q) -universal sets	30
6	Representative Family: Motivation, Definition and History	33
II	Representative Family in Set Systems	39
7	Computation of Representative Family in Set Systems	41
7.1	Computation using Lopsided Universal Sets	43
7.2	Computation using Separating Collections	45
8	Long Directed Cycle	73

8.1	Algorithm for Long Directed Cycle	74
8.2	Faster Algorithm for Long Directed Cycle	79
9	<i>k</i>-Path and <i>k</i>-Tree	85
9.1	<i>k</i> -PATH	87
9.2	<i>k</i> -TREE and <i>k</i> -SUBGRAPH ISOMORPHISM	88
10	<i>r</i>-Dimensional <i>k</i>-Matching	97
11	Representative Family computation for product family	109
12	Multilinear Monomial Detection	117
III	Representative Family in Linear Matroids	121
13	Computing Representative Family in Linear Matroids	123
14	Minimum Equivalent Graph	131
14.1	Algorithm for MEG	132
15	Editing to Connected <i>f</i>-Degree Graph	139
15.1	An overview of our algorithm	142
15.1.1	Characterizing the solution.	142
15.1.2	Towards the states of DP algorithm.	143
15.1.3	Pruning the DP table entry and an FPT algorithm.	144
15.2	Algorithm	145
15.2.1	Structural Characterization	146
15.2.2	An algorithm with running time $n^{\mathcal{O}(k)}$	149
15.2.3	Pruning the DP table – FPT algorithm	174
15.2.4	Derandomization	189

15.3 EDITING TO CONNECTED f -DEGREE GRAPH WITH COSTS	189
16 Representative Family Computation for a Product Family in a Linear Matroid	195
17 Dynamic Programming over graphs of bounded treewidth	201
17.1 STEINER TREE	203
17.2 FEEDBACK VERTEX SET	212
18 Matroidal Multilinear Monomial Detection	219
IV Matroid Girth and Connectivity	223
19 Matroid Girth	225
19.1 Algorithm for MATROID GIRTH	227
19.2 MATROID GIRTH on Specific Matroids	230
19.2.1 MATROID GIRTH on Transversal Matroid	230
19.2.2 MATROID GIRTH on Strict Gammoids	231
20 Matroid Connectivity	233
20.1 Hardness	234
20.2 Algorithm for MATROID CONNECTIVITY	236
V Steiner Tree	241
21 Polynomial Space Single Exponential FPT Algorithm	243
21.1 Notations and Separators	246
21.2 Algorithm	248
21.2.1 Obtaining better parameter dependence	257
21.3 $4^{(1+\epsilon)k}n^{\mathcal{O}(f(\epsilon))}$ time algorithm	258

21.3.1 SUBSET STEINER FOREST	258
21.3.2 Algorithm for STEINER TREE	259
VI Conclusion	265
22 Conclusion	267
Bibliography	269

Synopsis

In this thesis we study algorithms mainly in the parameterized complexity framework. The thesis has three conceptual parts – (i) efficient computation of representative families, and its application in FPT and exact algorithms, (ii) study of MATROID GIRTH and MATROID CONNECTIVITY problems under various natural parameters and (iii) single exponential polynomial space FPT algorithm for STEINER TREE parameterized by the number of terminals.

Let \mathcal{S} be a family of p sized sets over a universe U . A *subfamily* $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is called a *q -representative family* of \mathcal{S} , if $\widehat{\mathcal{S}}$ satisfies the following condition: if there is a q sized set $Y \subseteq U$ such that there is a set $S \in \mathcal{S}$ and $S \cap Y = \emptyset$, then there is a set $\widehat{S} \in \widehat{\mathcal{S}}$ and $\widehat{S} \cap Y = \emptyset$. The definition of representative family can be extended to matroids. We give efficient computation of representative family both for set systems and matroids. We demonstrate how the efficient construction of representative families can be a powerful tool for designing single-exponential parameterized and exact exponential time algorithms. We also use representative family together with other algorithmic techniques to get (*fast*) FPT algorithms. All the algorithms we designed using representative family are based on dynamic programming. Often, in dynamic programming, the family of partial solutions can have a representation with size sublinear in the size of the family of partial solutions. One such families are *product* families; a family \mathcal{F} is the *product* of \mathcal{A} and \mathcal{B} if $\mathcal{F} = \{A \cup B : A \in \mathcal{A}, B \in \mathcal{B}, A \cap B = \emptyset\}$. We give an algorithm for the computation of representative family for product families. Our algorithm, on input an integer q and families \mathcal{A}, \mathcal{B} of sets of sizes p_1 and p_2 over a universe of size n , computes a q -representative family \mathcal{F}' of \mathcal{F} . The running time of our algorithm is *sublinear* in $|\mathcal{F}|$ for many choices of \mathcal{A}, \mathcal{B} and q which occur naturally in several dynamic programming algorithms. Using representative families we obtain the following deterministic algorithms.

1. **LONG DIRECTED CYCLE.** In the LONG DIRECTED CYCLE problem we are interested in finding a cycle of length at least k in a directed graph. We give an algorithm of running time $\mathcal{O}(6.75^{k+o(k)}mn^2 \log n)$ for this problem.
2. **SHORT CHEAP TOUR.** In this problem we are given an undirected n -vertex graph G , $w : E(G) \rightarrow \mathbb{N}$ and an integer k . The objective is to find a path of length k with minimum weight. We give a $\mathcal{O}(2.619^k n^{\mathcal{O}(1)} \log W)$ time algorithm for SHORT CHEAP TOUR, where W is the largest edge weight in the

given input graph. We show that our algorithm can be generalized to the more general problem, k -TREE.

3. r -DIMENSIONAL MATCHING. Given a universe $U := U_1 \uplus \dots \uplus U_r$, and a r -uniform family $\mathcal{F} \subseteq U_1 \times \dots \times U_r$, the r -DIMENSIONAL MATCHING ((r, k) -DM) problem asks if \mathcal{F} admits a collection of k mutually disjoint sets. We give an algorithm for the problem running in time $2.619^{(r-1)k} |\mathcal{F}|^{\mathcal{O}(1)}$.
4. MULTILINEAR MONOMIAL DETECTION. Here the input is an arithmetic circuit C over \mathbb{Z}^+ representing a polynomial $P(X)$ over \mathbb{Z}^+ . The objective is to test whether $P(X)$ construed as a sum of monomials contain a multilinear monomial of degree k . For this problem we give an algorithm of running time $\mathcal{O}(3.8408^k 2^{\mathcal{O}(k)} s(C) n \log^2 n)$, where $s(C)$ is the size of the circuit.
5. MINIMUM EQUIVALENT GRAPH(MEG). In this problem we are seeking a spanning subdigraph D' of a given n -vertex digraph D with as few arcs as possible in which the reachability relation is the same as in the original digraph D . We give a single-exponential exact algorithm, i.e. of running time $2^{\mathcal{O}(n)}$, for the problem.
6. EDITING TO CONNECTED f -DEGREE GRAPH. In this problem we are given a graph G , an integer k and a function f assigning integers to vertices of G . The task is to decide whether there is a connected graph F on the same vertex set as G , such that for every vertex v , its degree in F is $f(v)$ and the number of edges in the symmetric difference of $E(G)$ and $E(F)$, is at most k . We show that EDITING TO CONNECTED f -DEGREE GRAPH parameterized by k is FPT by providing an algorithm solving the problem on an n -vertex graph in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.
7. **Dynamic Programming over graphs of bounded treewidth.** We give algorithms with running time $\mathcal{O}((1 + 2^{\omega-1} \cdot 3)^{\mathbf{tw}} \mathbf{tw}^{\mathcal{O}(1)} n)$ for FEEDBACK VERTEX SET and STEINER TREE, where \mathbf{tw} is the treewidth of the input graph, n is the number of vertices in the input graph and ω is the matrix multiplication exponent.

We study MATROID GIRTH and MATROID CONNECTIVITY problems on linear matroids representable over a field \mathbb{F}_q in the parameterized complexity framework. We consider the parameters – (i) solution size, k , (ii) $\mathbf{rank}(M)$, and (iii) $\mathbf{rank}(M) + q$, where M is the input matroid. We show that these problems are *unlikely* to be in FPT when parameterized by k or $\mathbf{rank}(M)$. We give fast FPT algorithm for these problems when parameterized by $\mathbf{rank}(M) + q$. We also study MATROID GIRTH on specific matroids like transversal matroids and gammoids.

Finally, we study STEINER TREE problem parameterized by the number of terminals. We give the first single-exponential time, polynomial-space FPT algorithm for the weighted STEINER TREE problem.

List of Publications from the Thesis

1. Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. *Submitted to a journal*.

Preliminary versions of this paper appeared in the proceedings of SODA 2014 and ESA 2014.

- Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. *In Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 142-151, 2014.
 - Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Representative sets of product families. *In Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 443-454, 2014.
2. Prachi Goyal, Neeldhara Misra, Fahad Panolan, and Meieav Zehavi. Deterministic algorithms for matching and packing problems based on representative sets. *SIAM Journal on Discrete Mathematics*, 29(4):1815-1836, 2015.

The conference version of the paper appeared in the proceedings of *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013*, volume 24, pages 237-248.
 3. Fedor V. Fomin, Petr Golovach, Fahad Panolan, and Saket Saurabh. Editing to connected f -degree graph. *To appear in STACS 2016*.
 4. Fahad Panolan, M. S. Ramanujan, Saket Saurabh. On the parameterized complexity of girth and connectivity problems on linear matroids. *In Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, pages 566-577, 2015.

5. Fedor V. Fomin, Petteri Kaski, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Parameterized single-exponential time polynomial space algorithm for steiner tree. *In Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 494-505, 2015.

List of Figures

8.1	Illustration to the proof of Lemma 8.1.	74
10.1	A schematic view of the proof of Lemma 10.2 when $r = 3$. It is meant to be read in clockwise order. Note that B is a representative for A with respect to $Y \cup \{S[1], S[2]\}$, and Z^* is a representative for $B \cup S$ with respect to Y . This eventually implies that Z^* is a representative for Z with respect to Y	106
11.1	Graph constructed from $\mathcal{L}_1, \mathcal{L}_2, \mathcal{F}$ and \mathcal{H}	111
15.1	Construction of T . The edges of cost 0 are shown by thin lines, the edges of cost 1 are shown by thick lines and the non-edges of cost 0 are shown by dashed lines. Notice that the graph G is encoded by assigning the cost 0 to every non-edge of T corresponding to an edge of G	190

List of Tables

9.1	Results for k -PATH	85
10.1	Algorithms for (r, k) -DM, with D denoting deterministic algorithms and R denoting randomized algorithms.	98
10.2	Algorithms for $(3, k)$ -DM, with D denoting deterministic algorithms and R denoting randomized algorithms.	99

Part I

Introduction

Chapter 1

Computational Framework

One of the main research in theoretical computer science is designing algorithms to solve problems “efficiently”. Formally a (decision) *problem* Π is defined as a subset of Σ^* , where Σ is a finite alphabet. An *algorithm* A for a problem Π is a finite and ordered set of instructions given to a machine (a computer), which takes as input $x \in \Sigma^*$ (also called an instance of Π) and decides whether $x \in \Pi$ or not. The efficiency of an algorithm is evaluated using various measures. Two such measures are how fast the algorithm runs and how much memory (space) the algorithm uses. In classical complexity theory the running time and space usage of an algorithm is measured in terms of the length of the input string x , denoted by $|x|$. To analyze the running time and space usage theoretically we need to define big- \mathcal{O} notation. Given two functions $f(n)$ and $g(n)$, we say $f(n) \in \mathcal{O}(g(n))$ (or $f(n) = \mathcal{O}(g(n))$), if there exists constants C and n_0 such that $f(n) \leq Cg(n)$ for all $n \geq n_0$. For more details about the asymptotic notations the reader is referred to monographs of algorithms like [75, 36]. If there is an algorithm for a problem Π , which runs in time $n^{\mathcal{O}(1)}$, where n is the length of the input (also called the input size), then we say Π is polynomial time solvable. If the space usage of an algorithm is bounded by $n^{\mathcal{O}(1)}$, where n is the length of the input, then we say that the algorithm is a polynomial space algorithm. In the classical complexity theory, the class P is defined as the set of problems that can be solved in polynomial time. For many problems, despite the intense effort, we were not able to come up with polynomial time algorithms.

For many problems, even though we were not able to get polynomial time algorithm, a polynomial time “verifier” exists— a polynomial time verifier for a problem Π is a polynomial time algorithm that takes an instance x of Π , and a string $y \in \Sigma^*$ (called a *certificate* for x) such that $|y| \in |x|^{\mathcal{O}(1)}$ as input and decides whether $x \in \Pi$ or not. The class of problems which has a polynomial time verifier is defined to be the class NP and clearly $P \subseteq NP$. But the question of “is $NP \subseteq P$?” remains as a notorious open problem in computer science. This leads to the study of intractability theory in algorithms. The intractability theory allows us to group together problems based on its hardness. One of the greatest discovery in computer science is the notion of

“reduction” and “completeness” for the class NP. The notion of reduction from a problem Π_1 to a problem Π_2 helps us to solve the problem Π_1 using an algorithm for Π_2 and thus to conclude that Π_2 is at least as hard as Π_1 . We briefly explain two kinds of reductions used in the literature.

- A *many-one reduction* (also called *Karp-reduction*) from a problem Π_1 to a problem Π_2 is an algorithm which takes an instance x of Π_1 as input and produces an instance y of Π_2 with the property that $x \in \Pi_1$ if and only if $y \in \Pi_2$. If the algorithm runs in polynomial time, then we say the reduction is a *polynomial time Karp-reduction*
- A *Turing-reduction* from a problem Π_1 to a problem Π_2 is an algorithm which solves Π_1 using a subroutine for Π_2 . If the number of steps the algorithm takes excluding the number of steps of the subroutine, but including the number of times the subroutine is called is polynomial in the input size, then we say the reduction is a *polynomial time Turing-reduction*

A problem Π is hard for the class NP (or NP-hard), if any problem in NP can be polynomial time many one reducible to Π and the problem Π is called NP-Complete if it is in NP as well. Unfortunately many interesting combinatorial problems are NP-Complete. One could observe that any problem in NP can be solved by enumerating all the certificates of the polynomial time verifier (brute force algorithm) for the problem. But can we do better than the brute force algorithm? This leads to study of algorithms in different frameworks rather than focusing only on algorithms which runs in polynomial time and always producing the correct solution. Some such frameworks are exact exponential time algorithms, approximation algorithms, fixed parameter tractability, randomized algorithms etc. Here we would like to mention that approximation algorithms are defined for *optimization problems*.

In this thesis we mainly study the algorithms in the realm of fixed parameter tractability. For brief overview of fixed parameter tractability see Section 1.1. We also design an exact exponential time algorithm for a problem and a brief overview of exact exponential time algorithms is given in Section 1.2. For ease of presentation for some problems we first design a randomized algorithm and then derandomize it. A brief overview of randomized algorithms is given in Section 1.3.

1.1 Fixed Parameter Tractability

In the classical computational complexity theory the running time of an algorithm is measured in terms of input size and classify problems based on whether they are polynomial time solvable or NP-hard. But in many natural problems, a small parameter compared to the input size may be associated with the input, like solution size of a problem, treewidth of a graph etc, and this can be exploited while designing an algorithm. Formally a parameterized problem Π is a subset of $\Sigma^* \times \mathbb{N}^+$, where

Σ is a finite alphabet and \mathbb{N}^+ is the set of positive natural numbers. The second component of the input is called the *parameter*. We say a parameterized problem Π is *fixed parameter tractable* (FPT) if there is an algorithm for the problem, runs in time $f(k)|x|^{\mathcal{O}(1)}$ on input (x, k) where f is an arbitrary function depending only on k . The study of algorithms where the running time of the algorithms measured in terms of multiple variables are referred as *Parameterized Complexity* in literature.

Like the P *verses* NP classification in the classical computational complexity theory, parameterized complexity is also enriched with an intractability theory. But for the analog of the complexity class NP, parameterized complexity contains a set of complexity classes which forms a hierarchy, called W-hierarchy. To define the classes in W-hierarchy, we need to define *boolean circuits*.

Definition 1.1. *A boolean circuit is a directed acyclic graph with one out-degree 0 node (called the output node) and each node is labeled in the following way.*

- *Every node of in-degree 0 is an input node*
- *Every node of in-degree at least 2 is either an AND-node or an OR-node*
- *Every node of in-degree 1 is a NOT-node (negation node).*

The nodes of in-degree strictly more than 2 is called large nodes. The depth of the circuit is the maximum length of a path from an input node to the output node and the weft of the circuit is the maximum number of large nodes on a path from an input node to the output node.

Assigning 0-1 values to the input nodes of a boolean circuit determines the value of every node in the natural way. If the value of the output node is 1 in an assignment to the input nodes, then we say that the assignment satisfies the circuit. The *weight* of an assignment is the number of ones assigned to input nodes. Note that given a circuit and an assignment to the input nodes in the circuit, we can check whether the assignment satisfies the circuit in polynomial time. Now we define a class of WEIGHTED CIRCUIT SATISFIABILITY problem and parameterized reduction which is crucial to define the W-hierarchy.

WEIGHTED CIRCUIT SATISFIABILITY $[t, d]$ (WCS $[t, d]$) **Parameter:** k
Input: A boolean circuit C of depth d and weft t and an integer k
Question: Is there a satisfying assignment of weight k for the circuit C

Definition 1.2. Let $\Pi_1, \Pi_2 \subseteq \Sigma^* \times \mathbb{N}^+$ be two parameterized problems. A parameterized reduction from Π_1 to Π_2 is an algorithm which takes an instance (x_1, k_1) of Π_1 and outputs an instance (x_2, k_2) of Π_2 such that

- $(x_1, k_1) \in \Pi_1$ if and only if $(x_2, k_2) \in \Pi_2$,
- $k_2 \leq g(k_1)$ for some computable function g , and
- the running time of the algorithm is bounded by $f(k_1)|x_1|^{\mathcal{O}(1)}$ for some computable function f .

Similar to the Turing reduction in the classical computational complexity, a *parameterized Turing reduction* from a parameterized problem Π_1 to a parameterized problem Π_2 is an FPT algorithm for Π_1 , which takes an instance (x, k) of Π_1 as input and decides $(x, k) \in \Pi_1$ or not, using a subroutine for Π_2 , where the parameter of each subroutine call is bounded by a computable function of k .

Proposition 1.1. Let Π_1 and Π_2 be two parameterized problems and there is a parameterized (Turing) reduction from Π_1 to Π_2 . If Π_2 is FPT, then Π_1 is also FPT.

Now we define the W-hierarchy.

Definition 1.3 (W-hierarchy). For $t \geq 1$, a parameterized problem Π belongs to the class $W[t]$, if there is parameterized reduction from Π to $WCS[t, d]$ for some constant $d \geq t$.

A parameterized problem Π is $W[t]$ -hard for any $t, d \geq 1$, if there is a parameterized reduction from $WCS[t, d]$ to Π and Π is $W[t]$ -complete if Π is in $W[t]$ as well. The classes $FPT \subseteq W[1] \subseteq W[2] \subseteq \dots$ forms a hierarchy. The following proposition is easy to see.

Proposition 1.2. Let Π_1 and Π_2 be two parameterized problems. If Π_1 is $W[t]$ -hard for some $t \geq 1$, there is parameterized (Turing) reduction from Π_1 to Π_2 and Π_2 is FPT, then $FPT = W[t]$

As like the P versus NP question, the questions “Is $FPT = W[1]$?” and “Is $W[t] = W[t + 1]$ for some $t \geq 1$?” are long standing open problems. It is believed that

$FPT \neq W[t]$ for any $t \geq 1$ and it is backed by standard complexity theory assumptions like *Exponential Time Hypothesis* [29].

Parameterized complexity also contains study of compression of problem instances called *kernelization*. But in this thesis we are only dealing with FPT algorithms and W-hardness. For more detailed reading about parameterized complexity we refer to monographs [48, 41, 37].

1.2 Exact Exponential Time Algorithms

One of the area of study in algorithms is Exact Exponential Time Algorithms. We have mentioned that each problem in NP can be solved by a brute force algorithm. Consider the HAMILTONIAN CYCLE problem, where the objective is to test whether the given n -vertex m -edge graph has a simple cycle containing all the vertices. This can be solved in time $\mathcal{O}(n!)$ time by enumerating all the permutation of the vertex set or in time $2^m n^{\mathcal{O}(1)}$ by enumerating all the edge subsets. But the algorithms of Bellman [9] and Held and Karp [69] from 1960s solves HAMILTONIAN CYCLE in time $\mathcal{O}(2^n n^2)$, which significantly outperforms the brute force algorithm. In exact algorithms we try to design algorithms which is better than the brute force. In the case of polynomial time algorithm we measure the quality of algorithm in terms of input size. How do we measure the quality of an exact exponential time algorithm? Consider any graph problem, where the input is a graph on n vertices and m edges. Here the input size is bounded bounded by $\mathcal{O}((m+n)\log(m+n))$. Most of the combinatorial problems on graphs has a simple brute force algorithm running in time $2^{\mathcal{O}(n+m)} n^{\mathcal{O}(1)}$. So, for any graph problem the quality of exact algorithms are measured in terms the number of vertices. If an algorithm for a graph problem runs in time $2^{\mathcal{O}(n)} n^{\mathcal{O}(1)}$, then we say it is an *Exact Exponential Time* algorithm. For more details about the exact exponential time algorithm we refer to the monograph [53].

1.3 Randomized Algorithms

In randomized algorithms, algorithm will have access to random bits and it uses these random bits to produce the output. There are mainly two kinds of randomized algorithms studied in the literature.

- **Monte Carlo algorithms.** In Monte Carlo algorithms, the algorithm runs in bounded time or space, but have a chance of producing incorrect output. The most basic randomized complexity class is RP, which is the class of decision problems for which there is a polynomial time randomized algorithm which recognizes NO-instances with probability 1 and recognizes YES-instances with a probability of at least $1/2$.

- **Las Vegas algorithms.** In Las Vegas algorithms, the algorithm always produces correct output and the expected running time is bounded. The class of decision problems having algorithms with polynomial time average case running time whose output is always correct are said to be ZPP.

In this thesis we design Monte Carlo algorithms for parameterized problems running in FPT time, with one sided error. That is the algorithm will recognize NO-instances with probability 1 and recognizes YES-instances with a probability of at least $1/2$. For more details about randomized algorithms we refer to the monograph [96].

Chapter 2

Organization of the thesis

In this chapter we explain how the rest of the thesis is organized. In Chapter 3 we give basic definitions and notations used in the thesis, which include notations from sets, functions, graphs and polynomials. In Chapter 4, we give basics about matroids, linear representation of matroids and lists some examples of matroids which we used in our algorithmic applications. In Chapter 5, we define and give known constructions about coloring families like family of perfect hash functions and universal sets, which are used extensively to derandomize algorithms. In this chapter we also define lopsided universal sets and generalized universal sets. Both the coloring families and the ideas used to construct these families are used in later chapters in the thesis like to construct representative family in set systems (Chapter 7) and derandomization of algorithm in Chapter 15. In Chapter 6, we define the main algorithmic tool used in the thesis, representative families, and give motivation to study it. We also talk about previous results about the construction representative families both in set systems and linear matroids.

In Part II we give efficient construction and many applications of representative families in set system. In Chapter 7 we give an efficient algorithm to compute representative families in set system and in Chapters 8,9 and 10 we show its applicability in the field of parameterized algorithms. The results in Chapters 7, 8 and 9 are from the paper [54] and its preliminary versions appeared in the proceedings of SODA 2014([57]) and ESA 2014 [55]. The Chapter 10 is based on the paper [66] and its conference version appeared in the proceedings of FSTTCS 2013 [65]. In Chapter 11, we give a faster algorithm to compute representative family for a product family. This computation can be used to design fast FPT algorithms. We show such an example in Chapter 12 by giving an algorithm for detecting a multilinear term on k variables, given an arithmetic circuit C representing n -variate polynomial over \mathbb{Z}^+ , running in time $\mathcal{O}(3.8408^k 2^{o(k)} s(C) n \log n)$, where $s(C)$ is the size of the circuit C . The results in chapters 11 and 12 are from the paper [55].

In Part III, we generalize representative families to matroids and see many applica-

tions using representative families in matroids which have linear representations. In Chapter 13 we give a fast algorithm to compute a representative family of a family of independent sets in a linear matroid. In Chapter 14, we give an exact exponential time algorithm for a problem MINIMUM EQUIVALENT GRAPH using representative families in linear matroids. In Chapter 15, we prove that EDITING TO CONNECTED f -DEGREE GRAPH is FPT (we refer to the chapter for definition of the problem) using an algorithm which uses both representative family in a linear matroid and color coding. The results in the Chapters 13 and 14 are based on the paper [54]. The results in the Chapter 15 are from the paper [49]. In Chapter 16 we give faster algorithm to compute representative family of a product family in a linear matroid and in Chapters 17 and 18 we show its applications. The results in Chapters 16, 17 and 18 are from the paper [55].

In Part IV we study problems on matroids like MATROID GIRTH and MATROID CONNECTIVITY in linear matroids for various natural parameters like solution size, rank of the input matroid and field size. For these parameters we either prove $W[1]$ -hardness or give fast FPT algorithms. We had a fast FPT algorithm for MATROID GIRTH in linear matroids when parameterized by the rank of the matroid and field size of the underlying linear representation of the matroid, using representative family. Later we designed a faster FPT algorithm for the problem using MacWilliams identity from coding theory. In Chapter 19 we included the algorithm using MacWilliams identity and studied the same problem on specific matroids like transversal matroids and gammoids. In Chapter 20 we study MATROID CONNECTIVITY in linear matroids for various natural parameters. The results in Part IV are from the paper [104].

In Part V we give the first polynomial space single exponential FPT algorithm for weighted STEINER TREE algorithm. Our algorithm is a combination of branching and dynamic programming using combinatorial facts about balanced separators in trees. This part is based on the paper [52].

In Chapter 22, we conclude the thesis by giving some open problems.

Chapter 3

Preliminaries

3.1 Sets, Families and Functions

Let U be a set. We use 2^U , $\binom{U}{i}$ and $\binom{U}{\leq i}$ to denote the family of all subsets of U , the family of all subsets of size i of U and the family of all subsets of size at most i of U respectively. A family \mathcal{F} of subsets of U is called a p -family if for all $X \in \mathcal{F}$, $|X| = p$. We use \mathbb{N} and \mathbb{N}^+ to denote the set of natural numbers and the set of positive natural numbers respectively. We use $[n]$ to denote the set $\{1, \dots, n\}$. For an integer n , we use \mathbb{Z}_n to denote the set $\{0, 1, \dots, n-1\}$. For a prime number p , \mathbb{Z}_p form a field with field addition and multiplication operations being addition and multiplication modulo p respectively.

Definition 3.1. *Given two families of sets \mathcal{L}_1 and \mathcal{L}_2 , we define*

$$\mathcal{L}_1 \circ \mathcal{L}_2 = \{X \cup Y \mid X \in \mathcal{L}_1, Y \in \mathcal{L}_2\}$$

$$\mathcal{L}_1 \bullet \mathcal{L}_2 = \{X \cup Y \mid X \in \mathcal{L}_1, Y \in \mathcal{L}_2, X \cap Y = \emptyset\}$$

For a function f from a domain D to a range R and $y \in R$, we use $f^{-1}(y)$ to denote the set $\{x \in D \mid f(x) = y\}$. If f is a function from a set X to a set Y and g is a function from the set Y to a set Z , then the composite function denoted by $g \circ f$, from X to Z is defined as $(g \circ f)(x) = g(f(x))$. We call a function $f : 2^U \rightarrow \mathbb{N}$, *additive* if for any subsets X and Y of U we have that $f(X) + f(Y) = f(X \cup Y) - f(X \cap Y)$. For a function $w : U \rightarrow \mathbb{N}$ and $S \subseteq U$, we use $w(S)$ to denote the number $\sum_{s \in S} w(s)$. We use ω to denote the matrix multiplication exponent. The current best known bound on $\omega < 2.373$ [120].

3.2 Graphs and Directed Graphs

We use “graph” to denote simple graphs without self-loops, directions, or labels, and “directed graph” or “digraph” for simple directed graphs without self-loops or labels. We use standard terminology from the books of Diestel [39], and Bang-Jensen and Gutin [7] for graph-related terms about undirected graphs and directed graphs respectively, which we do not explicitly define here. In general we use G to denote a graph and D to denote a digraph.

We use $V(G)$ and $E(G)$, respectively, to denote the vertex and edge sets of a graph G . We also use $G = (V, E)$ to denote a graph G with vertex set V and edge set E . We use $\overline{E(G)}$ to denote the set $(\binom{V(G)}{2}) \setminus E(G)$. For a vertex $v \in V(G)$, we use $E_G(v)$ to denote the set of edges of $E(G)$ incident to v , $\overline{E}_G(v)$ to denote the set of non-edges of $\overline{E(G)}$ incident to v , and $d_G(v)$ to denote $|E_G(v)|$, i.e the degree of vertex v . A graph G' is a *subgraph* of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. The subgraph G' is called an *induced subgraph* of G if $E(G') = \{uv \in E(G) \mid u, v \in V(G')\}$, in this case, G' is also called the subgraph *induced by* $V(G')$ and denoted by $G[V(G')]$. For a vertex set S , by $G - S$ we denote $G[V(G) \setminus S]$. By $N_G(u)$ we denote (open) neighborhood of u , that is, the set of all vertices adjacent to u . Similarly, by $N_G[u] = N_G(u) \cup \{u\}$ we define the closed neighborhood. For a subset $S \subseteq V(G)$, we define $N_G[S] = \cup_{v \in S} N_G[v]$ and $N_G(S) = N_G[S] \setminus S$. If the graph G is clear in the context then we may remove the subscript G from the notation. For an edge set $E' \subseteq E(G)$, we use (i) $V(E')$ to denote the set of *end vertices* of the edges in E' , (ii) $G - E'$ to denote the subgraph $G' = (V(G), E(G) \setminus E')$ of G , and (iii) $G[E']$ to denote the subgraph $(V(E'), E')$ of G . For a subset $B \subseteq \overline{E(G)}$, we use $G + B$ to denote the graph $G' = (V(G), E(G) \cup B)$. We say an edge $e \in E(G)$ is a *bridge* if $G - \{e\}$ has more connected components than G .

We use $V(D)$ and $A(D)$ respectively, to denote the vertex and arc sets of a digraph D . We use $U(D)$ denote the underlying undirected graph of D . A vertex u of D is an *in-neighbor* (*out-neighbor*) of a vertex v if $uv \in A(D)$ ($vu \in A(D)$, respectively). We denote the set of in-neighbors and out-neighbors of a vertex v by $N_D^-(v)$ and $N_D^+(v)$ correspondingly. The *in-degree* $d_D^-(v)$ (*out-degree* $d_D^+(v)$) of a vertex v is the number of its in-neighbors (out-neighbors). For $v \in V(D)$ we use $\text{In}_D(v)$ and $\text{Out}_D(v)$ to denote the sets of in-coming and out-going arcs incident with v respectively. If the digraph D is clear from the context, then we may remove the subscript D from the notation. Given a subset $V' \subseteq V(D)$ of a digraph D , let $D[V']$ denote the digraph induced by V' . A digraph D is *strong* if for every pair x, y of vertices there are directed paths from x to y and from y to x . A maximal strongly connected subdigraph of D is called a *strong component*. A *closed directed walk* in a digraph D is a sequence $v_0 v_1 \cdots v_\ell$ of vertices of D , not necessarily distinct, such that $v_0 = v_\ell$ and for every $0 \leq i \leq \ell - 1$, $v_i v_{i+1} \in A(D)$.

If P is a path from vertex u to vertex v in graph G (or in digraph D) then we say that (i) P *connects* u and v , (ii) u, v are, respectively, the *initial vertex* and the *final vertex* of P , and (iii) u, v are the *end vertices* of path P . Let $P_1 = x_1 x_2 \dots x_r$ and $P_2 = y_1 y_2 \dots y_s$ be two *edge-disjoint* paths in a graph G . If $x_r = y_1$ and

$V(P_1) \cap V(P_2) = \{x_r\}$, then we use P_1P_2 to denote the path $x_1x_2 \dots x_r y_2 \dots y_s$. For a path $P = u_1u_2 \dots u_\ell$ in a graph G , we use \overleftarrow{P} to denote the reverse path $u_\ell u_{\ell-1} \dots u_1$. A *path system* \mathcal{P} in graph G (resp., digraph D) is a collection of paths in G (resp. in D), and it is *edge-disjoint* if no two paths in the system share an edge. We use $V(\mathcal{P})$ and $E(\mathcal{P})$ ($A(\mathcal{P})$ for path system in digraph) for the set of vertices and edges, respectively, in a path system \mathcal{P} .

3.3 Polynomials

A polynomial over a field \mathbb{F} is an expression consisting of variables and coefficients from the field \mathbb{F} , that involves the operations of additions and multiplications. An example of a polynomial on two variables x and y over the field of irrational numbers is $P(x, y) = x^2 + xy - 3y$. A monomial $Z = x_1^{s_1} \dots x_n^{s_n}$ of a polynomial $P(x_1, \dots, x_n)$ is called *multilinear* if $s_i \in \{0, 1\}$ for all $i \in \{1, \dots, n\}$. We say a monomial $Z = x_1^{s_1} \dots x_n^{s_n}$ as *k-multilinear term*, if Z is multilinear and $\sum_{i=1}^n s_i = k$. In algorithms, polynomials are generally represented using *arithmetic circuits*.

Definition 3.2. *An arithmetic circuit C over a commutative ring R is a simple labelled directed acyclic graph with its internal nodes are labeled by $+$ or \times and leaves (in-degree zero nodes) are labeled from $X \cup R$, where $X = \{x_1, x_2, \dots, x_n\}$, a set of variables. There is a node of out-degree zero, called the root node or the output gate. The size of C , $s(C)$ is the number of vertices in the graph.*

Chapter 4

Matroids

In this chapter we give definitions related to matroids. For a broader overview on matroids we refer to [103].

Definition 4.1. A pair $M = (E, \mathcal{I})$, where E is a ground set and \mathcal{I} is a family of subsets (called independent sets) of E , is a matroid if it satisfies the following conditions:

(I1) $\emptyset \in \mathcal{I}$.

(I2) If $A' \subseteq A$ and $A \in \mathcal{I}$ then $A' \in \mathcal{I}$.

(I3) If $A, B \in \mathcal{I}$ and $|A| < |B|$, then there is $e \in (B \setminus A)$ such that $A \cup \{e\} \in \mathcal{I}$.

The axiom (I2) is also called the hereditary property and a pair (E, \mathcal{I}) satisfying only (I2) is called hereditary family. An inclusion wise maximal set of \mathcal{I} is called a *basis* of the matroid. Using axiom (I3) it is easy to show that all the bases of a matroid have the same size. A set $D \subseteq E$ is called a dependent set if D is not an independent set in M , i.e, $D \notin \mathcal{I}$. The cardinality of the smallest dependent set of a matroid M is called the *girth* of M , denoted by $g(M)$.

Rank and rank function of a matroid. Let $M = (E, \mathcal{I})$ be a matroid. The *rank* of a subset $S \subseteq E$ is the maximum cardinality of an independent set contained in S . The *rank function* r_M of M maps each subset of its ground set to its rank. That is $r_M : 2^E \rightarrow \mathbb{N}^+ \cup \{0\}$ such that $r_M(S) = \max\{|S'| : S' \subseteq S, S' \in \mathcal{I}\}$. From the definition of rank function r_M of M , it is clear that $r_M(E)$ is the cardinality of a basis of M . This number is called the *rank* of the matroid M , and is denoted

by $\text{rank}(M)$. That is, $\text{rank}(M) = r_M(E)$. Some times we use r to denote the rank function of a matroid M instead of r_M , if it is clear from the context. The closure $\text{cl}(A)$ of a subset A of E is the set $\text{cl}(A) = \{x \in E : r_M(A) = r_M(A \cup \{x\})\}$.

Contraction. For $X \subseteq E$, the contraction of M by X , written M/X , is the matroid on the underlying set $E \setminus X$ whose rank function $r_{M/X} : 2^{E \setminus X} \rightarrow \mathbb{N}^+ \cup \{0\}$ is defined as follows. For all $A \subseteq E \setminus X$, $r_{M/X}(A) = r_M(A \cup X) - r_M(X)$.

Dual of a matroid. The dual of a matroid $M = (E, \mathcal{I})$ is a matroid, denoted by M^* , which has the same ground set as M and a subset $A \subseteq E$ is independent in M^* if and only if there is a basis of M which is disjoint from A . In other words, a set B is a basis of M^* if and only if $E \setminus B$ is a basis of M . For a matroid $M = (E, \mathcal{I})$, it follows from the definition of dual of a matroid and rank of a matroid that $\text{rank}(M) + \text{rank}(M^*) = |E|$. The following proposition gives the relation between the rank function of a matroid and that of its dual.

Proposition 4.1 ([103]). *Let r_M and r_{M^*} be the rank functions associated with a matroid $M = (E, \mathcal{I})$ and its dual respectively. Then for any $S \subseteq E$, $r_{M^*}(S) = |S| - r_M(E) + r_M(E \setminus S)$.*

k -separation and connectivity of a matroid. Let $M = (E, \mathcal{I})$ be a matroid and r be the rank function associated with it. A k -separation of M is a partition (X, Y) of E such that $|X| \geq k$, $|Y| \geq k$ and $r(X) + r(Y) - r(E) \leq k - 1$. The connectivity of M , denoted by $\kappa(M)$, is the smallest k such that M has a k -separation.

4.1 Linear Matroids and Representable Matroids

Let A be a matrix over an arbitrary field \mathbb{F} and let E be the set of columns of A . For A , we define matroid $M = (E, \mathcal{I})$ as follows. A set $X \subseteq E$ is independent (that is $X \in \mathcal{I}$) if the corresponding columns are linearly independent over \mathbb{F} . The matroids that can be defined by such a construction are called *linear matroids*, and if a matroid can be defined by a matrix A over a field \mathbb{F} , then we say that the matroid is representable over \mathbb{F} . That is, a matroid $M = (E, \mathcal{I})$ of rank d is representable over a field \mathbb{F} if there exist vectors in \mathbb{F}^d corresponding to the elements such that linearly independent sets of vectors correspond to independent sets of the matroid. A matroid $M = (E, \mathcal{I})$ is called *representable* or *linear* if it is representable over some field \mathbb{F} . Following lemma shows that, we can get a linear representation of a matroid from its dual in polynomial time.

Lemma 4.1 ([103]). *If M is a representable matroid over a field \mathbb{F} , then the dual of M is also representable over \mathbb{F} . Moreover, given a linear representation of M over \mathbb{F} , we can compute a linear representation of M^* over the same field in polynomial time.*

4.2 Direct Sum of Matroids.

Let $M_1 = (E_1, \mathcal{I}_1)$, $M_2 = (E_2, \mathcal{I}_2)$, \dots , $M_t = (E_t, \mathcal{I}_t)$ be t matroids with $E_i \cap E_j = \emptyset$ for all $1 \leq i \neq j \leq t$. The direct sum $M_1 \oplus \dots \oplus M_t$ is a matroid $M = (E, \mathcal{I})$ with $E := \bigcup_{i=1}^t E_i$ and $X \subseteq E$ is independent if and only if $X \cap E_i \in \mathcal{I}_i$ for all $i \leq t$. Let A_i be the representation matrix of $M_i = (E_i, \mathcal{I}_i)$. Then,

$$A_M = \begin{pmatrix} A_1 & 0 & 0 & \dots & 0 \\ 0 & A_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & A_t \end{pmatrix}$$

is a representation matrix of $M_1 \oplus \dots \oplus M_t$. The correctness of this construction is proved in [93].

Proposition 4.2 ([93, Proposition 3.4]). *Given linear representations of matroids M_1, \dots, M_t over the same field \mathbb{F} , a representation of their direct sum can be found in polynomial time.*

4.3 Truncation and elongation of a Matroid.

Definition 4.2. *The t -truncation of a matroid $M = (E, \mathcal{I})$ is a matroid $M' = (E, \mathcal{I}')$ such that $S \subseteq E$ is independent in M' if and only if $|S| \leq t$ and S is independent in M (that is $S \in \mathcal{I}$). We use M_t to denote the t -truncation of a matroid M .*

Definition 4.3. *The ℓ -elongation of a matroid, where $\ell > \text{rank}(M)$ is defined as a matroid $M' = (E, \mathcal{I}')$ such that $S \subseteq E$ is a basis in M' if and only if, $r_M(S) = r_M(E)$ and $|S| = \ell$. We use $M(\ell)$ to denote the ℓ -elongation of a matroid M .*

Lemma 4.2 ([85]). *Given a $n \times m$ matrix A_M over a field \mathbb{F} , which is a linear representation of a matroid M , there is a deterministic algorithm running in $\mathcal{O}(nmt)$ field operations over \mathbb{F} and computes linear representations of t -truncation of M , M_t and t -elongation of M , $M(t)$ over a field $\mathbb{F}(X)$.*

4.4 Examples of Matroids

4.4.1 Uniform and Partition Matroids

A pair $M = (E, \mathcal{I})$ over an n -element ground set E , is called a uniform matroid if the family of independent sets is given by $\mathcal{I} = \{A \subseteq E \mid |A| \leq k\}$, where k is some constant. This matroid is also denoted as $U_{n,k}$. Every uniform matroid is linear and can be represented over a finite field by a $k \times n$ matrix A_M where the $A_M[i, j] = j^{i-1}$.

$$A_M = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & n \\ 1 & 2^2 & 3^2 & \cdots & n^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 2^{k-1} & 3^{k-1} & \cdots & n^{k-1} \end{pmatrix}$$

Matrix A_M is called Vandermonde matrix. Observe that for $U_{n,k}$ to be representable over a finite field \mathbb{F} , we need that the determinant of each $k \times k$ submatrix of A_M must not vanish over \mathbb{F} . Observe that any k columns corresponding to x_{i_1}, \dots, x_{i_k} itself form a Vandermonde matrix, whose determinant is given by

$$\prod_{1 \leq j < \ell \leq k} (x_{i_j} - x_{i_\ell}).$$

Combining this with the fact that x_1, \dots, x_n are n distinct elements of \mathbb{F} , we conclude that every subset of size at most k of the ground set is independent, while clearly each larger subset is dependent. Thus, choosing a field \mathbb{F} of size larger than n suffices. Note that this means that a representation of the uniform matroid $U_{n,k}$ can be stored using $\mathcal{O}(\log n)$ bits.

A partition matroid $M = (E, \mathcal{I})$ is defined by a ground set E being partitioned into (disjoint) sets E_1, \dots, E_ℓ and by ℓ non-negative integers k_1, \dots, k_ℓ . A set $X \subseteq E$ is independent if and only if $|X \cap E_i| \leq k_i$ for all $i \in \{1, \dots, \ell\}$. Observe that a partition matroid is a direct sum of uniform matroids $U_{|E_1|, k_1}, \dots, U_{|E_\ell|, k_\ell}$. Thus, by Proposition 4.2 and the fact that a uniform matroid $U_{n,k}$ is representable over a field

\mathbb{F} of size larger than n , we have that.

Proposition 4.3 ([93, Proposition 3.5]). *A representation over a field of size $\mathcal{O}(|E|)$ of a partition matroid can be constructed in polynomial time.*

4.4.2 Graphic and Co-graphic Matroids

Definition 4.4. *Given a graph G , a graphic matroid $M = (E, \mathcal{I})$ is defined by taking elements as edges of G (that is $E = E(G)$) and $F \subseteq E(G)$ is in \mathcal{I} if it forms a spanning forest in the graph G . The dual of graphic matroid is called co-graphic matroid. For a graph G , we use M_G and M_G^* to denote the graphic matroid and co-graphic matroid associated with the graph G respectively.*

The graphic matroid and co-graphic matroid are representable over any field of size at least 2. Consider the matrix A_M with a row for each vertex $i \in V(G)$ and a column for each edge $e = ij \in E(G)$. In the column corresponding to $e = ij$, all entries are 0, except for a 1 in i or j (arbitrarily) and a -1 in the other. This is a representation over reals. To obtain a representation over a field \mathbb{F} , one simply needs to take the representation given above over reals and simply replace all -1 by the additive inverse of 1.

Proposition 4.4 ([103]). *Graphic matroids and co-graphic matroids are representable over any field of size at least 2.*

Let G be a graph with ℓ connected components. Then from the definition of the graphic matroid M_G and the dual of a matroid, we have that any set $F \subseteq E(G)$ is independent in the co-graphic matroid M_G^* if and only if $G - F$ has exactly ℓ connected components. If G is a connected graph, then $F \subseteq E(G)$ is independent in M_G^* if and only if $G - F$ is connected.

4.4.3 Transversal matroids

Let G be a bipartite graph with the vertex set $V(G)$ being partitioned as A and B . The transversal matroid M of G has A as its ground set, and a subset $X \subseteq A$ is independent in M if and only if there is a matching that covers X . That is, X is independent if and only if there is an injective mapping $\phi : X \rightarrow B$ such that $\phi(v)$ is a neighbor of v for every $v \in X$.

4.4.4 Gammoid and Strict Gammoid

Let D be a directed graph. For $S, T \subseteq V(D)$, the set T is linked to S if there exist $|T|$ vertex-disjoint paths from S to T , where the end points of the paths are also disjoint. For a fixed $S, V' \subseteq V(D)$, a gammoid is a matroid with ground set V' , where $A \subseteq V'$ is independent if and only if A is linked to S . When $V' = V(D)$, the gammoid is called *strict gammoid*, and is denoted by the pair (D, S) .

Chapter 5

Pseudo Random Objects

In this chapter we discuss the (n, k, l) -family of perfect hash functions and (n, k) -universal sets which are family of functions satisfying some properties. Here we are interested in constructing an (n, k, k) -family of perfect hash functions and (n, k) -universal sets of size as small as possible, because we want to use these objects to derandomize algorithms whose running time will depend on the size of these objects. FKS Hashing can be used to create (n, k, k) -family of perfect hash functions of size $2^{\mathcal{O}(k)} \log^2 n$ [59, 113]. Since in parameterized complexity we are concerned about the the base of the exponential function, we describe another construction of (n, k, k) -family of perfect hash functions of size $e^k k^{\mathcal{O}(\log k)} \log^2 n$ via *splitters* [100]. In Section 5.1, we define these objects and splitters. Since construction of (n, k, k) -family of perfect hash functions using splitters uses the (n, k, k^2) -family of perfect hash functions constructed using FKS Hashing, we describe this construction in Section 5.2. In Section 5.3, we describe the construction of (n, k, k) -family of perfect hash functions and (n, k) -universal sets using *splitters*. In Section 5.4 we define *lopsided universal sets* and *generalized universal sets* and explain its efficient construction. The *lopsided universal sets* and *generalized universal sets* can be used to derandomize algorithms more efficiently in some situations.

5.1 Definitions

Definition 5.1. An (n, k, l) -splitter H is a family of functions from $[n]$ to $[l]$ such that for all $S \in \binom{[n]}{k}$, there is an $h \in H$ that splits S perfectly, i.e., into equal sized parts $h^{-1}(j) \cap S$, $j = 1, 2, \dots, l$ (or as equal as possible, if l does not divide k).

Definition 5.2. Let H be a family of functions from $[n]$ to $[l]$. The family H is

an (n, k, l) -family of perfect hash functions if for all $S \in \binom{[n]}{k}$, there is an $h \in H$ which is one-to-one on S . Notice that an (n, k, l) -family of perfect hash functions is a (n, k, l) -splitter, where $l \geq k$

Definition 5.3. A set of vectors $T \subseteq \{0, 1\}^n$ is called (n, k) -universal sets (or n - k -universal family), if for any index set $S \subseteq [n]$ with $|S| = k$, the projection of T on S contains all possible 2^k configurations. In other words, a family \mathcal{F} of sets over a universe U of size n , is an n - k -universal family if for every set $A \in \binom{U}{k}$ and every subset $A' \subseteq A$ there is some set $F \in \mathcal{F}$ whose intersection $F \cap A$ is exactly A' .

For the construction of pseudo random objects we need to define the notion of k -wise independent random variables from the probability theory.

Definition 5.4 (k -wise independent). Random variables $X_1 X_2 \dots X_n$ from a sample space $A_1 \times \dots \times A_n$ is said to be k -wise independent if for any k positions $i_1 < i_2 < \dots < i_k$ and $a_1 \in A_{i_1}, \dots, a_k \in A_{i_k}$, we have

$$\Pr[X_{i_1} = a_1 \wedge \dots \wedge X_{i_k} = a_k] = \prod_{j=1}^k \Pr[X_{i_j} = a_j]$$

Proposition 5.1 ([3]). There exists a k -wise independent probability space $H_{n,k,b} \subseteq [b]^n$, with each random variable taking values from $[b]$, of size $\mathcal{O}(n^k)$, and it can be constructed in time linear in the output size.

5.2 FKS Hashing

In this section we discuss about an explicit construction of an (n, k, k^2) -family of perfect hash functions developed in [59]. Consider the case where we want to construct a family of hash functions H from $[n]$ to $[l]$, where $l \geq k$, such that for any subset $S \in \binom{[n]}{k}$, there exists $h \in H$ such that h is one-to-one on S .

Lemma 5.1 ([59]). Fix a prime number p , such that $n < p < 2n$. Let $S \in \binom{[n]}{k}$ and $a \in \mathbb{Z}_p^*$ and $l \geq k$. Let $B(l, S, a, j) = |\{x | x \in S \text{ and } (ax \bmod p) \bmod l = j\}|$ for all $0 \leq j \leq l - 1$. In other words, $B(l, S, a, j)$ is the number of times the value j

is attained by the function $x \rightarrow (ax \bmod p) \bmod l$ when x is restricted to S . Then there exists $a \in \mathbb{Z}_p^*$ such that

$$\sum_{j=0}^{l-1} \binom{B(l, S, a, j)}{2} < \frac{k^2}{l}. \quad (5.1)$$

Hence we get the following corollary.

Corollary 5.1 ([59]). *For all $S \in \binom{[n]}{k}$, there exist $a \in \mathbb{Z}_p^*$ such that the mapping $x \rightarrow (ax \bmod p) \bmod k^2$ is one-to-one when restricted to S . In other words there exists an (n, k, k^2) -family of perfect hash functions of size $\mathcal{O}(n)$ and it can be constructed in time $\mathcal{O}(n)$.*

The following lemma can be proved using Lemma 5.1.

Lemma 5.2 ([59]). *For all $S \in \binom{[n]}{k}$, there exists $a \in \mathbb{Z}_p^*$ such that*

$$\sum_{j=0}^{k-1} B(k, S, a, j)^2 < 3k.$$

Using prime number theorem one can prove that an (n, k, l) -family of perfect hash functions of size $\mathcal{O}(k^2 \log n)$ can be constructed in time $(k \cdot \log n)^{\mathcal{O}(1)}$, where $l < k^2 \log n$.

Lemma 5.3 ([59]). *Let $S \in \binom{[n]}{k}$. Then there exists a prime $p < k^2 \log n$ such that the function $\lambda_p : x \rightarrow x \bmod p$ is one-to-one on S , i.e., for all $x, y \in S$ with $x \neq y$, $x \bmod p \neq y \bmod p$.*

Since primality testing can be done in polynomial time [1] and number of primes less than x is approximately equal to $\frac{x}{\log x}$, we get the following corollary.

Corollary 5.2. *An (n, k, l) -family of perfect hash functions of size $\mathcal{O}\left(\frac{k^2 \log n}{\log(k \log n)}\right)$ can be constructed in time $k^2 \log n \cdot (\log(k \log n))^{\mathcal{O}(1)}$, where $l < k^2 \log n$.*

Theorem 5.1. *An (n, k, k^2) -family of perfect hash functions of size $\mathcal{O}\left(\frac{k^4 \log^2 n}{\log(k \log n)}\right)$ can be constructed in time $k^{\mathcal{O}(1)} n \log^2 n$.*

Proof. Let H_1 be $(n, k, k^2 \log n)$ -family of perfect hash functions of size $\mathcal{O}(\frac{k^2 \log n}{\log(k \log n)})$ constructed as mentioned in Corollary 5.2. Let H_2 be $(k^2 \log n, k, k^2)$ -family of perfect hash functions of size $\mathcal{O}(k^2 \log n)$ constructed as mentioned in Corollary 5.1. Now consider the family of functions $H = \{g \circ f | f \in H_1, g \in H_2\}$. we claim that H is (n, k, k^2) -family of perfect hash functions because for any $S \in \binom{[n]}{k}$ there exist a function that maps elements of S to distinct values in $[k^2 \log n]$ and there exist a function in H_2 that maps these distinct values to distinct values in $[k^2]$. Since $|H_1| = \mathcal{O}(\frac{k^2 \log n}{\log(k \log n)})$ and $|H_2| = \mathcal{O}(k^2 \log n)$, the size of H is $\mathcal{O}(\frac{k^4 \log^2 n}{\log(k \log n)})$. The time to output H_1 and H_2 is bounded by $k^2 \log n \cdot (\log(k \log n))^{\mathcal{O}(1)}$. The time to output all the functions in H is equal to $\mathcal{O}(|H| \cdot n) = k^{\mathcal{O}(1)} n \log^2 n$ multiplication operations. \square

Alon et al. [4] give another efficient constructions of (n, k, k^2) -perfect families of hash functions:

Theorem 5.2 ([4]). *For any universe U of size n there is a (n, k, k^2) -perfect family f_1, \dots, f_t of hash functions from U to $[k^2]$ with $t = \mathcal{O}(k^{\mathcal{O}(1)} \cdot \log n)$. Such a family of hash functions can be constructed in time $\mathcal{O}(k^{\mathcal{O}(1)} n \log n)$.*

5.3 Splitters

In this section we describe about the construction of (n, k, k) -family of perfect hash functions and (n, k) -universal sets using (n, k, l) -splitters. In all three combinatorial objects- (n, k, k) -family of perfect hash functions, (n, k) -universal sets and (n, k, l) -splitters, our objective is to find a set of vectors of length n over an alphabet of size b (in case of (n, k, k) -family of perfect hash functions $b = k$, in case of (n, k) -universal sets $b = 2$ and in case of (n, k, l) -splitters $b = l$) such that for any k out of n indices, we will find some “nice” configurations. This generalized problem is called k -restriction problem. We will describe it formally in Section 5.3.1 and explain how these problems will fall into k -restriction problem. In Section 5.3.3 and Section 5.3.4 we construct (n, k, k) -family of perfect hash functions and (n, k) -universal sets respectively using the solution of k -restriction problem developed in Section 5.3.2 and using FKS hashing.

5.3.1 k -restriction problem

k -restriction problem is formally defined as follows

k-restriction problem

Input: Positive integers b, k, n and a list $\mathcal{C} = C_1, C_2, \dots, C_m$ where $C_i \subseteq [b]^k$ and with the collection \mathcal{C} being invariant under permutation of $[k]$
Output: Collection of vectors $\mathcal{V} \subseteq [b]^n$ such that $\forall S \subseteq [n]$ with $|S| = k$ and $\forall j : 1 \leq j \leq m, \exists v \in \mathcal{V}$ such that projection of v on $S, v(S) \in C_j$

An important parameter of k -restriction problem is $c = \min_{1 \leq j \leq m} |C_j|$. We call c/b^k the *density* of the problem. Now we explain how the combinatorial objects which we defined in Section 5.1, fall into the category of k -restriction problem.

1. **(n, k, l) -splitters.** To specify *splitters* as k -restriction problem, let $b = l$ and let \mathcal{C} consist of one set C_1 containing all vectors from $[b]^k$ such that each value in $[b]$ appears exactly k/l times (if l does not divide k , then some values in $[b]$ appear $\lceil k/l \rceil$ times and some appear $\lfloor k/l \rfloor$ times)
2. **(n, k, k) -family of perfect hash functions.** In this case, $b = k$ and \mathcal{C} consist of only one set C_1 containing all permutations of $[k]$
3. **(n, k) -universal sets.** In this case, $b = 2$ and \mathcal{C} consists of $C_x = \{x\}$ for all $x \in \{0, 1\}^k$

5.3.2 Solving k -restriction problem

We introduced the k -restriction problem to construct a *family of perfect hash functions* and *universal sets* of size as small as possible. So first we find the number of vectors that suffices to become a solution of k -restriction problem using probabilistic argument. If a vector $v \in [b]^n$ is chosen uniformly at random, then for any $S \in \binom{[n]}{k}$ and C_j the probability that $v(S) \in C_j$ is $\frac{|C_j|}{b^k} \geq \frac{c}{b^k}$, where $c = \min_{1 \leq i \leq m} |C_i|$. Therefore, if we choose t random vectors, $V_t = \{v_1, v_2, \dots, v_t\}$, we get via union bound that

$$\begin{aligned}
 \Pr[V_t \text{ is not a solution}] &\leq \sum_{S \in \binom{[n]}{k}} \sum_{j=1}^m \Pr[\forall v : v \in V_t \text{ and } v(S) \notin C_j] \\
 &\leq \binom{n}{k} \sum_{j=1}^m \left(1 - \frac{|C_j|}{b^k}\right)^t \\
 &\leq \binom{n}{k} m \left(1 - \frac{c}{b^k}\right)^t \tag{5.2}
 \end{aligned}$$

Restricting equation (5.2) to be less than 1 implies that

$$t \geq \left\lceil \frac{k \ln n + \ln m}{\ln(b^k/(b^k - c))} \right\rceil \quad (5.3)$$

Thus for *k-restriction problem*, there exist a solution of at most the size mentioned in the Equation 5.3. We will refer to (5.3) as the *union bound*. Let $H_{n,k,b}$ be a *k-wise independent probability space* with n random variables taking values in $[b]$, such as the one mentioned in Proposition 5.1. Note that the union bound (5.3) is applicable even when the vectors are not chosen uniformly at random from $[b]^n$, but chosen uniformly at random from a *k-wise independent space* $H_{n,k,b}$ because probability calculation only examines *k* sized sets of $[n]$.

Now we discuss about the construction of a solution of size equaling the union bound. This construction is computationally expensive, i.e, not in polynomial time or even in FPT time in parameter k . But we will use this construction for making a *family of perfect hash functions* and *universal sets* after reducing the size of the universe. Since we are discussing the general *k-restriction problem*, we assume that we have a *membership oracle*: a procedure that, given $v \in [b]^n$, $S \in \binom{[n]}{k}$ and $j \in [m]$, says whether or not $v(S) \in C_j$, within some time bound T . For the examples we are interested in, this oracle computation will be easy, usually taking just $\mathcal{O}(k)$ time.

Theorem 5.3 ([100]). *For k-restriction problem with $b \leq n$, there is a deterministic algorithm that outputs a collection obeying the k-restrictions, with the size of the collection equaling the union bound. The time taken to output the collection is*

$$\mathcal{O} \left(\frac{b^k}{c} \cdot \binom{n}{k} \cdot m \cdot T \cdot |H_{n,k,b}| \right) \quad (5.4)$$

where T is the time complexity of the membership oracle.

Proof. Consider a set-system in which the universe (ground set) is $H_{n,k,b}$. The sets are $T_{S,j}$, indexed by pairs (S, j) such that $S \in \binom{[n]}{k}$ and $1 \leq j \leq m$. $T_{S,j}$ consists of all $h \in H_{n,k,b}$ such that $h(S) \in C_j$. We do not explicitly list out the sets $T_{S,j}$: note that any given h can be tested for membership in $T_{S,j}$ in time T , using the given *membership oracle*. Any subset of $H_{n,k,b}$ that *hits* (intersects) all subsets $T_{S,j}$ is a good collection (i.e., is a collection satisfying the *k-restriction problem*). This is the well-known *hitting set* problem.

We can find such a collection by a greedy algorithm via a simple observation, which

follows fairly easily by inspecting (5.2) and by using the fact that (5.2) holds even if we pick vectors at random from $H_{n,k,b}$; the observation is that there must be an $h \in H_{n,k,b}$ such that h hits at least fraction c/b^k of the sets $T_{S,j}$. The obvious idea then is to find such an h using the *membership oracle* and add it to our current (partial) hitting set, removing the sets hit by h from the set system, and repeating this step. Finding such an h takes time at most $\mathcal{O}\left(\binom{n}{k} \cdot m \cdot T \cdot |H_{n,k,b}|\right)$; also, the number of sets in our set-system is effectively “shrunk” to at most $m \binom{n}{k} (1 - c/b^k)$ after picking h . Therefore the results of a greedy algorithm will produce a solution of size $\lceil \frac{k \ln n + \ln m}{\ln(b^k/(b^k - c))} \rceil$, same as that of (5.3). So, the total time taken is at most

$$\mathcal{O}\left(\binom{n}{k} \cdot m \cdot T \cdot |H_{n,k,b}| \left(\sum_{i=0}^{\infty} (1 - c/b^k)^i\right)\right) = \mathcal{O}\left(\frac{b^k}{c} \cdot \binom{n}{k} \cdot m \cdot T \cdot |H_{n,k,b}|\right) \quad (5.5)$$

□

For *family of perfect hash functions* and *universal sets*, we explicitly state the size and time complexity by substituting proper values (Note that $|H_{n,k,b}| \leq n^k$ as mentioned in Proposition 5.1) in Theorem 5.3 and get Theorem 5.4 as follows

Theorem 5.4 ([100]). (1) An (n, k, k) -family of perfect hash functions of cardinality $\mathcal{O}(e^k \sqrt{k} \log n)$ can be constructed deterministically in time $\mathcal{O}(k^{k+1} \binom{n}{k} n^k / k!)$. (2) An (n, k) -universal sets of cardinality $\mathcal{O}(k 2^k \log n)$ can be constructed deterministically in time $\mathcal{O}\left(\binom{n}{k} k 2^{2k} n^k\right)$.

5.3.3 (n, k, k) -family of perfect hash functions

First we give a brief overview of construction of (n, k, k) -family of perfect hash functions. Starting with the universe size n , we first reduce our problem to one with universe size k^2 by finding a polynomial time computable family A of (n, k, k^2) -family of perfect hash functions (Theorem 5.1). A construction of (k^2, k, k) -family of perfect hash functions will then be pulled back to (n, k, k) -family of perfect hash functions at a cost of $k^{\mathcal{O}(1)} \log^2 n$ in the size of the family. Towards the construction of (k^2, k, k) -family of perfect hash functions, we will first find (k^2, k, l) -splitters for $l = \mathcal{O}(\log k)$. This guarantees us, for each $S \in \binom{[k^2]}{k}$, there exists a function which partitions S equally in l blocks. Then for each block we construct $(k^2, k/l, k/l)$ -

family of perfect hash functions by applying [Theorem 5.4](#). We need splitters with universe size k^2 .

Lemma 5.4. *For any $k \leq n$ and for all $l \leq n$, there is an explicit family $B(n, k, l)$ of (n, k, l) -splitters of size $\binom{n}{l-1}$*

Proof. For every choice of $1 \leq i_1 < i_2 < \dots < i_{l-1} \leq n$, define a function $h : [n] \rightarrow [l]$ by $h(s) = j$ if and only if $i_{j-1} < s \leq i_j$, for all $s \in [n]$ (taking $i_0=0$ and $i_l=n$). \square

Construction. Let $l = c \log k$ for some constant c (we will fix c later). Let $A = A(n, k, k^2)$, $B = B(k^2, k, l)$ and $C = C(k^2, k/l, k/l)$ be respective function families presented by [Theorem 5.1](#), [Lemma 5.4](#) and (1) of [Theorem 5.4](#). Then our required family of perfect hash functions H can be defined as

$$H = \{(a, b, c_1, c_2, \dots, c_l) \mid a \in A, b \in B, \forall i \in [l] : c_i \in C\}$$

where each $(a, b, c_1, c_2, \dots, c_l) \in H$ is defined by

$$(a, b, c_1, c_2, \dots, c_l)(x) = c_{b(a(x))}(a(x)) + \frac{k}{l} (b(a(x)) - 1)$$

Correctness. It can be easily verified that each $h \in H$ maps $[n]$ to $[k]$. Let $S \in \binom{[n]}{k}$. We need to show that there exist a function in H which is *one-to-one* on S . By the property of A there exist a function $a \in A$ which is *one-to-one* on S . Let $S' = \{i \mid \exists j \in S : a(j) = i\}$. Since a is *one-to-one* on S , $|S'| = k$. By the property of B , there exist $b \in B$ such that b splits S' equally into l blocks. Let $S'_i = \{j \mid j \in S' \text{ and } b(j) = i\}$ for all $i \in [l]$. Now by the property of C , we have $c_i \in C$ for all i such that c_i is *one-to-one* on S'_i

Size and Time We know that, by [Theorem 5.1](#), $|A| = \mathcal{O}\left(\frac{k^4 \log^2 n}{\log(k \log n)}\right)$ and A can be constructed in time $k^{\mathcal{O}(1)} n \log^2 n$. By [Lemma 5.4](#), $|B| = \binom{k^2}{l-1} = k^{\mathcal{O}(\log k)}$ and B can be constructed in time $k^{\mathcal{O}(\log k)}$. By [Theorem 5.4](#), $|C| = \mathcal{O}(e^{k/l} \sqrt{k/l} \log k)$ and can be constructed in time $k^{\mathcal{O}(k/l)}$, which is equal to 2^k for a suitable choice of c . Hence the size of H is,

$$\begin{aligned} |H| &= |A| \cdot |B| \cdot |C|^l \\ &= \mathcal{O}\left(\frac{k^4 \log^2 n}{\log(k \log n)}\right) \cdot k^{\mathcal{O}(\log k)} \cdot (e^k k^{\mathcal{O}(\log k)}) \\ &= e^k k^{\mathcal{O}(\log k)} \log^2 n \end{aligned}$$

The running time to output H is $\mathcal{O}(|H| \cdot n)$. Thus we have the following theorem.

Theorem 5.5 ([100]). *An (n, k, k) -family of perfect hash functions of size $e^k k^{\mathcal{O}(\log k)} \log^2 n$ can be constructed in time linear in the output size.*

5.3.4 (n, k) -universal sets

The idea for (n, k) -universal sets is similar to that behind [Theorem 5.5](#), with the only modification being that we now need the universal sets guaranteed by (2) of [Theorem 5.4](#). Thus we get

Theorem 5.6 ([100]). *An (n, k) -universal sets of size $2^k k^{\mathcal{O}(\log k)} \log^2 n$ can be constructed in time linear in the output size*

Remark 5.1. *In the construction of (n, k, k) -family of perfect hash functions and (n, k) -universal sets, if we use [Theorem 5.2](#) instead [Theorem 5.1](#), then we can reduce the family size and running time in [Theorem 5.5](#) and [Theorem 5.6](#) by a factor of $\log n$.*

5.4 Lopsided universal sets and generalized universal sets

We tweak the notion of universal families as follows and it can be computed efficiently by slightly changing the construction of Naor et al. [100].

Definition 5.5. *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 $B \cap F = \emptyset$.*

It follows from the definition of n - p - q -lopsided-universal that a n - $(p+q)$ -universal family is also n - p - q -lopsided-universal. By slightly changing the construction of Naor et al. [100], one can prove the following result.

Lemma 5.5. *There is an algorithm that given n , p and q constructs an n - p - q -lopsided-universal family \mathcal{F} of size $\binom{p+q}{p} \cdot 2^{\mathcal{O}(p+q)} \cdot \log n$ in time $\mathcal{O}(\binom{p+q}{p} \cdot 2^{\mathcal{O}(p+q)} \cdot n \log n)$.*

Lemma 5.5 is a direct corollary of Lemma 7.4 proved in Chapter 7.

We generalize *universal sets* and show that the construction by Naor et al. [100] can be generalized to create generalized universal sets.

Definition 5.6. An (n, k, q) -universal set is a set of vectors $V \subseteq [q]^n$ such that for any index set $S \subseteq \binom{[n]}{k}$, the projection of V on S contains all possible q^k configurations.

An (n, k, q) -universal set is a special case of *k-restriction problem*. To specify (n, k, q) -universal set as *k-restriction problem*, let $b = q$ and \mathcal{C} consist of $C_x = \{x\}$ for all $x \in [q]^k$. By substituting values for (n, k, q) -universal sets in Theorem 5.3 we get the following corollary.

Corollary 5.3. An (n, k, q) -universal set of cardinality $\mathcal{O}(kq^k \log n)$ can be constructed deterministically in time $\mathcal{O}(q^{2k} \binom{n}{k} n^k k)$.

5.4.1 Efficient Construction of (n, k, q) -universal sets

In this subsection we generalize the construction of universal sets (where $q=2$) by M. Naor et al. [100].

Construction. Let $l = c \log k$ for some constant c (we will fix c later). Let $A = A(n, k, k^2)$, $B = B(k^2, k, l)$ and $C = C(k^2, k/l, q)$ be respective function families presented by Theorem 5.1, Lemma 5.4 and Corollary 5.3. Then our required (n, k, q) -universal sets is a family of functions H ,

$$H = \{(a, b, c_1, c_2, \dots, c_l) \mid a \in A, b \in B, \forall i \in [l] : c_i \in C\}$$

where each $(a, b, c_1, c_2, \dots, c_l) \in H$ is defined by

$$(a, b, c_1, c_2, \dots, c_l)(x) = c_{b(a(x))}(a(x))$$

Correctness. It can be easily verified that each $h \in H$ maps $[n]$ to $[q]$. Let $S \in \binom{[n]}{k}$. We need to show that the restriction of functions in H on S gives all possible functions from $S \rightarrow [q]$. By the property of A there exist a function $a \in A$ which is *one-to-one* on S . Let $S' = \{i \mid \exists j \in S : a(j) = i\}$. Since a is *one-to-one* on S , $|S'| = k$. By the property of B , there exist $b \in B$ such that b splits S' equally into l blocks. Let $S'_i = \{j \mid j \in S' \text{ and } b(j) = i\}$ for all $i \in [l]$. Now by the property of C , we have restriction of functions from C on S'_i is all possible functions from $S'_i \rightarrow [q]$.

Size and Time We know that $|A| = \mathcal{O}\left(\frac{k^4 \log^2 n}{\log(k \log n)}\right)$ and A can be constructed in

time $k^{\mathcal{O}(1)}n \log n$. By Lemma 5.4, $|B| = \binom{k^2}{l-1} = k^{\mathcal{O}(\log k)}$ and B can be constructed in time $k^{\mathcal{O}(\log k)}$. By Corollary 5.3, $|C| = \mathcal{O}(q^{k/l}k \log k)$ and can be constructed in time $q^{\mathcal{O}(k/l)}k^{\mathcal{O}(k/l)}$, which is bounded by q^k for a suitable choice of c . Hence size of H is,

$$\begin{aligned}
|H| &= |A| \cdot |B| \cdot |C|^l \\
&= \mathcal{O}\left(\frac{k^4 \log^2 n}{\log(k \log n)}\right) \cdot k^{\mathcal{O}(\log k)} \cdot (q^k k^{\mathcal{O}(\log k)}) \\
&= q^k k^{\mathcal{O}(\log k)} \log^2 n
\end{aligned}$$

We can output the family H in time $\mathcal{O}(|H| \cdot n)$.

Chapter 6

Representative Family:

Motivation, Definition and History

Representative families are used in literature to design FPT algorithms. In this thesis we mainly study efficient construction these families and show that indeed it is a powerful algorithmic tool in the field of parameterized and exact algorithms by designing efficient algorithms for many important problems.

In fact the definition of representative family, naturally comes when we try to design algorithms for problems where ‘disjointness’ is a requirement for solutions. To explain this, we first consider k -PATH problem. In this problem we are given an n -vertex graph G and a positive integer k , and the objective is to check whether there is a simple path on k vertices (path of length $k - 1$), called k -path, in G . A simple dynamic programming algorithm for k -PATH computes a table \mathcal{A} , where each table entry in \mathcal{A} is indexed by a vertex $v \in V(G)$ and $i \in [k]$, and it stores the following information. For any $v \in V(G)$ and $i \in [k]$, $\mathcal{A}[v, i]$ contains all sets $X \subseteq V(G)$ such that there is path of length $i - 1$ ending at v using all the vertices of X . The algorithm can compute the DP table entries $\mathcal{A}[v, i]$ in the increasing order of i using the following recurrence relation.

$$\mathcal{A}[v, i] = \begin{cases} \{\{v\}\} & \text{if } i = 1 \\ \bigcup_{u \in N_G(v)} (\mathcal{A}[u, i - 1] \bullet \{v\}) & \text{if } i \in [k] \setminus \{1\} \end{cases}$$

Clearly the graph G has a k -path, if and only if there is a vertex $v \in V(G)$ such that $\mathcal{A}[v, k] \neq \emptyset$. Notice that the cardinality of $\mathcal{A}[v, i]$ for any $v \in V(G)$ and $i \in [k]$, is potentially be $\binom{n}{i}$, and the algorithm runs in time $n^{\mathcal{O}(k)}$.

Note that the table entries $\mathcal{A}[v, i]$ in our dynamic programming stores potential partial solutions such that one of them leads to a final solution. Perhaps we do

not need to store all the all partial solutions, instead store some partial solutions such that one of them leads to a final solution. To maintain the correctness of the algorithm it is enough keep a subfamily $\widehat{\mathcal{A}}[v, i]$ of $\mathcal{A}[v, i]$ with following property.

Property 1: For every $Y \in \binom{V(G)}{k-i}$, if there is a set $X \in \mathcal{A}[v, i]$ such that X is disjoint from Y , then there is a set $\widehat{X} \in \widehat{\mathcal{A}}[v, i]$ which is disjoint from Y .

Notice that Property 1 ensures that if there is a set $X \in \mathcal{A}[v, i]$ and $Y \in \binom{V(G)}{k-i}$ such that there is a simple path of length $k - 1$ using the vertices of $X \cup Y$, then there is a set $\widehat{X} \in \widehat{\mathcal{A}}[v, i]$ such that there is a simple path of length $k - 1$ using the vertices of $\widehat{X} \cup Y$.

Alon et al [4] introduced the famous algorithmic technique *color coding* which does the pruning of $\mathcal{A}[v, i]$ as follows. We first uniformly at random color the vertices of G using k colors. Then for any fixed k -path P , all the vertices in P will be colored with different colors with probability $\geq e^{-k}$ and the above algorithm is modified to output such a k -path (*colorful k -path*). Now in $\widehat{\mathcal{A}}[v, i]$, for each subset C of i colors it is enough to keep one partial solution X such that the colors of X is exactly same as C . Note that, since the number of colors is k , the number of partial solutions stored in $\widehat{\mathcal{A}}[v, i]$ is at most $\binom{k}{i}$. This modification to the DP table entries gives an algorithm of running time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$. This algorithm may have one-sided error, outputs a correct solution with probability at least e^{-k} and its success probability can be increased to a constant by running the above algorithm e^k times. This leads to a randomized algorithm for k -PATH running in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ with constant success probability. This algorithm can be derandomized using (n, k, k) -family of perfect hash functions \mathcal{F} , where the random coloring is replaced with colorings specified by the functions in the \mathcal{F} . In fact one can think this step as another way of pruning the DP table entries $\mathcal{A}[v, i]$ which leads to a deterministic algorithm for k -PATH. That is for each coloring specified by a function $f \in \mathcal{F}$, a set C of i colors we keep one set $X \in \widehat{\mathcal{A}}[v, i]$ such that $f(X) = C$.

The concept of representative family captures Property 1, which is implicitly used by the color coding technique. That is, if $\widehat{\mathcal{A}}[v, i]$ satisfies Property 1, then we say that $\widehat{\mathcal{A}}[v, i]$ is a $(k - i)$ -representative family of $\mathcal{A}[v, i]$.

Definition 6.1 ([97, 92]). *Let \mathcal{S} is a p -family over a universe U . A subfamily $\widehat{\mathcal{S}}$ of \mathcal{S} is called a q -representative family of \mathcal{S} if it satisfies the following condition. For every $Y \in \binom{U}{q}$, if there is a set $X \in \mathcal{S}$ such that $X \cap Y = \emptyset$, then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ such that $\widehat{X} \cap Y = \emptyset$.*

The basic questions regarding representative families are : “what is the smallest cardinality of a representative family of a family of sets and how fast we can compute these representative family?” The derandomization of the above mentioned algorithm of Alon et al. [4] implies that there is a q -representative family of a p -family \mathcal{S} over a universe U , of cardinality $\binom{p+q}{p}e^{p+q+o(p+q)}\log^2 n$ and it can be computed

in time $|\mathcal{S}| \binom{p+q}{p} e^{p+q+o(p+q)} \log^2 n$, where $n = |U|$. In fact the classic Two-Families Theorem of Bollobás [24] for extremal set systems imply that every p -family has a q -representative family with at most $\binom{p+q}{p}$ sets. In fact we can show that the size of a q -representative family of a p -family is lower bounded by $\binom{p+q}{p}$:

Lemma 6.1. *Let U be a universe of size $p + q$ and $\mathcal{S} = \binom{U}{p}$. Then the only q -representative family of \mathcal{S} is \mathcal{S} itself.*

Proof. Suppose there is a q -representative family $\widehat{\mathcal{S}}$ of \mathcal{S} such that $\widehat{\mathcal{S}} \neq \mathcal{S}$. This implies that there is a set $S \in \mathcal{S} \setminus \widehat{\mathcal{S}}$. Let $Y = U \setminus S$. Note that $|Y| = q$ and the only set in \mathcal{S} which is disjoint from Y is S . This contradicts the fact that $\widehat{\mathcal{S}}$ is a q -representative family of \mathcal{S} . \square

Monien provided an algorithm computing a q -representative family of size at most $\sum_{i=0}^q p^i$ in time $\mathcal{O}(pq \cdot \sum_{i=0}^q p^i \cdot |\mathcal{S}|)$ [97]. Marx in [92] provided another algorithm, for finding q -representative family of size at most $\binom{p+q}{p}$ in time $\mathcal{O}(p^q \cdot |\mathcal{S}|^2)$. We give a faster algorithm to compute representative family.

Theorem 6.1. *Let \mathcal{S} be a p -family of sets over a universe of size n and let $0 < x < 1$. For a given q , a q -representative family $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ for \mathcal{S} with at most $x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)}$ sets can be computed in time $\mathcal{O}((1-x)^{-q} \cdot 2^{o(p+q)} \cdot |\mathcal{S}| \cdot \log n)$.*

When $x = \frac{p}{p+q}$, the size of the q -representative family in Theorem 6.1 is upper bounded by $\binom{p+q}{p} 2^{o(p+q)}$. See chapter 7 for more details.

Marx [93], generalized the concept of representative families to matroids and used to design FPT algorithms for problems like finding a k -element set in the intersection of ℓ linear matroids.

Definition 6.2 ([93]). *Let $M = (E, \mathcal{I})$ be a matroid and \mathcal{S} be a family of subsets of E . A subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is called a q -representative for \mathcal{S} if the following holds: for every set $Y \subseteq E$ of size at most q , if there is a set $X \in \mathcal{S}$ disjoint from Y with $X \cup Y \in \mathcal{I}$, then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y with $\widehat{X} \cup Y \in \mathcal{I}$*

The generalization Two-Families Theorem of Bollobás [24] to subspaces of a vector space of Lovász [87] (see also [58]) imply that every family of sets of size p has a q -representative family with at most $\binom{p+q}{p}$ sets. Marx [93] has shown how Lovász's proof can be transformed into an algorithm computing a q -representative family. However, the running time of the algorithm given in [93] is $f(p, q)(|A_M|t)^{\mathcal{O}(1)}$, where

$f(p, q)$ is a polynomial in $(p+q)^p$ and $\binom{p+q}{p}$, that is, $f(p, q) = 2^{\mathcal{O}(p \log(p+q))} \cdot \binom{p+q}{p}^{\mathcal{O}(1)}$, and A_M is the matroid's representation matrix. Thus, when p is a constant, we have that $f(p, q) = (p+q)^{\mathcal{O}(1)}$. However, for unbounded p (for an example when $p = q = \frac{k}{2}$) the running time of this algorithm is bounded by $2^{\mathcal{O}(k \log k)} (||A_M||t)^{\mathcal{O}(1)}$. We give faster algorithm to compute representative family in linear matroids and its proof is based on the exterior algebra based proof of Lovász [87] and exploits the multi-linearity of the determinant function.

Theorem 6.2. *Let $M = (E, \mathcal{I})$ be a linear matroid of rank $p + q = k$, $\mathcal{S} = \{S_1, \dots, S_t\}$ be a p -family of independent sets. Given a representation A_M of M over a field \mathbb{F} , we can find a q -representative family $\widehat{\mathcal{S}}$ for \mathcal{S} , of size at most $\binom{p+q}{p}$ in $\mathcal{O}\left(\binom{p+q}{p} t p^\omega + t \binom{p+q}{q} \omega^{-1}\right)$ operations over \mathbb{F} .*

We show that the concept of representative family in linear matroids is a powerful algorithmic tool by designing FPT and exact algorithms for many basic problems in graph theory (See Part III for more details). The algorithmic tool representative family can be used as a better alternative for the linear algebra based approach of Bodlaender et al. [23] in many cases like solving “connectivity” problems such as STEINER TREE in bounded treewidth graphs. In the STEINER TREE problem the input is a graph G and a set of terminals $T \subseteq V(G)$, and the objective is to output a connected subgraph containing T with minimum number of edges. The approach of Bodlaender et al. [23] for solving STEINER TREE in bounded treewidth graphs is to do a dynamic programming over the tree decomposition of the input graph G . At each step the algorithm prune the set of partial solutions \mathcal{S} computed so far by computing linearly independent rows in a $|\mathcal{S}| \times 2^{\text{tw}}$ matrix. These linearly independent rows corresponds a subset of \mathcal{S} and which will form a small set of representative partial solutions. In our approach we relate each partial solution to an independent set in a graphic matroid of an auxiliary graph. By replacing the partial solution pruning step of the algorithm by Bodlaender et al. [23] with that of representative families in a graphic matroid (Theorem 6.2), we get another algorithm with same running time. Here the bottle neck of the running time is the pruning of the partial solution in the join node of the tree decomposition. This motivate us to study about the computation of representative family for families that arise naturally in the dynamic programming. One such family, like the one arise in the join node of tree decomposition, is *product family*. A family \mathcal{F} is the product of two families of independent sets \mathcal{A} and \mathcal{B} of a matroid $M = (E, \mathcal{I})$, if $\mathcal{A} = \{A \cup B \in \mathcal{I} : A \in \mathcal{A}, B \in \mathcal{B}, A \cap B = \emptyset\}$. For many computational problems on graphs of bounded treewidth in the join nodes of the decomposition, the family of partial solutions is the product of the families of its children, and we wish to store a representative set (for a graphic matroid) for this product family. In chapter 16 we design a faster algorithm to compute representative family for a product family in a linear matroid. By making use of this algorithm one can obtain faster deterministic algorithms for many connectivity problems. We exemplify this by providing algorithms with running time $\mathcal{O}(1 + 2^{\omega-1} \cdot 3)^{\text{tw}} \text{tw}^{\mathcal{O}(1)n}$ for and STEINER

TREE and FEEDBACK VERTEX SET on n -vertex graph with treewidth at most \mathbf{tw} , where ω is the matrix multiplication constant (See chapter 17 for algorithms and definition of FEEDBACK VERTEX SET). In fact we don't know how to get the same running time improvement for algorithms of Bodlaender et al. [23] for STEINER TREE and FEEDBACK VERTEX SET. We would like to remark that the algorithm of Bodlaender et al. [23] for HAMILTONIAN CYCLE is better than that of the one using representative family.

In fact all our computations of representative families are for weighted versions, which allows sets to have weights, with an additional $\log W$ overhead in the computation where W is the maximum weight assigned to any set. This allows us to solve weighted version of the problems in many representative set based algorithms. We also show that the algorithmic tool representative family can be combined with other algorithmic technique like color coding to design FPT algorithms (see chapter 15 for example).

Part II

Representative Family in Set Systems

Chapter 7

Computation of Representative Family in Set Systems

Let \mathcal{S} be a p -family of subsets of a universe U . Recall the definition of a representative family. A subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is q -representative for \mathcal{S} (denoted by $\widehat{\mathcal{S}} \subseteq_{rep}^q \mathcal{S}$), if for every set $Y \subseteq U$ of size at most q , if there is a set $X \in \mathcal{S}$ disjoint from Y , then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y . We give a faster algorithm for computing representative families and in subsequent chapters we show how they can be used to obtain improved parameterized algorithms for several fundamental and well studied problems. Essentially we prove the following theorem.

Theorem 7.1. *Let \mathcal{S} be a p -family of sets over a universe of size n and let $0 < x < 1$. For a given q , a q -representative family $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ for \mathcal{S} with at most $x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)}$ sets can be computed in time $\mathcal{O}((1-x)^{-q} \cdot 2^{o(p+q)} \cdot |\mathcal{S}| \cdot \log n)$.*

In fact, we prove a variant of Theorem 7.1, which allows sets to have weights, with an additive factor $|\mathcal{S}| \cdot \log |\mathcal{S}| \cdot \log W$ in the running time. This extension will be used in several applications. This theorem uses the notion of weighted representative families and computes a weighted q -representative family of size claimed in Theorem 7.1.

Definition 7.1 (Min/Max q -Representative Family). *Given a family \mathcal{S} of subsets of a universe U and a non-negative weight function $w : \mathcal{S} \rightarrow \mathbb{N}$, we say that a subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is min q -representative (max q -representative) for \mathcal{S} if the following holds: for every set $Y \subseteq U$ of size at most q , if there is a set $X \in \mathcal{S}$ disjoint from*

Y , then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y with $w(\widehat{X}) \leq w(X)$ ($w(\widehat{X}) \geq w(X)$). We use $\widehat{\mathcal{S}} \subseteq_{\min rep}^q \mathcal{S}$ ($\widehat{\mathcal{S}} \subseteq_{\max rep}^q \mathcal{S}$) to denote a min q -representative (max q -representative) family for \mathcal{S} .

When $x = \frac{p}{p+q}$, the size of the q -representative family in Theorem 7.1 is bounded by $\binom{p+q}{p} 2^{o(p+q)}$. Based on the values for x , we can get an interesting trade-off between the size of the computed representative family and the time taken to compute the representative family. This trade-off can be exploited algorithmically to speed up “representative families based” algorithms (see Chapters 8, 9 and 10).

The proof of Theorem 7.1 is essentially an algorithmic variant of the “random permutation” proof of Bollobás Lemma (see [72, Theorem 8.7]). A slightly weaker variant of Bollobás Lemma can be proved using random partitions instead of random permutations, the advantage of the random partitions proof being that it can be de-randomized using efficient constructions of *universal sets* [100]. In Section 7.1 we give an algorithm for computing q -representative family of size approximately $\binom{p+q}{p}$ using *lopsided universal sets*.

To obtain our result we define *separating collections* and give efficient constructions of them (see Section 7.2). Separating collections can be seen as a variant of universal sets. In its simplest form, an n - p - q -*separating collection* \mathcal{C} is a pair (\mathcal{F}, χ) , where \mathcal{F} is a family of sets over a universe U of size n and χ is a function from $\binom{U}{p}$ to $2^{\mathcal{F}}$ such that the following two properties are satisfied; (a) for every $A \in \binom{U}{p}$ and every $F \in \chi(A)$, $A \subseteq F$, (b) for every $A \in \binom{U}{p}$ and $B \in \binom{U \setminus A}{q}$, there is an $F \in \chi(A)$ such that $A \subseteq F$ and $F \cap B = \emptyset$. The *size* of (\mathcal{F}, χ) is $|\mathcal{F}|$, whereas the *max degree* of (\mathcal{F}, χ) is $\max_{A \in \binom{U}{p}} |\chi(A)|$. An efficient construction of separating collections is an algorithm that given n , p and q outputs the family \mathcal{F} of a separating collection (\mathcal{F}, χ) and then allows queries $\chi(A)$ for $A \in \binom{U}{p}$. In Section 7.2, we first give constructions of separating collections with construction and query time bounded by linear in the size of the output. The size of the separating collections produced is optimal up to subexponential factors in $p + q$ (for $x = \frac{p}{p+q}$). Then using separating collections we prove weighted version of the Theorem 7.1.

Theorem 7.1 is a generalization of a result in [57]. In the paper [57] Theorem 7.1 is proved for $x = \frac{p}{p+q}$. Independently, at the same time, Shachnai and Zehavi [114] also observed that the initial proof in [57] could be generalized in essentially the same way as what is stated in Theorem 7.1, and this generalization is used to speed up the algorithms for k -PATH and LONG DIRECTED CYCLE (see Chapters 8 and 9 for definitions of k -PATH and LONG DIRECTED CYCLE).

Basic properties of Representative Family. Before giving computations of representative families in the subsequent sections of this chapter, we first give three lemmata providing basic results about representative families. We prove them for unweighted representative families but they can be easily modified to work for weighted variant.

Lemma 7.1. *Let \mathcal{S} be a family of subsets of a universe U . If $\mathcal{S}' \subseteq_{rep}^q \mathcal{S}$ and $\widehat{\mathcal{S}} \subseteq_{rep}^q \mathcal{S}'$, then $\widehat{\mathcal{S}} \subseteq_{rep}^q \mathcal{S}$.*

Proof. Let $Y \subseteq U$ of size at most q such that there is a set $X \in \mathcal{S}$ disjoint from Y . By the definition of q -representative family we have that there is a set $X' \in \mathcal{S}'$ disjoint from Y . Now the fact that $\widehat{\mathcal{S}} \subseteq_{rep}^q \mathcal{S}'$ yields that there exists a $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y . \square

Lemma 7.2. *Let \mathcal{S} be a family of subsets of a universe U . If $\mathcal{S} = \mathcal{S}_1 \cup \dots \cup \mathcal{S}_\ell$ and $\widehat{\mathcal{S}}_i \subseteq_{rep}^q \mathcal{S}_i$, then $\cup_{i=1}^\ell \widehat{\mathcal{S}}_i \subseteq_{rep}^q \mathcal{S}$.*

Proof. Let $Y \subseteq U$ of size at most q such that there is a set $X \in \mathcal{S}$ disjoint from Y . Since $\mathcal{S} = \mathcal{S}_1 \cup \dots \cup \mathcal{S}_\ell$, there exists an i such that $X \in \mathcal{S}_i$. This implies that there exists a $\widehat{X} \in \widehat{\mathcal{S}}_i \subseteq \cup_{i=1}^\ell \widehat{\mathcal{S}}_i$ disjoint from Y . \square

Lemma 7.3. *Let \mathcal{S}_1 be a p_1 -family and \mathcal{S}_2 be a p_2 -family of subsets of a universe U . Let $\widehat{\mathcal{S}}_1 \subseteq_{rep}^{k-p_1} \mathcal{S}_1$ and $\widehat{\mathcal{S}}_2 \subseteq_{rep}^{k-p_2} \mathcal{S}_2$. Then $\widehat{\mathcal{S}}_1 \bullet \widehat{\mathcal{S}}_2 \subseteq_{rep}^{k-p_1-p_2} \mathcal{S}_1 \bullet \mathcal{S}_2$.*

Proof. Let $Y \subseteq U$ of size at most $q = k - p_1 - p_2$ such that there is a set $X \in \mathcal{S}_1 \bullet \mathcal{S}_2$ disjoint from Y . This implies that there exist $X_1 \in \mathcal{S}_1$ and $X_2 \in \mathcal{S}_2$ such that $X_1 \cup X_2 = X$ and $X_1 \cap X_2 = \emptyset$. Since $\widehat{\mathcal{S}}_1 \subseteq_{rep}^{k-p_1} \mathcal{S}_1$, we have that there exists a $\widehat{X}_1 \in \widehat{\mathcal{S}}_1$ such that $\widehat{X}_1 \cap (X_2 \cup Y) = \emptyset$. Now since $\widehat{\mathcal{S}}_2 \subseteq_{rep}^{k-p_2} \mathcal{S}_2$, we have that there exists a $\widehat{X}_2 \in \widehat{\mathcal{S}}_2$ such that $\widehat{X}_2 \cap (\widehat{X}_1 \cup Y) = \emptyset$. This shows that $\widehat{X}_1 \cup \widehat{X}_2 \in \widehat{\mathcal{S}}_1 \bullet \widehat{\mathcal{S}}_2$. Thus $\widehat{\mathcal{S}}_1 \bullet \widehat{\mathcal{S}}_2 \subseteq_{rep}^{k-p_1-p_2} \mathcal{S}_1 \bullet \mathcal{S}_2$. \square

7.1 Computation using Lopsided Universal Sets

Our aim in this subsection is to prove the following theorem using Lemma 5.5 (computation of n - p - q -lopsided universal family).

Theorem 7.2. *There is an algorithm that given a family \mathcal{A} of p -sets over a universe U of size n and an integer q , computes in time $|\mathcal{A}| \cdot \binom{p+q}{p} \cdot 2^{o(p+q)} \cdot \log n$ a subfamily $\mathcal{A}' \subseteq \mathcal{A}$ such that $|\mathcal{A}'| \leq \binom{p+q}{p} \cdot 2^{o(p+q)} \cdot \log n$ and \mathcal{A}' q -represents \mathcal{A} .*

Proof. The algorithm starts by constructing an $n-p-q$ -lopsided universal family \mathcal{F} as guaranteed by Lemma 5.5. If $|\mathcal{A}| \leq |\mathcal{F}|$ the algorithm outputs \mathcal{A} and halts. Otherwise it builds the set \mathcal{A}' as follows. Initially \mathcal{A}' is equal to \emptyset and all sets in \mathcal{F} are marked as unused. The algorithm goes through every $A \in \mathcal{A}$ and unused sets $F \in \mathcal{F}$. If an unused set $F \in \mathcal{F}$ is found such that $A \subseteq F$, the algorithm marks F as used, inserts A into \mathcal{A}' and proceeds to the next set in \mathcal{A} . If no such set F is found the algorithm proceeds to the next set in \mathcal{A} without inserting A into \mathcal{A}' .

The size of \mathcal{A}' is upper bounded by $|\mathcal{F}| \leq \binom{p+q}{p} \cdot 2^{o(p+q)} \cdot \log n$ since every time a set is added to \mathcal{A}' an unused set in \mathcal{F} is marked as used. For the running time analysis, constructing \mathcal{F} takes time $\binom{p+q}{p} \cdot 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot n \log n$. Then we run through all of \mathcal{F} for each set $A \in \mathcal{A}$, spending time $|\mathcal{A}| \cdot |\mathcal{F}| \cdot (p+q)^{\mathcal{O}(1)}$, which is at most $|\mathcal{A}| \cdot \binom{p+q}{p} \cdot 2^{o(p+q)} \cdot \log n$. Thus in total the running time is bounded by $|\mathcal{A}| \cdot \binom{p+q}{p} \cdot 2^{o(p+q)} \cdot \log n$.

Finally we need to argue that \mathcal{A}' q -represents \mathcal{A} . Consider any set $A \in \mathcal{A}$ and B such that $|B| = q$ and $A \cap B = \emptyset$. If $A \in \mathcal{A}'$ we are done, so assume that $A \notin \mathcal{A}'$. Since \mathcal{F} is $n-p-q$ -lopsided universal there is a set $F \in \mathcal{F}$ such that $A \subseteq F$ and $F \cap B = \emptyset$. Since $A \notin \mathcal{A}'$ we know that F was already marked as used when A was considered by the algorithm. When the algorithm marked F as used it also inserted a set A' into \mathcal{A}' . For the insertion to be made, F must satisfy $A' \subseteq F$. But then $A' \cap B = \emptyset$, completing the proof. \square

One of the factors that drive up the running time of the algorithm in Theorem 7.2 is that one needs to consider all of \mathcal{F} for each set $A \in \mathcal{A}$. Doing some computations it is possible to convince oneself that in an $n-p-q$ -lopsided universal family \mathcal{F} the number of sets $F \in \mathcal{F}$ containing a fixed set A of size p should be approximately $|\mathcal{F}| \cdot \left(\frac{p}{p+q}\right)^p$. Thus, if we could only make sure that this estimation is in fact correct for every $A \in \mathcal{A}$, *and* we could make sure that for a given $A \in \mathcal{A}$ we can list all of the sets in \mathcal{F} that contain A without having to go through the sets that don't, then we could speed up our algorithm by a factor $\left(\frac{p+q}{p}\right)^p$. This is exactly the strategy behind the main theorem of Section 7.2.

7.2 Computation using Separating Collections

In this section we design a faster algorithm to find q -representative family. Our main technical tool is a construction of n - p - q -separating collection. We start with the formal definition of n - p - q -separating collection.

Definition 7.2. An n - p - q -separating collection \mathcal{C} is a tuple $(\mathcal{F}, \chi, \chi')$, where \mathcal{F} is a family of sets over a universe U of size n , χ is a function from $\bigcup_{p' \leq p} \binom{U}{p'}$ to $2^{\mathcal{F}}$ and χ' is a function from $\bigcup_{q' \leq q} \binom{U}{q'}$ to $2^{\mathcal{F}}$ such that the following properties are satisfied

1. for every $A \in \bigcup_{p' \leq p} \binom{U}{p'}$ and $F \in \chi(A)$, $A \subseteq F$,
2. for every $B \in \bigcup_{q' \leq q} \binom{U}{q'}$ and $F \in \chi'(B)$, $F \cap B = \emptyset$,
3. for every pairwise disjoint sets $A_1 \in \binom{U}{p_1}, A_2 \in \binom{U}{p_2}, \dots, A_r \in \binom{U}{p_r}$ and $B \in \binom{U}{q}$ such that $p_1 + \dots + p_r = p$, $\exists F \in \chi(A_1) \cap \chi(A_2) \dots \chi(A_r) \cap \chi'(B)$.

The size of $(\mathcal{F}, \chi, \chi')$ is $|\mathcal{F}|$, the (χ, p') -degree of $(\mathcal{F}, \chi, \chi')$ for $p' \leq p$ is

$$\max_{A \in \binom{U}{p'}} |\chi(A)|,$$

and the (χ', q') -degree of $(\mathcal{F}, \chi, \chi')$ for $q' \leq q$ is

$$\max_{B \in \binom{U}{q'}} |\chi'(B)|.$$

A construction of separating collections is a data structure, that given n , p and q initializes and outputs a family \mathcal{F} of sets over the universe U of size n . After the initialization one can query the data structure by giving it a set $A \in \bigcup_{p' \leq p} \binom{U}{p'}$ or $B \in \bigcup_{q' \leq q} \binom{U}{q'}$, the data structure then outputs a family $\chi(A) \subseteq 2^{\mathcal{F}}$ or $\chi'(B) \subseteq 2^{\mathcal{F}}$ respectively. Together the tuple $\mathcal{C} = (\mathcal{F}, \chi, \chi')$ computed by the data structure should form a n - p - q -separating collection.

We call the time the data structure takes to initialize and output \mathcal{F} the *initialization time*. The (χ, p') -query time, $p' \leq p$, of the data structure is the maximum time the data structure uses to compute $\chi(A)$ over all $A \in \binom{U}{p'}$. Similarly, the (χ', q') -query time, $q' \leq q$, of the data structure is the maximum time the data structure uses to compute $\chi'(B)$ over all $B \in \binom{U}{q'}$. The initialization time of the data structure and the

size of \mathcal{C} are functions of n , p and q . The initialization time is denoted by $\tau_I(n, p, q)$, size of \mathcal{C} is denoted by $\zeta(n, p, q)$. The (χ, p') -query time and (χ, p') -degree of \mathcal{C} , $p' \leq p$, are functions of n, p', p, q and is denoted by $Q_{(\chi, p')}(n, p, q)$ and $\Delta_{(\chi, p')}(n, p, q)$ respectively. Similarly, the (χ', q') -query time and (χ', q') -degree of \mathcal{C} , $q' \leq q$, are functions of n, q', p, q and are denoted by $Q_{(\chi', q')}(n, p, q)$ and $\Delta_{(\chi', q')}(n, p, q)$ respectively. We are now ready to state the main technical tool of this subsection.

Lemma 7.4. *Given $0 < x < 1$, there is a construction of n - p - q -separating collection with the following parameters*

- *size*, $\zeta(n, p, q) \leq 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^{p(1-x)^q}} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n$
- *initialization time*, $\tau_I(n, p, q) \leq 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^{p(1-x)^q}} \cdot (p+q)^{\mathcal{O}(1)} \cdot n \log n$
- *(χ, p') -degree*, $\Delta_{(\chi, p')}(n, p, q) \leq 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^{p-p'(1-x)^q}} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n$
- *(χ, p') -query time*, $Q_{(\chi, p')}(n, p, q) \leq 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^{p-p'(1-x)^q}} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n$
- *(χ', q') -degree*, $\Delta_{(\chi', q')}(n, p, q) \leq 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^{p(1-x)^{q-q'}}} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n$
- *(χ', q') -query time*, $Q_{(\chi', q')}(n, p, q) \leq 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^{p(1-x)^{q-q'}}} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n$

We first give the road map that we take to prove Lemma 7.4. The proof of Lemma 7.4 uses three auxiliary lemmata.

- (a.) **Existential Proof (Lemma 7.5).** This lemma shows that there is indeed a n - p - q -separating collection with the required sizes, degrees and query time. Essentially, it shows that if we form a family $\mathcal{F} = \{F_1, \dots, F_t\}$ of sets of U such that each F_i is a random subset of U where each element is inserted into F_i with probability x , then \mathcal{F} has the desired sizes, degrees and query time. Thus, this also gives a brute force algorithm to design the family \mathcal{F} by just guessing the family of desired size and then checking whether it is indeed a n - p - q -separating collection.
- (b.) **Universe Reduction (Lemma 7.6).** The construction obtained in Lemma 7.5 has only one drawback that the initialization time is much larger than claimed in Lemma 7.4. To overcome this lacuna, we do not apply the construction in Lemma 7.5 directly. We first prove a Lemma 7.6 which helps us in reducing the universe size to $(p+q)^2$. This is done using the known construction of k -perfect hash families of size $(p+q)^{\mathcal{O}(1)} \log n$. However, Lemma 7.6 alone can not reduce the universe size sufficiently, that we can apply the construction of Lemma 7.5.
- (c.) **Splitting Lemma (Lemma 7.7).** We give a splitter type construction in Lemma 7.7 that when applied with Lemma 7.6 makes the universe and other parameters small enough that we can apply the construction given in

Lemma 7.5. In this construction we consider all the “consecutive partitions” of the universe into t parts, assume that the sets $A \cup B$, $A = \cup_{i=1}^r A_i$, are distributed uniformly into t parts and then use this information to obtain a construction of separating collections in each part and then take the product of these collections to obtain a collection for the original instance.

We start with the existential proof.

Lemma 7.5. *Given $0 < x < 1$, there is a construction of n - p - q -separating collections with*

- size $\zeta(n, p, q) = \mathcal{O}\left(\frac{1}{x^p(1-x)^q} \cdot (p^2 + q^2 + 1) \log n\right)$
- initialization time $\tau_I(n, p, q) = \mathcal{O}\left(\binom{2^n}{\zeta(n, p, q)} \cdot \frac{1}{x^p(1-x)^q} \cdot n^{\mathcal{O}(p+q)}\right)$
- (χ, p') -degree for $p' \leq p$, $\Delta_{(\chi, p')}(n, p, q) = \mathcal{O}\left(\frac{1}{x^{p-p'}} \cdot \frac{(p^2+q^2+1)}{(1-x)^q} \cdot \log n\right)$
- (χ, p') -query time $Q_{(\chi, p')}(n, p, q) = \mathcal{O}\left(\frac{1}{x^p(1-x)^q} \cdot n^{\mathcal{O}(1)}\right)$
- (χ', q') -degree $\Delta_{(\chi', q')}(n, p, q) = \mathcal{O}\left(\frac{1}{x^p(1-x)^{q-q'}} \cdot (p^2 + q^2 + 1) \cdot \log n\right)$
- (χ', q') -query time $Q_{(\chi', q')}(n, p, q) = \mathcal{O}\left(\frac{1}{x^p(1-x)^q} \cdot n^{\mathcal{O}(1)}\right)$

Proof. We start by giving a randomized algorithm that with positive probability constructs a n - p - q -separating collection $\mathcal{C} = (\mathcal{F}, \chi, \chi')$ with the desired size and degree parameters. We will then discuss how to deterministically compute such a \mathcal{C} within the required time bound. Set $t = \frac{1}{x^p(1-x)^q} \cdot (p^2 + q^2 + 1) \log n$ and construct the family $\mathcal{F} = \{F_1, \dots, F_t\}$ as follows. Each set F_i is a random subset of U , where each element of U is inserted into F_i with probability x . Distinct elements are inserted (or not) into F_i independently, and the construction of the different sets in \mathcal{F} is also independent. For each $A \in \bigcup_{p' \leq p} \binom{U}{p'}$ we set $\chi(A) = \{F \in \mathcal{F} : A \subseteq F\}$ and for each $B \in \bigcup_{q' \leq q} \binom{U}{q'}$ we set $\chi'(B) = \{F \in \mathcal{F} : F \cap B = \emptyset\}$.

The size of \mathcal{F} is within the required bound by construction. We now argue that with positive probability $(\mathcal{F}, \chi, \chi')$ is indeed a n - p - q -separating collection, and that the degrees of \mathcal{C} is within the required bounds as well. For fixed sets $A \in \binom{U}{p}$, $B \in \binom{U \setminus A}{q}$, and integer $i \leq t$, we consider the probability that $A \subseteq F_i$ and $B \cap F_i = \emptyset$. This

probability is $x^p(1-x)^q$. Since each F_i is constructed independently from the other sets in \mathcal{F} , the probability that *no* F_i satisfies $A \subseteq F_i$ and $B \cap F_i = \emptyset$ is

$$(1 - x^p(1-x)^q)^t \leq e^{-(p^2+q^2+1)\log n} = \frac{1}{n^{p^2+q^2+1}}.$$

For a fixed A_1, \dots, A_r and B (choices in condition 3), the probability that no F_i is in $\chi(A_1) \cap \chi(A_2) \cap \dots \cap \chi(A_r) \cap \chi'(B)$ is equal to the probability that no F_i is in $\chi(A_1 \cup A_2 \dots \cup A_r) \cap \chi'(B)$ (since $\chi(A')$ contains all the sets in \mathcal{F} that contains A' and $\chi'(B)$ contains all the sets in \mathcal{F} that are disjoint from B). Hence the probability that condition 3 fails is upper bounded by

$$Y \cdot \frac{1}{n^{p^2+q^2+1}}$$

where Y is the number of choices for A_1, \dots, A_r and B in condition 3. We upper bound Y as follows. There are $\binom{n}{p}$ choices for $A_1 \cup \dots \cup A_r$ and $\binom{n}{q}$ choices for B . For each choice of $A_1 \cup \dots \cup A_r$ there are at most r^p choices of making A_1, \dots, A_r with some of them being empty as well. Note that $r \leq p$. Therefore the number of possible choices of sets A_1, A_2, \dots, A_r and B in condition 3 is upper bounded by $\binom{n}{p} \binom{n}{q} p^p \leq n^{2p+q} \leq n^{p^2+q^2}$. Hence the probability that condition 3 in Definition 7.2 fails is at most $\frac{1}{n}$.

We also need to upper bound the maximum degree of \mathcal{C} . For every $A \in \binom{U}{p'}$, $|\chi(A)|$ is a random variable. For a fixed $A \in \binom{U}{p'}$ and $i \leq t$ the probability that $A \subseteq F_i$ is exactly $x^{p'}$. Hence $|\chi(A)|$ is the sum of t independent 0/1-random variables that each take value 1 with probability $x^{p'}$. Hence the expected value of $|\chi(A)|$ is

$$E[|\chi(A)|] = t \cdot x^{p'} = \frac{1}{x^{p-p'}(1-x)^q} \cdot (p^2 + q^2 + 1) \log n$$

For every $B \in \binom{U}{q'}$, $|\chi'(B)|$ is also a random variable. For a fixed $B \in \binom{U}{q'}$ and $i \leq t$ the probability that $A \cap F_i = \emptyset$ is exactly $(1-x)^{q'}$. Hence the expected value of

$|\chi'(B)|$ is,

$$E[|\chi'(B)|] = t \cdot (1-x)^{q'} = \frac{1}{x^p(1-x)^{q-q'}} \cdot (p^2 + q^2 + 1) \log n.$$

Standard Chernoff bounds [96, Theorem 4.4] show that the probability that for any $A \in \binom{U}{p}$, $|\chi(A)|$ is at least $6E[|\chi(A)|]$ is upper bounded by $2^{-6E[|\chi(A)|]} \leq \frac{1}{n^{p^2+q^2+1}}$. Similarly the probability that for any $B \in \binom{U}{q'}$, $|\chi'(B)|$ is at least $6E[|\chi'(B)|]$ is upper bounded by $2^{-6E[|\chi'(B)|]} \leq \frac{1}{n^{p^2+q^2+1}}$. There are $\sum_{p' \leq p} \binom{n}{p'} \leq n^{p^2}$ choices for $A \in \bigcup_{p' \leq p} \binom{U}{p'}$ and $\sum_{q' \leq q} \binom{n}{q'} \leq n^{q^2}$ choices for $B \in \bigcup_{q' \leq q} \binom{U}{q'}$. Hence the union bound yields that the probability that there exists an $A \in \bigcup_{p' \leq p} \binom{U}{p'}$ such that $|\chi(A)| > 6E[|\chi(A)|]$ or there exists $B \in \bigcup_{q' \leq q} \binom{U}{q'}$ such that $|\chi'(B)| > 6E[|\chi'(B)|]$ is upper bounded by $\frac{1}{n}$. Thus \mathcal{C} is a family of n - p - q -separating collections with the desired size and degree parameters with probability at least $1 - \frac{2}{n} > 0$. The degenerate case that $1 - \frac{2}{n} \leq 0$ is handled by the family \mathcal{F} containing all (at most four) subsets of U .

To construct \mathcal{F} within the stated initialization time bound, it is sufficient to try all families \mathcal{F} of size t and for each of the $\binom{2^n}{\zeta(n,p,q)}$ guesses, test whether it is indeed a family of n - p - q -separating collections in time $\mathcal{O}(t \cdot n^{\mathcal{O}(p+q)}) = \mathcal{O}(\frac{1}{x^p(1-x)^q} \cdot n^{\mathcal{O}(p+q)})$.

For the queries, we need to give an algorithm that given A , computes $\chi(A)$ (or $\chi'(A)$), under the assumption that \mathcal{F} has already been computed in the initialization step. This is easily done within the stated running time bound by going through every set $F \in \mathcal{F}$, checking whether $A \subseteq F$ (or $A \cap F = \emptyset$), and if so, inserting F into $\chi(A)$ ($\chi'(A)$). This concludes the proof. \square

We will now work towards improving the time bounds of Lemma 7.5.

Lemma 7.6. *If there is a construction of n - p - q -separating collections $(\hat{\mathcal{F}}, \hat{\chi}, \hat{\chi}')$ with initialization time $\tau_I(n, p, q)$, size $\zeta(n, p, q)$, $(\hat{\chi}, p')$ -query time $Q_{(\hat{\chi}, p')}(n, p, q)$, $(\hat{\chi}', q')$ -query time $Q_{(\hat{\chi}', q')}(n, p, q)$, $(\hat{\chi}, p')$ -degree $\Delta_{(\hat{\chi}, p')}(n, p, q)$, and $(\hat{\chi}', q')$ -degree $\Delta_{(\hat{\chi}', q')}(n, p, q)$ then there is a construction of n - p - q -separating collections with fol-*

lowing parameters.

- $\zeta'(n, p, q) \leq \zeta((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n,$
- $\tau_I'(n, p, q) = \mathcal{O}(\tau_I((p+q)^2, p, q) + \zeta((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot n \log n),$
- $\Delta'_{(\hat{x}, p')}(n, p, q) \leq \Delta_{(\hat{x}, p')}((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n,$
- $Q'_{(\hat{x}, p')}(n, p, q) =$
 $\mathcal{O}((Q_{(\hat{x}, p')}((p+q)^2, p, q) + \Delta_{(\hat{x}, p')}((p+q)^2, p, q)) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n),$
- $\Delta'_{(\hat{x}', q')}(n, p, q) \leq \Delta_{(\hat{x}', q')}((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n,$
- $Q'_{(\hat{x}', q')}(n, p, q) =$
 $\mathcal{O}((Q_{(\hat{x}', q')}((p+q)^2, p, q) + \Delta_{(\hat{x}', q')}((p+q)^2, p, q)) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n)$

Proof. We give a construction of n - p - q -separating collections with initialization time, query time, size and degree τ_I' , Q' , ζ' and Δ' respectively using the construction with initialization time, query time, size and degree τ_I , Q , ζ and Δ as a black box.

We first describe the initialization of the data structure. Given n , p , and q , we construct using [Theorem 5.2](#) a $(p+q)$ -perfect family f_1, \dots, f_t of hash functions from the universe U to $[(p+q)^2]$. The construction takes time $\mathcal{O}((p+q)^{\mathcal{O}(1)} n \log n)$ and $t \leq (p+q)^{\mathcal{O}(1)} \cdot \log n$. We will store these hash functions in memory. We use the following notations.

- For a set $S \subseteq U$ and $T \subseteq [(p+q)^2]$,
 $f_i(S) = \{f_i(s) : s \in S\}$ and $f_i^{-1}(T) = \{s \in U : f_i(s) \in T\}.$
- For a family \mathcal{Z} of sets over U and family \mathcal{W} of sets over $[(p+q)^2]$,
 $f_i(\mathcal{Z}) = \{f_i(S) : S \in \mathcal{Z}\}$ and $f_i^{-1}(\mathcal{W}) = \{f_i^{-1}(T) : T \in \mathcal{W}\}.$

We first use the given black box construction for $(p+q)^2$ - p - q -separating collections $(\hat{\mathcal{F}}, \hat{\chi}, \hat{\chi}')$ over the universe $[(p+q)^2]$. We run the initialization algorithm of this

construction and store the family $\hat{\mathcal{F}}$ in memory. We then set

$$\mathcal{F} = \bigcup_{i \leq t} f_i^{-1}(\hat{\mathcal{F}}).$$

We spent $\mathcal{O}((p+q)^{\mathcal{O}(1)} n \log n)$ time to construct a $(p+q)$ -perfect family of hash functions, $\mathcal{O}(\tau_I((p+q)^2, p, q))$ to construct $\hat{\mathcal{F}}$ of size $\zeta((p+q)^2, p, q)$, and $\mathcal{O}(\zeta((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot n \log n)$ time to construct \mathcal{F} from $\hat{\mathcal{F}}$ and the family of perfect hash functions. Thus the upper bound on $\tau'_I(n, p, q)$ follows. Furthermore, $|\mathcal{F}| \leq |\hat{\mathcal{F}}| \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n$, yielding the claimed bound for ζ' .

We now define $\chi(A)$ for every $A \in \bigcup_{p' \leq p} \binom{U}{p'}$ and describe the query algorithm. For every $A \in \bigcup_{p' \leq p} \binom{U}{p'}$ we let

$$\chi(A) = \bigcup_{\substack{i \leq t \\ |f_i(A)|=|A|}} f_i^{-1}(\hat{\chi}(f_i(A))).$$

Since for every $\hat{F} \in \hat{\chi}(f_i(A))$, $f_i(A) \subseteq \hat{F}$, it follows that $A \subseteq F$ for every $F \in \chi(A)$. Furthermore we can bound $|\chi(A)|$ for any $A \in \bigcup_{p' \leq p} \binom{U}{p'}$, as follows

$$|\chi(A)| \leq \sum_{\substack{i \leq t \\ |f_i(A)|=|A|}} |\hat{\chi}(f_i(A))| \leq \Delta_{(\hat{\chi}, p')}((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n.$$

Thus the claimed bound for $\Delta'_{(\chi, p')}$ follows. Similarly, way can define $\chi'(B)$ for every $B \in \bigcup_{q' \leq q} \binom{U}{q'}$ as

$$\chi'(B) = \bigcup_{\substack{i \leq t \\ |f_i(A)|=|A|}} f_i^{-1}(\hat{\chi}'(f_i(A))).$$

$$|\chi'(B)| \leq \sum_{\substack{i \leq t \\ |f_i(A)|=|A|}} |\hat{\chi}'(f_i(A))| \leq \Delta_{(\hat{\chi}', q')}((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n.$$

To compute $\chi(A)$ for any $A \in \bigcup_{p' \leq p} \binom{U}{p'}$, we go over every $i \leq t$ and check whether f_i is injective on A . This takes time $\mathcal{O}((p+q)^{\mathcal{O}(1)} \cdot \log n)$. For each i such that f_i is injective on A , we compute $f_i(A)$ and then $\hat{\chi}(f_i(A))$ in time $\mathcal{O}(Q_{(\hat{\chi}, p')}((p+q)^2, p, q))$. Then we compute $f_i^{-1}(\hat{\chi}(f_i(A)))$ in time $\mathcal{O}(|\hat{\chi}(f_i(A))| \cdot (p+q)^{\mathcal{O}(1)}) = \mathcal{O}(\Delta_{(\hat{\chi}, p')}((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)})$ and add this set to $\chi(A)$. As we need to do this $\mathcal{O}((p+q)^{\mathcal{O}(1)} \cdot \log n)$ times, the total time to compute $\chi(A)$ is upper bounded by $\mathcal{O}((Q_{(\hat{\chi}, p')}((p+q)^2, p, q) + \Delta_{(\hat{\chi}, p')}((p+q)^2, p, q)) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n)$, yielding the claimed upper bound on $Q'_{(\chi, p')}$. Similar way we can bound $Q'_{(\chi', q')}$.

It remains to argue that $(\mathcal{F}, \chi, \chi')$ is in fact a n - p - q -separating collection. For any r , consider pairwise disjoint sets $A_1 \in \binom{U}{p_1}, \dots, A_r \in \binom{U}{p_r}$, and $B \in \binom{U}{q}$ such that $p_1 + \dots + p_r = p$. We need to show that there is $F \in \chi(A_1) \cap \dots \cap \chi(A_r) \cap \chi'(B)$. Since f_1, \dots, f_t is a $(p+q)$ -perfect family of hash functions, there is an i such that f_i is injective on $A_1 \cup \dots \cup A_r \cup B$. Since $(\hat{\mathcal{F}}, \hat{\chi}, \hat{\chi}')$ is a $(p+q)^2$ - p - q -separating collection, $\exists \hat{F} \in \hat{\chi}(f_i(A_1)) \cap \dots \cap \hat{\chi}(f_i(A_r)) \cap \hat{\chi}'(f_i(B))$. Since f_i is injective on A_1, \dots, A_r and B , $f_i^{-1}(\hat{F}) \in \chi(A_1) \cap \dots \cap \chi(A_r) \cap \chi'(B)$. This concludes the proof. \square

We now give a *splitting lemma*, which allows us to reduce the problem of finding n - p - q -separating collections to the same problem, but with much smaller values for p and q .

A *partition* of U is a family $\mathcal{U}_P = \{U_1, U_2, \dots, U_t\}$ of sets over U such that $U_i \cap U_j = \emptyset$ for every $i \neq j$ and $U = \bigcup_{i \leq t} U_i$. Each of the sets U_i are called the *parts* of the partition. A *consecutive partition* of $\{1, \dots, n\}$ is a partition $\mathcal{U}_P = \{U_1, U_2, \dots, U_t\}$ of $\{1, \dots, n\}$ such that for every integer $i \leq t$ and integers $1 \leq x \leq y \leq z$, if $x \in U_i$ and $z \in U_i$ then $y \in U_i$ as well. In other words, in a consecutive partition each part is a consecutive interval of integers. For every integer t , let \mathcal{P}_t^n denote the collection of all consecutive partitions of $\{1, \dots, n\}$ with exactly t parts. We do not demand that all of the parts in a partition in \mathcal{P}_t^n are non-empty. Simple counting arguments show that for every t , $|\mathcal{P}_t^n| = \binom{n+t-1}{t-1}$.

We will denote by $\mathcal{Z}_{s,t}^p$ the set of all t -tuples (p_1, p_2, \dots, p_t) of integers such that $\sum_{i \leq t} p_i = p$ and $0 \leq p_i \leq s$ for all i . Clearly $|\mathcal{Z}_{s,t}^p| \leq \binom{p+t-1}{t-1}$, since this counts all the ways of writing p as a sum of t non-negative integers, without considering the upper bound on each one. For an ease of convenience we summarize the above in the next definition and the proposition.

Definition 7.3. A partition of U is a family $\mathcal{U}_P = \{U_1, U_2, \dots, U_t\}$ of sets over

U such that $\forall i \neq j, U_i \cap U_j = \emptyset$ and $U = \bigcup_{i \leq t} U_i$. Each of the sets U_i are called the parts of the partition. A consecutive partition of $\{1, \dots, n\}$ is a partition $\mathcal{U}_P = \{U_1, U_2, \dots, U_t\}$ of $\{1, \dots, n\}$ such that for every integer $i \leq t$ and integers $1 \leq x \leq y \leq z$, if $x \in U_i$ and $z \in U_i$ then $y \in U_i$ as well.

Proposition 7.1. Let \mathcal{P}_t^n denote the collection of all consecutive partitions of $\{1, \dots, n\}$ with exactly t parts. Let $\mathcal{Z}_{s,t}^p$ be the set of all t -tuples (p_1, p_2, \dots, p_t) of integers such that $\sum_{i \leq t} p_i = p$ and $0 \leq p_i \leq s$ for all i . Then for every t , $|\mathcal{P}_t^n| = \binom{n+t-1}{t-1}$ and $|\mathcal{Z}_{s,t}^p| \leq \binom{p+t-1}{t-1}$.

Lemma 7.7. For any p, q let $s = \lfloor (\log(p+q))^2 \rfloor$ and $t = \lceil \frac{p+q}{s} \rceil$. If there is a construction of n - p - q -separating collections $(\mathcal{F}_p, \chi_p, \chi'_p)$

- with size $\zeta(n, p, q)$ and initialization time $\tau_I(n, p, q)$,
- (χ_p, p') -degree $\Delta_{(\chi_p, p')}(n, p, q)$ and (χ'_p, q') -degree $\Delta_{(\chi'_p, q')}(n, p, q)$, and
- query times $Q_{(\chi_p, p')}(n, p, q)$ and $Q_{(\chi'_p, q')}(n, p, q)$,

then there is a construction of n - p - q -separating collection with following parameters

•

$$\zeta'(n, p, q) \leq |\mathcal{P}_t^n| \cdot \sum_{(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p} \prod_{i \leq t} \zeta(n, p_i, s - p_i),$$

•

$$\tau'_I(n, p, q) = \mathcal{O}\left(\sum_{\substack{\hat{p} \leq s, p \\ s - \hat{p} \leq q}} \tau_I(n, \hat{p}, s - \hat{p})\right) + \zeta'(n, p, q) \cdot n^{\mathcal{O}(1)},$$

•

$$\Delta'_{(\chi, p')}(n, p, q) \leq \Delta^*_{(\chi, p')}(n, p, q) = |\mathcal{P}_t^n| \cdot |\mathcal{Z}_{s,t}^p| \cdot \max_{\substack{(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p \\ p'_1 \leq p_1, \dots, p'_t \leq p_t \\ p'_1 + \dots + p'_t = p'}} \prod_{i \leq t} \Delta_{(\chi_{p_i}, p'_i)}(n, p_i, s - p_i),$$

•

$$Q'_{(\chi, p')} (n, p, q) = \mathcal{O}\left(\Delta^*_{(\chi, p')} (n, p, q) \cdot n^{\mathcal{O}(1)} + \right. \\ \left. |\mathcal{P}_t^n| \cdot |\mathcal{Z}_{s,t}^p| \cdot t \cdot \left(\max_{\substack{\hat{p}' \leq \hat{p} \leq s \\ \hat{p} - \hat{p}' \leq p - p' \\ s - \hat{p} \leq q}} Q_{(\chi_{\hat{p}}, \hat{p}')} (n, \hat{p}, s - \hat{p}) \right) \right),$$

•

$$\Delta'_{(\chi', q')} (n, p, q) \leq \Delta^*_{(\chi', q')} (n, p, q) \\ = |\mathcal{P}_t^n| \cdot |\mathcal{Z}_{s,t}^p| \cdot \max_{\substack{(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p \\ q'_1 \leq s - p_1, \dots, q'_t \leq s - p_t \\ q'_1 + \dots + q'_t = q'}} \prod_{i=1}^t \Delta_{(\chi'_{p_i}, q'_i)} (n, p_i, s - p_i),$$

•

$$Q'_{(\chi', q')} (n, p, q) = \mathcal{O}\left(\Delta^*_{(\chi', q')} (n, p, q) \cdot n^{\mathcal{O}(1)} + \right. \\ \left. |\mathcal{P}_t^n| \cdot |\mathcal{Z}_{s,t}^p| \cdot t \cdot \left(\max_{\substack{\hat{q}' \leq \hat{q} \leq s \\ \hat{q} - \hat{q}' \leq q - q' \\ s - \hat{q} \leq p}} Q_{(\chi'_{s-\hat{q}}, \hat{q}')} (n, s - \hat{q}, \hat{q}) \right) \right).$$

Proof. Set $s = \lfloor (\log(p + q))^2 \rfloor$ and $t = \lceil \frac{p+q}{s} \rceil$. We will give a construction of n - p - q -separating collections with initialization time, query time, size and degree within the claimed bounds above. In this construction we will use the given construction as a black box. We may assume without loss of generality that $U = \{1, \dots, n\}$. Our algorithm first runs for every \hat{p} , $0 \leq \hat{p} \leq s, \hat{p} \leq p, s - \hat{p} \leq q$, and initializes n - \hat{p} - $(s - \hat{p})$ -separating collections,

$$(\mathcal{F}_{\hat{p}}, \chi_{\hat{p}}, \chi'_{\hat{p}}).$$

These will be the building blocks of our construction. For a family of sets \mathcal{A} over a universe U and subset $U' \subseteq U$ we define $\mathcal{A} \cap U' = \{A \cap U' : A \in \mathcal{A}\}$. We now

define \mathcal{F} as follows.

$$\mathcal{F} = \bigcup_{\substack{\{U_1, \dots, U_t\} \in \mathcal{P}_t^n \\ (p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p \text{ such that} \\ \forall i : s - p_i \leq q}} (\mathcal{F}_{p_1} \sqcap U_1) \circ (\mathcal{F}_{p_2} \sqcap U_2) \circ \dots \circ (\mathcal{F}_{p_t} \sqcap U_t) \quad (7.1)$$

It follows directly from the definition of \mathcal{F} that $|\mathcal{F}|$ is within the claimed bound for $\zeta'(n, p, q)$. For the initialization time, the algorithm spends $\mathcal{O}\left(\sum_{\substack{\hat{p} \leq s, p \\ s - \hat{p} \leq q}} \tau_I(n, \hat{p}, s - \hat{p})\right)$ time to initialize the constructions of the $n - \hat{p} - (s - \hat{p})$ -separating collections for all $\hat{p} \leq s$ such that $\hat{p} \leq p$ and $s - \hat{p} \leq q$ together. Now the algorithm can output the entries of \mathcal{F} one set at a time by using Equation (7.1), spending $n^{\mathcal{O}(1)}$ time per output set. Hence the time bound for $\tau'_I(n, p, q)$ follows.

For every set $A \in \bigcup_{p' \leq p} \binom{U}{p'}$ we define $\chi(A)$ as follows.

$$\chi(A) = \bigcup_{\substack{\{U_1, \dots, U_t\} \in \mathcal{P}_t^n \\ (p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p \text{ such that} \\ \forall U_i : |U_i \cap A| \leq p_i, s - p_i \leq q}} \left[(\chi_{p_1}(A \cap U_1) \sqcap U_1) \circ (\chi_{p_2}(A \cap U_2) \sqcap U_2) \circ \dots \right. \quad (7.2) \\ \left. \dots \circ (\chi_{p_t}(A \cap U_t) \sqcap U_t) \right]$$

Now we show that $\chi(A) \subseteq \mathcal{F}$. From the definition of $n - p_i - (s - p_i)$ -separating collections $(\mathcal{F}_{p_i}, \chi_{p_i}, \chi'_{p_i})$, each family $\chi_{p_i}(A \cap U_i)$ in Equation (7.2) is a subset of \mathcal{F}_{p_i} . This implies that $\chi_{p_i}(A \cap U_i) \sqcap U_i \subseteq \mathcal{F}_{p_i} \sqcap U_i$. Hence $\chi(A) \subseteq \mathcal{F}$. Similarly we can define $\chi'(B)$ for any $B \in \bigcup_{q' \leq q} \binom{U}{q'}$ as

$$\chi'(B) = \bigcup_{\substack{\{U_1, \dots, U_t\} \in \mathcal{P}_t^n \\ (p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p \text{ such that} \\ \forall U_i : |U_i \cap B| \leq s - p_i \leq q}} \left[(\chi'_{p_1}(B \cap U_1) \sqcap U_1) \circ (\chi'_{p_2}(B \cap U_2) \sqcap U_2) \circ \dots \right. \quad (7.3) \\ \left. \dots \circ (\chi'_{p_t}(B \cap U_t) \sqcap U_t) \right]$$

Similar to the proof of $\chi(A) \subseteq \mathcal{F}$, we can show that $\chi'(B) \subseteq \mathcal{F}$. It follows directly from the definition of $\chi(A)$ and $\chi'(B)$ that $|\chi(A)|$ and $|\chi'(B)|$ is within the

claimed bound for $\Delta'_{(\chi,p')}(n,p,q)$ and $\Delta'_{(\chi',q')}(n,p,q)$ respectively. We now describe how queries $\chi(A)$ can be answered, and analyze how much time it takes. Given A we will compute $\chi(A)$ using Equation (7.2). Let $|A| = p'$. For each $\{U_1, \dots, U_t\} \in \mathcal{P}_t^n$ and $(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p$ such that $p'_i = |U_i \cap A| \leq p_i, s - p_i \leq q$ for all $i \leq t$, we proceed as follows. First we compute $\chi_{p_i}(A \cap U_i)$ for each $i \leq t$, spending in total $\mathcal{O}(\sum_{i \leq t} Q_{(\chi_{p_i}, p'_i)}(n, p_i, s - p_i))$ time. Now we add each set in

$$(\chi_{p_1}(A \cap U_1) \sqcap U_1) \circ (\chi_{p_2}(A \cap U_2) \sqcap U_2) \circ \dots \circ (\chi_{p_t}(A \cap U_t) \sqcap U_t)$$

to $\chi(A)$, spending $n^{\mathcal{O}(1)}$ time per set, yielding the bound below,

$$\begin{aligned} Q'_{(\chi,p')}(n,p,q) &\leq \mathcal{O}\left(\Delta^*_{(\chi,p')}(n,p,q) \cdot n^{\mathcal{O}(1)} + \right. \\ &\quad \left. \sum_{\substack{\{U_1, \dots, U_t\} \in \mathcal{P}_t \\ (p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p \text{ such that} \\ \forall U_i : p'_i = |U_i \cap A| \leq p_i, s - p_i \leq q}} \left[\sum_{i \leq t} Q_{(\chi_{p_i}, p'_i)}(n, p_i, s - p_i) \right] \right) \\ &\leq \mathcal{O}\left(\Delta^*_{(\chi,p')}(n,p,q) \cdot n^{\mathcal{O}(1)} + \right. \\ &\quad \left. |\mathcal{P}_t^n| \cdot |\mathcal{Z}_{s,t}^p| \cdot \max_{\substack{(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p \\ p'_1 \leq p_1, \dots, p'_t \leq p_t \text{ such that} \\ p'_1 + \dots + p'_t = p', \forall i: s - p_i \leq q}} \left(\sum_{i \leq t} Q_{(\chi_{p_i}, p'_i)}(n, p_i, s - p_i) \right) \right) \\ &\leq \mathcal{O}\left(\Delta^*_{(\chi,p')}(n,p,q) \cdot n^{\mathcal{O}(1)} + \right. \\ &\quad \left. |\mathcal{P}_t^n| \cdot |\mathcal{Z}_{s,t}^p| \cdot t \cdot \max_{\substack{(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p \\ p'_1 \leq p_1, \dots, p'_t \leq p_t \text{ such that} \\ p'_1 + \dots + p'_t = p', \forall i: s - p_i \leq q}} \left(Q_{(\chi_{p_i}, p'_i)}(n, p_i, s - p_i) \right) \right) \\ &\leq \mathcal{O}\left(\Delta^*_{(\chi,p')}(n,p,q) \cdot n^{\mathcal{O}(1)} + \right. \\ &\quad \left. |\mathcal{P}_t^n| \cdot |\mathcal{Z}_{s,t}^p| \cdot t \cdot \left(\max_{\substack{\hat{p}' \leq \hat{p} \leq s \\ \hat{p} - \hat{p}' \leq p - p' \\ s - \hat{p} \leq q}} Q_{(\chi_{\hat{p}}, \hat{p}')} (n, \hat{p}, s - \hat{p}) \right) \right) \end{aligned}$$

For any $(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p$ and $p'_1 \leq p_1, \dots, p'_t \leq p_t$ such that $\sum_{i=1}^t p'_i = p'$, we have that $\sum_{i=1}^t p_i - p'_i = p - p'$ and so $p_i - p'_i \leq p - p'$ for all i . This shows the correctness of the last inequality in the above query time analysis.

By doing similar analysis, we get required bound for $Q'_{(\chi',q')}$. We now need to argue

that $(\mathcal{F}, \chi, \chi')$ is in fact a n - p - q -separating collection. For any r , consider pairwise disjoint sets $A_1 \in \binom{U}{b_1}, \dots, A_r \in \binom{U}{b_r}$ and $B \in \binom{U}{q}$ such that $b_1 + \dots + b_r = p$. Let $A = A_1 \cup \dots \cup A_r$. There exists a consecutive partition $\{U_1, \dots, U_t\} \in \mathcal{P}_t^n$ of U such that for every $i \leq t$ we have that $|(A \cup B) \cap U_i| \leq \lceil \frac{p+q}{t} \rceil = s$. For each $i \leq t$ set $p_i = |A \cap U_i|$ and $q_i = |B \cap U_i| = s - p_i$. Note that $p_i \leq p$ and $q_i \leq q$ for all i . For every $i \leq t$ the tuple $(\mathcal{F}_{p_i}, \chi_{p_i}, \chi'_{p_i})$ form a n - p_i - q_i -separating collection. Hence there exists a $F_i \in \chi_{p_i}(A_1 \cap U_i) \cap \dots \cap \chi_{p_i}(A_r \cap U_i) \cap \chi'_{p_i}(B \cap U_i)$ because $|A_1 \cap U_i| + \dots + |A_r \cap U_i| = p_i$, $|B \cap U_i| = q_i$ and $(\mathcal{F}_{p_i}, \chi_{p_i}, \chi'_{p_i})$ is a n - p_i - q_i -separating collection. That is $F_i \in \chi_{p_i}(A_j \cap U_i)$ for all $j \leq r$ and $F_i \in \chi'_{p_i}(B \cap U_i)$. Let $F = \bigcup_{i \leq t} F_i \cap U_i$. By construction of χ and χ' , $F \in \chi(A_j)$ for all $j \leq r$ and $F \in \chi'(B)$. Hence $F \in \chi(A_1) \cap \dots \cap \chi(A_r) \cap \chi'(B)$. This completes the proof \square

Now we are ready to prove Lemma 7.4. We restate the lemma for easiness of presentation.

Lemma 7.4 *Given $0 < x < 1$, there is a construction of n - p - q -separating collection with the following parameters*

- *size:* $\zeta(n, p, q) \leq 2^{O(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^p(1-x)^q} \cdot (p+q)^{O(1)} \cdot \log n$
- *initialization time:* $\tau_I(n, p, q) \leq 2^{O(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^p(1-x)^q} \cdot (p+q)^{O(1)} \cdot n \log n$
- *(χ, p') -degree:* $\Delta_{(\chi, p')}(n, p, q) \leq 2^{O(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^{p-p'}(1-x)^q} \cdot (p+q)^{O(1)} \cdot \log n$
- *(χ, p') -query time:* $Q_{(\chi, p')}(n, p, q) \leq 2^{O(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^{p-p'}(1-x)^q} \cdot (p+q)^{O(1)} \cdot \log n$
- *(χ', q') -degree:* $\Delta_{(\chi', q')}(n, p, q) \leq 2^{O(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^p(1-x)^{q-q'}} \cdot (p+q)^{O(1)} \cdot \log n$
- *(χ', q') -query time:* $Q_{(\chi', q')}(n, p, q) \leq 2^{O(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^p(1-x)^{q-q'}} \cdot (p+q)^{O(1)} \cdot \log n$

Proof. We first explain a brute force construction of n - p - q -separating collection when the value of x is close to 0 or close to 1. These are discussed in Cases 1 and 2 and the result for all other values of x is explained in Case 3. Let U be the universe.

Case 1: $x \leq \frac{1}{n}$. In this case the algorithm will output all subset of size p of the universe as the family \mathcal{F} of sets in the n - p - q - separating collection. That is $\mathcal{F} = \{F \subseteq U \mid |F| = p\}$. We define χ and χ' as follows. For any $A \in \bigcup_{p' \leq p} \binom{U}{p'}$, $\chi(A) = \{F \in \mathcal{F} \mid A \subseteq F\}$. For any $B \in \bigcup_{q' \leq q} \binom{U}{q'}$, $\chi'(B) = \{F \in \mathcal{F} \mid B \cap F = \emptyset\}$. It is easy to see that $(\mathcal{F}, \chi, \chi')$ is a n - p - q - separating collection. Note that $|\mathcal{F}| = \binom{n}{p} \leq n^p$. Since $n \leq \frac{1}{x}$, the size of the n - p - q - separating collection is upperbound by the claimed bound. Since we can list all the elements in \mathcal{F} in n^p time, the initialization time is upper bounded by the claimed bound. For any $A \subseteq U$, $|A| = p'$, the cardinality of $\chi(A)$ is exactly equal to $\binom{n}{p-p'}$ which is upper bounded by $\frac{1}{x^{p-p'}}$. Thus the (χ, p') -degree and (χ, p') -query time is bounded by the claimed bound. For any $B \subseteq U$, $|B| = q'$, the cardinality of $\chi'(B)$ is at most $|\mathcal{F}|$, which is upper bounded by $\frac{1}{x^p}$. Thus the (χ', q') -degree and (χ', q') -query time is bounded by the claimed bound.

Case 2: $1 - x \leq \frac{1}{n}$. In this case the algorithm will output all subset of size $n - q$ of the universe as the family \mathcal{F} of sets in the n - p - q - separating collection. That is $\mathcal{F} = \{F \subseteq U \mid |F| = n - q\}$. We define χ and χ' as follows. For any $A \in \bigcup_{p' \leq p} \binom{U}{p'}$, $\chi(A) = \{F \in \mathcal{F} \mid A \subseteq F\}$. For any $B \in \bigcup_{q' \leq q} \binom{U}{q'}$, $\chi'(B) = \{F \in \mathcal{F} \mid B \cap F = \emptyset\}$. It is easy to see that $(\mathcal{F}, \chi, \chi')$ is a n - p - q - separating collection. Note that $|\mathcal{F}| = \binom{n}{n-q} \leq n^q$. Since $n \leq \frac{1}{1-x}$, the size of the n - p - q - separating collection is upperbound by the claimed bound. Since we can list all the elements in \mathcal{F} in n^q time, the initialization time is upper bounded by the claimed bound. For any $A \subseteq U$, $|A| = p'$, the cardinality of $\chi(A)$ is is at most $|\mathcal{F}|$ which is upper bounded by $\frac{1}{(1-x)^q}$. Thus the (χ, p') -degree and (χ, p') -query time is bounded by the claimed bound. For any $B \subseteq U$, $|B| = q'$, the cardinality of $\chi'(B)$ is exactly equal to $\binom{n}{q-q'}$, which is upper bounded by $\frac{1}{(1-x)^{q-q'}}$. Thus the (χ', q') -degree and (χ', q') -query time is bounded by the claimed bound.

Case 3: $x, 1 - x > \frac{1}{n}$. The structure of the proof in this case is as follows. We first create a collection using Lemma 7.5. Then we apply Lemma 7.6 and obtain another

construction. From here onwards we keep applying Lemma 7.7 and Lemma 7.6 in phases until we achieve the required bounds on size, degree, query and initialization time.

We first apply Lemma 7.5 and get a construction of n - p - q -separating collections with the following parameters.

- size, $\zeta^1(n, p, q) = \mathcal{O}\left(\frac{1}{x^p(1-x)^q} \cdot (p^2 + q^2 + 1) \log n\right)$,
- initialization time, $\tau_I^1(n, p, q) = \mathcal{O}\left(\binom{2^n}{\zeta(n, p, q)} \cdot \frac{1}{x^p(1-x)^q} \cdot n^{\mathcal{O}(p+q)}\right)$,
- (χ_1, p') -degree for $p' \leq p$, $\Delta_{(\chi_1, p')}^1(n, p, q) = \mathcal{O}\left(\frac{1}{x^{p-p'}} \cdot \frac{(p^2+q^2+1)}{(1-x)^q} \cdot \log n\right)$
- (χ_1, p') -query time $Q_{(\chi_1, p')}^1(n, p, q) = \mathcal{O}\left(\frac{1}{x^p(1-x)^q} \cdot n^{\mathcal{O}(1)}\right) = \mathcal{O}(2^n n^{\mathcal{O}(1)})$
- (χ'_1, q') -degree for $q' \leq q$, $\Delta_{(\chi'_1, q')}^1(n, p, q) = \mathcal{O}\left(\frac{1}{x^p(1-x)^{q-q'}} \cdot (p^2 + q^2 + 1) \cdot \log n\right)$
- (χ'_1, q') -query time, $Q_{(\chi'_1, q')}^1(n, p, q) = \mathcal{O}\left(\frac{1}{x^p(1-x)^q} \cdot n^{\mathcal{O}(1)}\right) = \mathcal{O}(2^n n^{\mathcal{O}(1)})$

We apply Lemma 7.6 to this construction to get a new construction with the following parameters.

- size, $\zeta^2(n, p, q) = \mathcal{O}\left(\frac{1}{x^p(1-x)^q} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n\right)$
- initialization time,

$$\begin{aligned} \tau_I^2(n, p, q) &= \mathcal{O}\left(\tau_I^1((p+q)^2, p, q) + \zeta^1((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot n \log n\right) \\ &= \mathcal{O}\left(\frac{2^{2(p+q)^2}}{x^p(1-x)^q} \cdot (p+q)^{\mathcal{O}(p+q)} + \left(\frac{1}{x^p(1-x)^q} \cdot (p+q)^{\mathcal{O}(1)} \cdot n \log n\right)\right) \\ &= \mathcal{O}\left(\frac{(p+q)^{\mathcal{O}(p+q)}}{x^p(1-x)^q} \left(2^{2(p+q)^2} + n \log n\right)\right) \end{aligned}$$

- (χ_2, p') -degree, $\Delta_{(\chi_2, p')}^2(n, p, q) = \mathcal{O}\left(\frac{1}{x^{p-p'}(1-x)^q} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n\right)$
- (χ_2, p') -query time, $Q_{(\chi_2, p')}^2(n, p, q) = \mathcal{O}\left(\left(2^{(p+q)^2} + \frac{1}{x^{p-p'}(1-x)^q}\right) (p+q)^{\mathcal{O}(1)} \cdot \log n\right)$
- (χ'_2, q') -degree, $\Delta_{(\chi'_2, q')}^2(n, p, q) = \mathcal{O}\left(\frac{1}{x^p(1-x)^{q-q'}} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n\right)$

- (χ_2, q') -query time,

$$Q_{(\chi_2, q')}^2(n, p, q) = \mathcal{O}\left(\left(2^{(p+q)^2} + \frac{1}{x^p(1-x)^{q-q'}}\right) (p+q)^{\mathcal{O}(1)} \cdot \log n\right)$$

We apply Lemma 7.7 to this construction. Recall that in Lemma 7.7 we set $s = \lfloor (\log(p+q))^2 \rfloor$ and $t = \lceil \frac{p+q}{s} \rceil$.

$$\begin{aligned} \zeta^3(n, p, q) &\leq |\mathcal{P}_t^n| \cdot \sum_{(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p} \prod_{i \leq t} \zeta^2(n, p_i, s - p_i) \\ &\leq n^{\mathcal{O}(t)} \cdot |\mathcal{Z}_{s,t}^p| \cdot \max_{(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p} \prod_{i \leq t} \zeta^2(n, p_i, s - p_i) \\ &\leq n^{\mathcal{O}(t)} \cdot (p+q)^{\mathcal{O}(t)} \cdot \frac{1}{x^p(1-x)^{q+s}} \cdot s^{\mathcal{O}(t)} \cdot (\log n)^{\mathcal{O}(t)} \\ &\leq n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot \frac{1}{x^p(1-x)^q} \quad \left(\text{Because } \left(\frac{1}{1-x}\right)^s \leq n^s \leq n^{\mathcal{O}(t)}\right) \end{aligned}$$

$$\begin{aligned} \tau_I^3(n, p, q) &= \mathcal{O}\left(\left(\sum_{\substack{\hat{p} \leq s, p \\ s - \hat{p} \leq q}} \tau_I^2(n, \hat{p}, s - \hat{p})\right) + \zeta^3(n, p, q) \cdot n^{\mathcal{O}(1)}\right) \\ &= \mathcal{O}\left(\left(\sum_{\substack{\hat{p} \leq s, p \\ s - \hat{p} \leq q}} \frac{s^{\mathcal{O}(s)}}{x^{\hat{p}}(1-x)^{s-\hat{p}}} \left(2^{2s^2} + n \log n\right)\right) + \zeta^3(n, p, q) \cdot n^{\mathcal{O}(1)}\right) \\ &= \mathcal{O}\left(\frac{(\log(p+q))^{\mathcal{O}(\log^2(p+q))}}{x^p(1-x)^q} \left(2^{2^{\log^4(p+q)}} + n \log n\right) + \frac{n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})}}{x^p(1-x)^q}\right) \end{aligned}$$

$$\begin{aligned}
\Delta_{(\chi_3, p')}^3(n, p, q) &\leq \Delta_{(\chi_3, p')}^{*3}(n, p, q) \\
&= |\mathcal{P}_t^n| \cdot |\mathcal{Z}_{s,t}^p| \cdot \max_{\substack{(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p \\ p'_1 \leq p_1, \dots, p'_t \leq p_t \\ p'_1 + \dots + p'_t = p'}} \prod_{i \leq t} \Delta_{(\chi, p')}^2(n, p_i, s - p_i) \\
&\leq n^{\mathcal{O}(t)} \cdot (p+q)^{\mathcal{O}(t)} \cdot \frac{1}{x^{p-p'}(1-x)^{q+s}} \cdot s^{\mathcal{O}(t)} \cdot (\log n)^{\mathcal{O}(t)} \\
&\leq n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot \frac{1}{x^{p-p'}(1-x)^q} \quad \left(\text{Because } \left(\frac{1}{1-x} \right)^s \in n^{\mathcal{O}(t)} \right) \\
\Delta_{(\chi'_3, q')}^3(n, p, q) &\leq \Delta_{(\chi'_3, q')}^{*3}(n, p, q) \\
&= |\mathcal{P}_t^n| \cdot |\mathcal{Z}_{s,t}^p| \cdot \max_{\substack{(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p \\ q'_1 \leq s-p_1, \dots, q'_t \leq s-p_t \\ q'_1 + \dots + q'_t = q'}} \prod_{i \leq t} \Delta_{(\chi', q'_i)}^2(n, p_i, s - p_i) \\
&\leq n^{\mathcal{O}(t)} \cdot (p+q)^{\mathcal{O}(t)} \cdot \frac{1}{x^p(1-x)^{q+s-q'}} \cdot s^{\mathcal{O}(t)} \cdot (\log n)^{\mathcal{O}(t)} \\
&\leq n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot \frac{1}{x^p(1-x)^{q-q'}} \quad \left(\text{Because } \left(\frac{1}{1-x} \right)^s \in n^{\mathcal{O}(t)} \right)
\end{aligned}$$

$$\begin{aligned}
Q_{(\chi_3, p')}^3(n, p, q) &\leq \mathcal{O}\left(\Delta_{(\chi_3, p')}^{*3}(n, p, q) \cdot n^{\mathcal{O}(1)} + \right. \\
&\quad \left. |\mathcal{P}_t^n| \cdot |\mathcal{Z}_{s,t}^p| \cdot t \cdot \max_{\substack{\hat{p}' \leq \hat{p} \leq s \\ \hat{p} - \hat{p}' \leq p - p' \\ s - \hat{p} \leq q}} Q_{(\chi_2, \hat{p}')}^2(n, \hat{p}, s - \hat{p}) \right) \\
&\leq \mathcal{O}\left(\Delta_{(\chi_3, p')}^{*3}(n, p, q) \cdot n^{\mathcal{O}(1)} + \right. \\
&\quad \left. n^{\mathcal{O}(t)} \cdot \max_{\substack{\hat{p}' \leq \hat{p} \leq s \\ \hat{p} - \hat{p}' \leq p - p' \\ s - \hat{p} \leq q}} \left(2^{s^2} + \frac{1}{x^{\hat{p}-\hat{p}'}(1-x)^{s-\hat{p}}} \right) s^{\mathcal{O}(1)} \log n \right) \\
&\leq \mathcal{O}\left(\frac{n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})}}{x^{p-p'}(1-x)^q} + n^{\mathcal{O}(t)} \cdot s^{\mathcal{O}(1)} \cdot \log n \left(2^{s^2} + \frac{1}{x^{p-p'}(1-x)^q} \right) \right) \\
&\leq \mathcal{O}\left(\frac{n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})}}{x^{p-p'}(1-x)^q} \right)
\end{aligned}$$

Similar way we can bound $Q_{(\chi'_3, q')}^3$ as,

$$Q_{(\chi'_3, q')}^3(n, p, q) \leq \mathcal{O}\left(\frac{n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})}}{x^p(1-x)^{q-q'}} \right)$$

We apply Lemma 7.6 to this construction to get a new construction with the following parameters.

- size, $\zeta^4(n, p, q) \leq 2^{\mathcal{O}(\frac{p+q}{\log(p+q)})} \cdot \frac{1}{x^p(1-x)^q} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n$,
- initialization time,

$$\begin{aligned} \tau_I^4(n, p, q) &\leq \mathcal{O}\left(\tau_I^3((p+q)^2, p, q) + \zeta^3((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot n \log n\right) \\ &\leq 2^{2 \log^4(p+q)} \cdot \frac{(\log(p+q))^{\mathcal{O}(\log^2(p+q))}}{x^p(1-x)^q} + \\ &\quad \frac{2^{\mathcal{O}(\frac{p+q}{\log(p+q)})}}{x^p(1-x)^q} \cdot (p+q)^{\mathcal{O}(1)} n \log n \end{aligned}$$

- (χ_4, p') -degree,

$$\begin{aligned} \Delta_{(\chi_4, p')}^4(n, p, q) &\leq \Delta_{(\chi_3, p')}^3((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n \\ &\leq \frac{2^{\mathcal{O}(\frac{p+q}{\log(p+q)})}}{x^{p-p'}(1-x)^q} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n \end{aligned}$$

- (χ'_4, q') -degree,

$$\begin{aligned} \Delta_{(\chi'_4, q')}^4(n, p, q) &\leq \Delta_{(\chi'_3, q')}^3((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n \\ &\leq \frac{2^{\mathcal{O}(\frac{p+q}{\log(p+q)})}}{x^p(1-x)^{q-q'}} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n \end{aligned}$$

- (χ_4, p') -query time,

$$\begin{aligned} Q_{(\chi_4, p')}^4(n, p, q) &\leq \mathcal{O}\left(Q_{(\chi_3, p')}^3((p+q)^2, p, q) + \Delta_{(\chi_3, p')}^3((p+q)^2, p, q)\right) \cdot \\ &\quad (p+q)^{\mathcal{O}(1)} \cdot \log n \\ &\leq \frac{2^{\mathcal{O}(\frac{p+q}{\log(p+q)})}}{x^{p-p'}(1-x)^q} \cdot (p+q)^{\mathcal{O}(1)} \log n \end{aligned}$$

- (χ'_4, q') -query time,

$$Q_{(\chi'_4, q')}^4(n, p, q) \leq \frac{2^{\mathcal{O}(\frac{p+q}{\log(p+q)})}}{x^p(1-x)^{q-q'}} \cdot (p+q)^{\mathcal{O}(1)} \log n$$

We apply Lemma 7.7 to this construction by setting $s = \lfloor (\log(p+q))^2 \rfloor$ and $t = \lceil \frac{p+q}{s} \rceil$.

- size,

$$\begin{aligned} \zeta^5(n, p, q) &\leq |\mathcal{P}_t^n| \cdot \sum_{(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p} \prod_{i \leq t} \zeta^4(n, p_i, s - p_i) \\ &\leq n^{\mathcal{O}(t)} \cdot (p+q)^{\mathcal{O}(t)} \cdot s^{\mathcal{O}(t)} \cdot 2^{\mathcal{O}(\frac{st}{\log s})} \cdot (\log n)^{\mathcal{O}(t)} \cdot \frac{1}{x^p(1-x)^{q+s}} \\ &\leq n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \frac{1}{x^p(1-x)^q} \\ &\quad \left(\text{Because } \left(\frac{1}{1-x} \right)^s \in n^{\mathcal{O}(t)} \right) \end{aligned}$$

- initialization time,

$$\begin{aligned}
\tau_I^5(n, p, q) &\leq \mathcal{O} \left(\left(\sum_{\substack{\hat{p} \leq s, p \\ s - \hat{p} \leq q}} \tau_I^4(n, \hat{p}, s - \hat{p}) \right) + \zeta^5(n, p, q) \cdot n^{\mathcal{O}(1)} \right) \\
&\leq \mathcal{O} \left(s \frac{2^{2 \log^4 s} \cdot (\log s)^{\mathcal{O}(\log^2 s)}}{x^p (1-x)^q} + \frac{2^{\mathcal{O}(\frac{s}{\log s})}}{x^p (1-x)^q} \cdot n \log n + \right. \\
&\quad \left. n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot \frac{2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})}}{x^p (1-x)^q} \right) \\
&\leq \mathcal{O} \left(s \frac{2^{2 \log^4 s} \cdot (\log s)^{\mathcal{O}(\log^2 s)}}{x^p (1-x)^q} + n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot \frac{2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})}}{x^p (1-x)^q} \right) \\
&\leq \mathcal{O} \left(\frac{2^{2 \log^4 s} \cdot (s)^{\mathcal{O}(s)}}{x^p (1-x)^q} + n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot \frac{2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})}}{x^p (1-x)^q} \right) \\
&\leq \mathcal{O} \left(\frac{2^{2(2 \log \log(p+q))^4} \cdot (\log(p+q))^{\mathcal{O}((\log(p+q))^2)}}{x^p (1-x)^q} + \right. \\
&\quad \left. n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot \frac{2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})}}{x^p (1-x)^q} \right) \\
&\leq \mathcal{O} \left(n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot \frac{2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})}}{x^p (1-x)^q} \right) \\
&\quad \left(\text{Because } 2^{2(2 \log \log(p+q))^4}, (\log(p+q))^{\mathcal{O}(\log^2(p+q))} \leq 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \right)
\end{aligned}$$

- (χ_5, p') -degree,

$$\begin{aligned}
\Delta_{(\chi_5, p')}^5(n, p, q) &\leq \Delta_{(\chi_5, p')}^{*5}(n, p, q) \\
&= |\mathcal{P}_t^n| \cdot |\mathcal{Z}_{s,t}^p| \cdot \max_{\substack{(p_1, \dots, p_t) \in \mathcal{Z}_{s,t}^p \\ p'_1 \leq p_1, \dots, p'_t \leq p_t \\ p'_1 + \dots + p'_t = p'}} \prod_{i \leq t} \Delta_{(\chi_4, p'_i)}^4(n, p_i, s - p_i) \\
&\leq n^{\mathcal{O}(t)} \cdot (p+q)^{\mathcal{O}(t)} \cdot \frac{2^{\mathcal{O}(\frac{st}{\log s})}}{x^{p-p'} (1-x)^{q+s}} \cdot s^{\mathcal{O}(t)} \cdot (\log n)^{\mathcal{O}(t)} \\
&\leq n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^{p-p'} (1-x)^q} \\
&\quad \left(\text{Because } \left(\frac{1}{1-x} \right)^s \in n^{\mathcal{O}(t)} \right)
\end{aligned}$$

- (χ'_5, q') -degree,

$$\begin{aligned} \Delta_{(\chi'_5, q')}^5(n, p, q) &\leq \Delta_{(\chi'_5, q')}^{*5}(n, p, q) \\ &\leq n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^p(1-x)^{q-q'}} \end{aligned}$$

- (χ_5, p') -query time,

$$\begin{aligned} Q_{(\chi_5, p')}^5(n, p, q) &\leq \mathcal{O}\left(\Delta_{(\chi_5, p')}^{*5}(n, p, q) \cdot n^{\mathcal{O}(1)} + \right. \\ &\quad \left. |\mathcal{P}_t^n| \cdot |\mathcal{Z}_{s,t}^p| \cdot \max_{\substack{\hat{p}' \leq \hat{p} \leq s \\ \hat{p} - \hat{p}' \leq p - p' \\ s - \hat{p} \leq q}} Q_{(\chi_4, \hat{p}')}^4(n, \hat{p}, s - \hat{p})\right) \\ &\leq n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^{p-p'}(1-x)^q} \end{aligned}$$

- (χ'_5, q') -query time,

$$Q_{(\chi'_5, q')}^5(n, p, q) \leq n^{\mathcal{O}(\frac{p+q}{\log^2(p+q)})} \cdot 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^p(1-x)^{q-q'}}$$

We apply Lemma 7.6 to this construction to get a new construction with the following parameters.

- size,

$$\begin{aligned} \zeta(n, p, q) &\leq \zeta^5((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n \\ &\leq 2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^p(1-x)^q} \cdot (p+q)^{\mathcal{O}(1)} \log n \end{aligned}$$

- initialization time,

$$\begin{aligned} \tau_I(n, p, q) &\leq \mathcal{O}\left(\tau_I^5((p+q)^2, p, q) + \zeta^5((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot n \log n\right) \\ &= \mathcal{O}\left(2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x^p(1-x)^q} \cdot (p+q)^{\mathcal{O}(1)} n \log n\right) \end{aligned}$$

- (χ, p') -degree,

$$\begin{aligned}\Delta_{(\chi, p')}(n, p, q) &\leq \Delta_{(\chi_5, p')}^5((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n \\ &\leq \mathcal{O}\left(2^{\mathcal{O}\left(\frac{p+q}{\log \log(p+q)}\right)} \cdot \frac{1}{x^{p-p'}(1-x)^q} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n\right)\end{aligned}$$

- (χ, p') -query time,

$$\begin{aligned}Q_{(\chi, p')}(n, p, q) &\leq \mathcal{O}\left(\left(Q_{(\chi_5, p')}^5((p+q)^2, p, q) + \Delta_{(\chi_5, p')}^5((p+q)^2, p, q)\right) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n\right) \\ &\leq \mathcal{O}\left(2^{\mathcal{O}\left(\frac{p+q}{\log \log(p+q)}\right)} \cdot \frac{1}{x^{p-p'}(1-x)^q} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n\right)\end{aligned}$$

- (χ', q') -degree,

$$\begin{aligned}\Delta_{(\chi', q')}(n, p, q) &= \Delta_{(\chi'_5, q')}^5((p+q)^2, p, q) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n \\ &\leq \mathcal{O}\left(2^{\mathcal{O}\left(\frac{p+q}{\log \log(p+q)}\right)} \cdot \frac{1}{x^p(1-x)^{q-q'}} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n\right)\end{aligned}$$

- (χ', q') -query time,

$$\begin{aligned}Q_{(\chi', q')}(n, p, q) &= \mathcal{O}\left(\left(Q_{(\chi'_5, q')}^5((p+q)^2, p, q) + \Delta_{(\chi'_5, q')}^5((p+q)^2, p, q)\right) \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n\right) \\ &\leq \mathcal{O}\left(2^{\mathcal{O}\left(\frac{p+q}{\log \log(p+q)}\right)} \cdot \frac{1}{x^p(1-x)^{q-q'}} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n\right)\end{aligned}$$

The final construction satisfies all the claimed bounds. This concludes the proof. \square

Now using n - p - q -separating collections we give a fast computation of representative family (Theorem 7.1). Towards that consider the following lemma.

Lemma 7.8. *There is an algorithm that given a p -family \mathcal{A} of sets over a universe U of size n , an integer q , a $0 < x < 1$, and a non-negative weight function $w : \mathcal{A} \rightarrow \mathbb{N}$*

with maximum value at most W , computes in time

$$\mathcal{O}(x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)} \cdot n \log n + |\mathcal{A}| \cdot \log |\mathcal{A}| \cdot \log W + |\mathcal{A}| \cdot (1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n)$$

a subfamily $\hat{\mathcal{A}} \subseteq \mathcal{A}$ such that $|\hat{\mathcal{A}}| \leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n$ and $\hat{\mathcal{A}} \subseteq_{\minrep}^q \mathcal{A}$ ($\hat{\mathcal{A}} \subseteq_{\maxrep}^q \mathcal{A}$).

Proof. The algorithm first checks whether $|\mathcal{A}| \leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n$. If yes then it outputs \mathcal{A} (as $\hat{\mathcal{A}}$) and halts. So we assume that $|\mathcal{A}| > x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n$. The algorithm starts by constructing a generalized n - p - q -separating collection $(\mathcal{F}, \chi, \chi')$ as guaranteed by Lemma 7.4. If $|\mathcal{A}| \leq |\mathcal{F}|$ the algorithm outputs \mathcal{A} and halts. Otherwise it builds the set $\hat{\mathcal{A}}$ as follows. Initially $\hat{\mathcal{A}}$ is equal to \emptyset and all sets in \mathcal{F} are marked as unused. Now we sort the sets in \mathcal{A} in the increasing order of weights, given by $w : \mathcal{A} \rightarrow \mathbb{N}$. The algorithm goes through every $A \in \mathcal{A}$ in the sorted order and queries the separating collection to get the set $\chi(A)$. It then looks for a set $F \in \chi(A)$ that is not yet marked as used. The first time such a set F is found the algorithm marks F as used, inserts A into $\hat{\mathcal{A}}$ and proceeds to the next set in \mathcal{A} . If no such set F is found the algorithm proceeds to the next set in \mathcal{A} without inserting A into $\hat{\mathcal{A}}$.

The size of $\hat{\mathcal{A}}$ is upper bounded by $|\mathcal{F}| \leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n$ since every time a set is added to $\hat{\mathcal{A}}$ an unused set in \mathcal{F} is marked as used. For the running time analysis, the initialization of (\mathcal{F}, χ) takes time $x^{-p}(1-x)^{-q} \cdot (p+q)^{\mathcal{O}(1)} \cdot 2^{o(p+q)} \cdot n \log n$. Sorting \mathcal{A} takes $\mathcal{O}(|\mathcal{A}| \cdot \log |\mathcal{A}| \cdot \log W)$ time. For each element $A \in \mathcal{A}$ the algorithm first queries $\chi(A)$, using time $(1-x)^{-q} \cdot 2^{o(p+q)} \cdot (p+q)^{\mathcal{O}(1)} \cdot \log n$. Then it goes through all sets in $\chi(A)$ and checks whether they have already been marked as used, taking time $(1-x)^{-q} \cdot (p+q)^{\mathcal{O}(1)} \cdot 2^{o(p+q)} \cdot \log n$. Thus in total, the running time for these steps is bounded by $\mathcal{O}(|\mathcal{A}| \cdot (1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n + |\mathcal{A}| \cdot \log |\mathcal{A}| \cdot \log W)$. Adding the initialization time to this gives the claimed running time.

Finally we need to argue that $\hat{\mathcal{A}} \subseteq_{\minrep}^q \mathcal{A}$. Consider any set $A \in \mathcal{A}$ and B such

that $|B| = q$ and $A \cap B = \emptyset$. If $A \in \hat{\mathcal{A}}$ we are done, so assume that $A \notin \hat{\mathcal{A}}$. Since $(\mathcal{F}, \chi, \chi')$ is a n - p - q -separating collection, we have that there exists $F \in \chi(A) \cap \chi'(B)$, i.e., $A \subseteq F$ and $F \cap B = \emptyset$. Since $A \notin \hat{\mathcal{A}}$ we know that F was marked as used when A was considered by the algorithm. When the algorithm marked F as used it also inserted a set A' into $\hat{\mathcal{A}}$, with the property that $F \in \chi(A')$. Thus $A' \subseteq F$ and hence $A' \cap B = \emptyset$. Furthermore, A' was considered before A and thus $w(A') \leq w(A)$. But $A' \in \hat{\mathcal{A}}$, completing the proof. \square

Next we prove a “faster version of Lemma 7.8”, that speeds up the running time to compute the representative families.

Lemma 7.9. *There is an algorithm that given a p -family \mathcal{A} of sets over a universe U of size n , an integer q , a $0 < x < 1$, and a non-negative weight function $w : \mathcal{A} \rightarrow \mathbb{N}$ with maximum value at most W , computes in time*

$$\mathcal{O}((p+q)^{\mathcal{O}(1)} n \log n + |\mathcal{A}| \cdot \log |\mathcal{A}| \cdot \log W + |\mathcal{A}| \cdot (1-x)^{-q} \cdot 2^{\mathcal{O}(p+q)} \cdot \log n)$$

a subfamily $\hat{\mathcal{A}} \subseteq \mathcal{A}$ such that $|\hat{\mathcal{A}}| \leq x^{-p}(1-x)^{-q} \cdot 2^{\mathcal{O}(p+q)} \cdot \log n$ and $\hat{\mathcal{A}} \subseteq_{\minrep}^q \mathcal{A}$ ($\hat{\mathcal{A}} \subseteq_{\maxrep}^q \mathcal{A}$).

Proof. The algorithm first checks whether $|\mathcal{A}| \leq x^{-p}(1-x)^{-q} \cdot 2^{\mathcal{O}(p+q)} \cdot \log n$. If yes then it outputs \mathcal{A} (as $\hat{\mathcal{A}}$) and halts. So we assume that $|\mathcal{A}| > x^{-p}(1-x)^{-q} \cdot 2^{\mathcal{O}(p+q)} \cdot \log n$.

We start by constructing a $(p+q)$ -perfect family f_1, \dots, f_t of hash functions from U to $[(p+q)^2]$ with $t = \mathcal{O}((p+q)^{\mathcal{O}(1)} \cdot \log n)$ in time $\mathcal{O}(k^{\mathcal{O}(1)} n \log n)$ using Theorem 5.2. Now we sort the sets in \mathcal{A} in the increasing order of weights, given by $w : \mathcal{A} \rightarrow \mathbb{N}$. For every f_j , $1 \leq j \leq t$, we construct a family $\hat{\mathcal{A}}_j$ as follows. The algorithm starts by constructing a generalized $[(p+q)^2]$ - p - q -separating collection $(\mathcal{F}_j, \chi_j, \chi'_j)$ as guaranteed by Lemma 7.4. It builds the set $\hat{\mathcal{A}}_j$ as follows. Initially $\hat{\mathcal{A}}_j$ is equal to \emptyset and all sets in \mathcal{F} are marked as unused. The algorithm goes through every $A \in \mathcal{A}$ in the sorted order and does as follows.

- It first check whether every element in A gets mapped to distinct integers by f_j . That is, $|\{f_j(a) \mid a \in A\}| = |A|$. If $|\{f_j(a) \mid a \in A\}| < |A|$ then the algorithm proceeds to the next set in \mathcal{A} without inserting A into $\hat{\mathcal{A}}$. Else, we move to the next step.
- It queries the separating collection to get the set $\chi(A)$. It looks for a set $F \in \chi_j(A)$ that is not yet marked as used. The first time such a set F is found the algorithm marks F as used, inserts A into $\hat{\mathcal{A}}_j$ and proceeds to the next set in \mathcal{A} . If no such set F is found the algorithm proceeds to the next set in \mathcal{A} without inserting A into $\hat{\mathcal{A}}_j$.

Finally, we return $\hat{\mathcal{A}} = \bigcup_{j=1}^t \hat{\mathcal{A}}_j$.

The size of $\hat{\mathcal{A}}_j$ is upper bounded by $|\mathcal{F}| \leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log(p+q)$ since every time a set is added to $\hat{\mathcal{A}}$ an unused set in \mathcal{F} is marked as used. Thus, the size of $\hat{\mathcal{A}}$ is upper bounded by $|\mathcal{F}| \leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log(p+q) \cdot (p+q)^{O(1)} \cdot \log n \leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n$. The running time analysis follows similar to the one given in Lemma 7.8.

Finally we need to argue that $\hat{\mathcal{A}} \subseteq_{\minrep}^q \mathcal{A}$. Consider any set $A \in \mathcal{A}$ and B such that $|B| = q$ and $A \cap B = \emptyset$. If $A \in \hat{\mathcal{A}}$ we are done, so assume that $A \notin \hat{\mathcal{A}}$. By the properties of $(p+q)$ -perfect family f_1, \dots, f_t of hash functions from U to $[(p+q)^2]$, there exists an integer $j \in \{1, \dots, t\}$ such that f_j is injective on $A \cup B$. We focus now on the construction of $\hat{\mathcal{A}}_j$. Since $(\mathcal{F}_j, \chi_j, \chi'_j)$ is a $[(p+q)^2]$ - p - q -separating collection, we have that there exists $F \in \chi_j(A) \cap \chi'_j(B)$, i.e, $A \subseteq F$ and $F \cap B = \emptyset$. Since $A \notin \hat{\mathcal{A}}_j$ (as $A \notin \hat{\mathcal{A}}$) we know that F was marked as used when A was considered by the algorithm. When the algorithm marked F as used it also inserted a set A' into $\hat{\mathcal{A}}$, with the property that $F \in \chi(A')$. Thus $A' \subseteq F$ and hence $A' \cap B = \emptyset$. Furthermore, A' was considered before A and thus $w(A') \leq w(A)$. But $A' \in \hat{\mathcal{A}}_j \subseteq \hat{\mathcal{A}}$, completing the proof. \square

While applying Lemma 7.9 we can reduce the universe size to at most $|\mathcal{A}|p+q$. The next lemma formalizes this.

Lemma 7.10. *There is an algorithm that given a p -family \mathcal{A} of sets over a universe U of size n , an integer q , $0 < x < 1$ and a non-negative weight function $w : \mathcal{A} \rightarrow \mathbb{N}$ with maximum value at most W , computes in time*

$$\mathcal{O}(|\mathcal{A}| \cdot \log |\mathcal{A}| \cdot \log W + |\mathcal{A}| \cdot (1 - x)^{-q} \cdot 2^{o(p+q)} \cdot \log n)$$

a subfamily $\widehat{\mathcal{A}} \subseteq \mathcal{A}$ such that $|\widehat{\mathcal{A}}| \leq x^{-p}(1 - x)^{-q} \cdot 2^{o(p+q)} \cdot \log |\mathcal{A}|$ and $\widehat{\mathcal{A}} \subseteq_{\minrep}^q \mathcal{A}$ ($\widehat{\mathcal{A}} \subseteq_{\maxrep}^q \mathcal{A}$).

Proof. We first construct a new universe U' as follows. If $n \leq |\mathcal{A}|p + q$, then we set $U' = U$, otherwise U' will consist of elements from U , which are part of any set in \mathcal{A} and q new elements. The universe U' can be constructed in $\mathcal{O}(|\mathcal{A}|p + q)$ time. Also note that $|U'| \leq |\mathcal{A}|p + q$ and $|U'| \leq n$. Now we claim that a q -representative family $\widehat{\mathcal{A}}$ of \mathcal{A} with respect to the universe U' is also the required representative family over U . Suppose $X \in \mathcal{A}$ and $Y \subseteq U$, $|Y| \leq q$ such that $X \cap Y = \emptyset$. Let $Y' = Y \setminus U'$ and let Y'' be an arbitrary subset of size $|Y'|$ of $U' \setminus U$. Let $Z = (Y \setminus Y') \cup Y''$. It is easy to see that $|Z| = |Y|$ and $X \cap Z = \emptyset$. By the definition of q -representative family, there exists $\widehat{X} \in \widehat{\mathcal{A}}$ such that $\widehat{X} \cap Z = \emptyset$. Since $Y' \cap \widehat{X} = \emptyset$, we have that $\widehat{X} \cap Y = \emptyset$.

Thus we apply Lemma 7.9 to compute q -representative family $\widehat{\mathcal{A}}$ of \mathcal{A} with respect to the universe U' and output it as the desired family. The claimed running time as well as the size bound on the output representative family follow by substituting the upper bound on $|U'|$ in the bounds coming from Lemma 7.9. \square

Finally, we give our main theorem.

Theorem 7.3. *There is an algorithm that given a p -family \mathcal{A} of sets over a universe U of size n , an integer q , $0 < x < 1$ and a non-negative weight function $w : \mathcal{A} \rightarrow \mathbb{N}$ with maximum value at most W , computes in time*

$$\mathcal{O}(|\mathcal{A}| \cdot \log |\mathcal{A}| \cdot \log W + |\mathcal{A}| \cdot (1 - x)^{-q} \cdot 2^{o(p+q)} \cdot \log n)$$

a subfamily $\widehat{\mathcal{A}} \subseteq \mathcal{A}$ such that $|\widehat{\mathcal{A}}| \leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)}$ and $\widehat{\mathcal{A}} \subseteq_{\minrep}^q \mathcal{A}$ ($\widehat{\mathcal{A}} \subseteq_{\maxrep}^q \mathcal{A}$).

Proof. Let $\mathcal{A} = \mathcal{A}_1$. We compute a sequence of representative families

$$\mathcal{A}_2 \subseteq_{\minrep}^q \mathcal{A}_1, \dots, \mathcal{A}_m \subseteq_{\minrep}^q \mathcal{A}_{m-1}$$

using Lemma 7.10, such that m is the least integer with the property that $|\mathcal{A}_m| \geq |\mathcal{A}_{m-1}|/2$. In other words, for all $i < m$ we have that $|\mathcal{A}_i| \leq |\mathcal{A}_{i-1}|/2$ and $|\mathcal{A}_m| \geq |\mathcal{A}_{m-1}|/2$. We output \mathcal{A}_m as the q -representative family for \mathcal{A} . The correctness of this following from Lemma 7.1. By Lemma 7.10,

$$\begin{aligned} |\mathcal{A}_m| &\leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log |\mathcal{A}_{m-1}| \\ &\leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log 2|\mathcal{A}_m| \end{aligned}$$

Thus, $\frac{|\mathcal{A}_m|}{\log |\mathcal{A}_m|} \leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)}$.

We know that for some number a and b , if $a \leq b$ then $a \log^2 a \leq b \log^2 b$. Applying this identity we get the following.

$$\frac{|\mathcal{A}_m|}{\log |\mathcal{A}_m|} \log^2 \left(\frac{|\mathcal{A}_m|}{\log |\mathcal{A}_m|} \right) \leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)}$$

The above inequality implies that

$$|\mathcal{A}_m| \leq \frac{|\mathcal{A}_m|}{\log |\mathcal{A}_m|} \log^2 \left(\frac{|\mathcal{A}_m|}{\log |\mathcal{A}_m|} \right) \leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)}$$

and thus $|\mathcal{A}_m| \leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)}$. By Lemma 7.10, the total running time T

to compute \mathcal{A}_m is,

$$\begin{aligned}
T &= \sum_{i=1}^{m-1} |\mathcal{A}_i| \cdot \log |\mathcal{A}_i| \cdot \log W + |\mathcal{A}_i| \cdot (1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n \\
&= \sum_{i=1}^{m-1} \mathcal{O}\left(\frac{|\mathcal{A}|}{2^{i-1}} \cdot \log |\mathcal{A}| \cdot \log W + \frac{|\mathcal{A}|}{2^{i-1}} \cdot (1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n\right) \\
&\hspace{25em} (\text{since } |\mathcal{A}_i| \leq \frac{|\mathcal{A}|}{2^{i-1}}) \\
&= \mathcal{O}(|\mathcal{A}| \cdot \log |\mathcal{A}| \cdot \log W + |\mathcal{A}| \cdot (1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n)
\end{aligned}$$

This concludes the proof. □

The size of the output representative family in Theorem 7.3 is minimized when $x = \frac{p}{p+q}$. By substituting $x = \frac{p}{p+q}$ in Theorem 7.3 we get the following corollary.

Corollary 7.1. *There is an algorithm that given a p -family \mathcal{A} of sets over a universe U of size n , an integer q , and a non-negative weight function $w : \mathcal{A} \rightarrow \mathbb{N}$ with maximum value at most W , computes in time*

$$\mathcal{O}\left(|\mathcal{A}| \cdot \log |\mathcal{A}| \cdot \log W + |\mathcal{A}| \cdot \left(\frac{p+q}{q}\right)^q \cdot 2^{o(p+q)} \cdot \log n\right)$$

a subfamily $\widehat{\mathcal{A}} \subseteq \mathcal{A}$ such that $|\widehat{\mathcal{A}}| \leq \binom{p+q}{p} \cdot 2^{o(p+q)}$ and $\widehat{\mathcal{A}} \subseteq_{\minrep}^q \mathcal{A}$ ($\widehat{\mathcal{A}} \subseteq_{\maxrep}^q \mathcal{A}$).

Chapter 8

Long Directed Cycle

In the LONG DIRECTED CYCLE problem we are interested in finding a cycle of length at least k in a directed graph. The problem LONG DIRECTED CYCLE is formally defined as follows.

LONG DIRECTED CYCLE

Parameter: k

Input: A n -vertex and m -arc directed graph D and a positive integer k .

Question: Does there exist a directed cycle of length at least k in D ?

In this chapter we prove that LONG DIRECTED CYCLE can be solved in parameterized single exponential time. While at the first glance the problem is similar to the problem of finding a cycle or a path of length exactly k , it is more tricky. The reason is that the problem of finding a cycle of length $\geq k$ may entail finding a much longer, potentially even a Hamiltonian cycle. This is why color-coding, and other techniques applicable to k -PATH do not seem to work here. Even for undirected graphs color-coding alone is not sufficient, and one needs an additional clever trick to make it work. The first fixed-parameter tractable algorithm for LONG DIRECTED CYCLE is due to Gabow and Nie [61], who gave algorithms with expected running time $k^{2k}2^{\mathcal{O}(k)}nm$ and worst-case times $\mathcal{O}(k^{2k}2^{\mathcal{O}(k)}nm \log n)$ or $\mathcal{O}(k^{3k}nm)$. These running times allow them to find a directed cycle of length at least $\log n / \log \log n$ in expected polynomial time, if it exists. Let us note, that our result implies that one can find in polynomial time a directed cycle of length at least $\log n$ if there is such a cycle. On the other hand, Björklund et al. [18] have shown that assuming Exponential Time Hypothesis (ETH) of Impagliazzo et al. [71], there is no polynomial time algorithm that finds a directed cycle of length $\Omega(f(n) \log n)$, for any nondecreasing, unbounded, polynomial time computable function f that tends to infinity. Thus, our work closes the gap between the upper and lower bounds for this problem.

In Section 8.1, we give an algorithm running in time $\mathcal{O}(8^{k+o(k)}mn^2)$ for LONG DIRECTED CYCLE. In Section 8.2, we give a faster algorithm for the problem which runs in time $\mathcal{O}(6.75^{k+o(k)}mn^2)$.

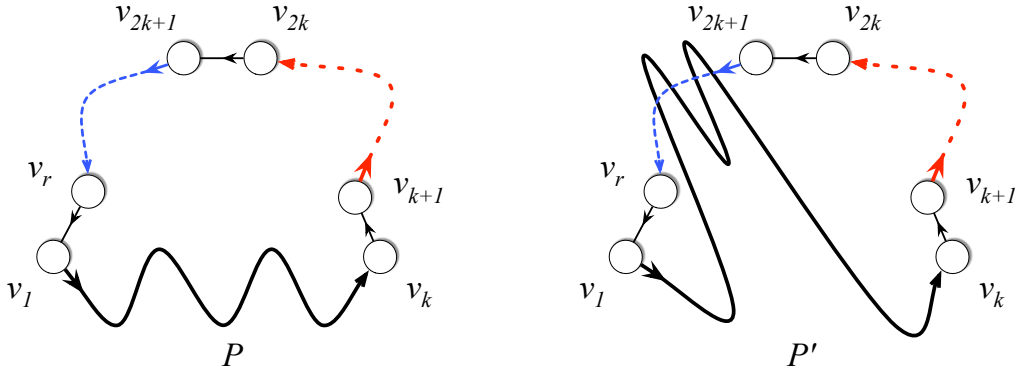


Figure 8.1: Illustration to the proof of Lemma 8.1.

8.1 Algorithm for Long Directed Cycle

First we define some notations. We use $\mathcal{T}_{um}(t, p, q)$ to denote the time required to compute a family $\widehat{\mathcal{S}} \subseteq_{rep}^q \mathcal{S}$ of size $\binom{p+q}{p} \cdot 2^{o(p+q)}$, where \mathcal{S} is a p -family of size t over a universe of size n . By Corollary 7.1, $\mathcal{T}_{um}(t, p, q) = \mathcal{O}(t \cdot (\frac{p+q}{q})^q \cdot \log n)$. For a pair of vertices $u, v \in V(D)$, we define

$$\mathcal{P}_{uv}^i = \left\{ X \mid X \subseteq V(D), u, v \in X, |X| = i, \text{ and there is a directed } uv\text{-path in } D \right. \\ \left. \text{of length } i - 1 \text{ with all the vertices belonging to } X. \right\}$$

Now we start with a structural lemma providing the key insight to our algorithm.

Lemma 8.1. *Let D be a directed graph. Then D has a directed cycle of length at least k if and only if there exists a pair of vertices $u, v \in V(D)$ and $X \in \widehat{\mathcal{P}}_{uv}^k \subseteq_{rep}^k \mathcal{P}_{uv}^k$ such that D has a directed cycle C and in this cycle vertices of X induce a directed path (that is, vertices of X form a consecutive segment in C).*

Proof. The reverse direction of the proof is straightforward—if cycle C contains a path of length k , the length of C is at least k . We proceed with the proof of the forward direction. Let $C^* = v_1 v_2 \cdots v_r v_1$ be a smallest directed cycle in D of length at least k . That is, $r \geq k$ and there is no directed cycle of length r' where $k \leq r' < r$. We consider two cases.

Case A: $r \leq 2k$. If $r \leq 2k$, then we take $u = v_1$ and $v = v_k$. We define paths $P = v_1v_2 \cdots v_k$ and $Q = v_{k+1} \cdots v_r$. Because $|Q| \leq k$, by the definition of $\widehat{\mathcal{P}}_{uv}^k \subseteq_{rep}^k \mathcal{P}_{uv}^k$, there exists a directed uv -path P' such that $X = V(P') \in \widehat{\mathcal{P}}_{uv}^k$ and $X \cap Q = \emptyset$. By replacing P with P' in C^* we obtain a directed cycle C of length at least k containing P' as a subpath.

Case B: $r \geq 2k + 1$. In this case we set $u = v_1$, $v = v_k$, and split C^* into three paths $P = v_1 \cdots v_k$, $Q = v_{k+1} \cdots v_{2k}$, and $R = v_{2k+1} \cdots v_r$. Since $|Q| = k$ and $\widehat{\mathcal{P}}_{uv}^k \subseteq_{rep}^k \mathcal{P}_{uv}^k$, it follows that there exists an uv -path P' such that $X = V(P') \in \widehat{\mathcal{P}}_{uv}^k$ and $X \cap Q = \emptyset$. However, P' is not necessarily disjoint with R and by replacing P with P' in C^* we can obtain a *closed walk* C' containing P' as a subpath. See Fig. 8.1 for an illustration.

If $X \cap R = \emptyset$, then C' is a simple cycle and we take C' as the desired C . We claim that this is the only possibility. Let us assume targeting towards a contradiction that $X \cap R \neq \emptyset$. We want to show that in this case there is a cycle of length at least k but shorter than C^* , contradicting the choice of C^* . Let v_α be the last vertex in $X \cap R$ when we walk from v_1 to v_k along P' . Let $P'[v_\alpha, v_k]$ be the subpath of P' starting at v_α and ending at v_k . If $v_\alpha = v_{2k+1}$, we set $R' = \emptyset$. Otherwise we put $R' = R[v_{2k+1}, v_{\alpha-1}]$ to be the subpath of R starting at v_{2k+1} and ending at $v_{\alpha-1}$. Observe that since the arc $v_{\alpha-1}v_\alpha$ is present in D (in fact it is an arc of the cycle C^*), we have that $\overline{C} = P'[v_\alpha, v_k]QR'$ is a simple cycle in D . Clearly, $|\overline{C}| \geq |Q| \geq k$. Furthermore, since v_1 is not present in $P'[v_\alpha, v_k]$ we have that $|P'[v_\alpha, v_k]| < |P'| = |P|$. Similarly since v_α is not present in R' , we have that $|R'| < |R|$. Thus we have

$$k \leq |\overline{C}| = |P'[v_\alpha, v_k]| + |Q| + |R'| < |P| + |Q| + |R| = |C^*|.$$

This implies that \overline{C} is a directed simple cycle of length at least k and strictly smaller than r . This is a contradiction. Hence by replacing P with P' in C^* we obtain a

directed cycle C containing P' as a subpath. This concludes the proof. \square

Next lemma provides an efficient computation of family $\widehat{\mathcal{P}}_{uv}^k \subseteq_{rep}^k \mathcal{P}_{uv}^k$. The next lemma is provided to give a simple exposition of representative families based dynamic programming algorithm.

Lemma 8.2. *Let D be a directed/undirected graph with n vertices and m edges, and $u \in V(D)$. Let ℓ be a positive integer. Then for every $p \leq \ell$ and $v \in V(D) \setminus \{u\}$, a family $\widehat{\mathcal{P}}_{uv}^p \subseteq_{rep}^{\ell-p} \mathcal{P}_{uv}^p$ of size at most*

$$\binom{\ell}{p} \cdot 2^{o(\ell)}$$

can be found in time

$$\mathcal{O} \left(2^{o(\ell)} m \log n \max_{i \in [p]} \left\{ \binom{\ell}{i-1} \left(\frac{\ell}{\ell-i} \right)^{\ell-i} \right\} \right).$$

Furthermore, within the same running time every set in $\widehat{\mathcal{P}}_{uv}^p$ can be ordered in a way that it corresponds to a directed (undirected) path in D .

Proof. We prove the lemma only for digraphs. The proof for undirected graphs is analogous and we only point out the differences with the proof for the directed case. We describe a dynamic programming based algorithm. Let $V(D) = \{u, v_1, \dots, v_{n-1}\}$ and \mathcal{D} be a $(p-1) \times (n-1)$ matrix where the rows are indexed from integers in $\{2, \dots, p\}$ and the columns are indexed from vertices in $\{v_1, \dots, v_{n-1}\}$. The entry $\mathcal{D}[i, v]$ will store the family $\widehat{\mathcal{P}}_{uv}^i \subseteq_{rep}^{\ell-i} \mathcal{P}_{uv}^i$. We fill the entries in the matrix \mathcal{D} in the increasing order of rows. For $i = 2$, $\mathcal{D}[2, v] = \{\{u, v\}\}$ if $uv \in A(D)$ (for an undirected graph we check whether u and v are adjacent). Assume that we have filled all the entries until the row i . Let

$$\mathcal{N}_{uv}^{i+1} = \bigcup_{w \in N^-(v)} \widehat{\mathcal{P}}_{uw}^i \bullet \{v\}.$$

For undirected graphs we use the following definition

$$\mathcal{N}_{uv}^{i+1} = \bigcup_{w \in N(v)} \widehat{\mathcal{P}}_{uw}^i \bullet \{v\}.$$

Claim 8.1. $\mathcal{N}_{uv}^{i+1} \subseteq_{rep}^{\ell-(i+1)} \mathcal{P}_{uv}^{i+1}$.

Proof. Let $S \in \mathcal{P}_{uv}^{i+1}$ and Y be a set of size $\ell - (i + 1)$ (which is essentially an independent set of $U_{n,\ell}$) such that $S \cap Y = \emptyset$. We will show that there exists a set $S' \in \mathcal{N}_{uv}^{i+1}$ such that $S' \cap Y = \emptyset$. This will imply the desired result. Since $S \in \mathcal{P}_{uv}^{i+1}$ there exists a directed path $P = ua_1 \cdots a_{i-1}v$ in D such that $S = \{u, a_1, \dots, a_{i-1}, v\}$ and $a_{i-1} \in N^-(v)$. The existence of path $P[u, a_{i-1}]$, the subpath of P between u and a_{i-1} , implies that $X^* = S \setminus \{v\} \in \mathcal{P}_{ua_{i-1}}^i$. Take $Y^* = Y \cup \{v\}$. Observe that $X^* \cap Y^* = \emptyset$ and $|Y^*| = \ell - i$. Since $\widehat{\mathcal{P}}_{ua_{i-1}}^i \subseteq_{rep}^{\ell-i} \mathcal{P}_{ua_{i-1}}^i$ there exists a set $\widehat{X}^* \in \widehat{\mathcal{P}}_{ua_{i-1}}^i$ such that $\widehat{X}^* \cap Y^* = \emptyset$. However, since $a_{i-1} \in N^-(v)$ and $\widehat{X}^* \cap \{v\} = \emptyset$ (as $\widehat{X}^* \cap Y^* = \emptyset$), we have $\widehat{X}^* \bullet \{v\} = \widehat{X}^* \cup \{v\}$ and $\widehat{X}^* \cup \{v\} \in \mathcal{N}_v^{i+1}$. Taking $S' = \widehat{X}^* \cup \{v\}$ suffices for our purpose. This completes the proof of the lemma. \square

We fill the entry for $\mathcal{D}[i + 1, v]$ as follows. Observe that

$$\mathcal{N}_{uv}^{i+1} = \bigcup_{w \in N^-(v)} \mathcal{D}[i, w] \bullet \{v\}.$$

We already have computed the family corresponding to $\mathcal{D}[i, w]$ for $w \in N^-(v)$. By Corollary 7.1, $|\widehat{\mathcal{P}}_{uw}^i| \leq \binom{\ell}{i} 2^{o(\ell)}$ and thus $|\mathcal{N}_{uv}^{i+1}| \leq d^-(v) \binom{\ell}{i} 2^{o(\ell)}$. Furthermore, we can compute \mathcal{N}_{uv}^{i+1} in time $\mathcal{O}(d^-(v) \binom{\ell}{i} 2^{o(\ell)})$. Now using Corollary 7.1, we compute $\widehat{\mathcal{N}}_{uv}^{i+1} \subseteq_{rep}^{\ell-i-1} \mathcal{N}_{uv}^{i+1}$ in time $\mathcal{T}_{um}(t, i + 1, \ell - i - 1)$, where $t = d(v) \binom{\ell}{i} 2^{o(\ell)}$. By Claim 8.1, we know that $\mathcal{N}_{uv}^{i+1} \subseteq_{rep}^{\ell-i-1} \mathcal{P}_{uv}^{i+1}$. Thus Lemma 7.1 implies that $\widehat{\mathcal{N}}_{uv}^{i+1} = \widehat{\mathcal{P}}_{uv}^{i+1} \subseteq_{rep}^{\ell-i-1} \mathcal{P}_{uv}^{i+1}$. We assign this family to $\mathcal{D}[i + 1, v]$. This completes the description and the correctness of the algorithm. We give ordering to the vertices of the sets in $\widehat{\mathcal{P}}_{uv}^p$ in the following way so that it corresponds to a directed (undirected) path in D . We keep the sets in the order in which they are built using the \bullet operation.

That is, we can view these sets as strings and \bullet operation as concatenation. Then every ordered set in our family represents a path in the graph. The running time of the algorithm is bounded by

$$\begin{aligned}
& \mathcal{O} \left(\sum_{i=2}^p \sum_{j=1}^{n-1} \mathcal{T}_{um} \left(d^-(v_j) \binom{\ell}{i-1} 2^{o(\ell)}, i, \ell-i \right) \right) \\
&= \mathcal{O} \left(\sum_{i=2}^p \sum_{j=1}^{n-1} d^-(v_j) \binom{\ell}{i-1} \left(\frac{\ell}{\ell-i} \right)^{\ell-i} 2^{o(\ell)} \log n \right) \\
&= \mathcal{O} \left(2^{o(\ell)} \log n \sum_{i=2}^p \sum_{j=1}^{n-1} d^-(v_j) \binom{\ell}{i-1} \left(\frac{\ell}{\ell-i} \right)^{\ell-i} \right) \\
&= \mathcal{O} \left(2^{o(\ell)} m \log n \max_{i \in [p]} \left\{ \binom{\ell}{i-1} \left(\frac{\ell}{\ell-i} \right)^{\ell-i} \right\} \right)
\end{aligned}$$

This completes the proof □

Finally, we are ready to state the main result of this section.

Theorem 8.1. LONG DIRECTED CYCLE *can be solved in time* $\mathcal{O}(8^{k+o(k)} mn^2)$.

Proof. Let D be a directed graph. We solve the problem by applying the structural characterization proved in Lemma 8.1. By Lemma 8.1, D has a directed cycle of length at least k if and only if there exists a pair of vertices $u, v \in V(D)$ and a path P' with $V(P') \in \widehat{\mathcal{P}}_{uv}^k \subseteq_{rep}^k \mathcal{P}_{uv}^k$ such that D has a directed cycle C containing P' as a subpath.

We first compute $\widehat{\mathcal{P}}_{uv}^k \subseteq_{rep}^k \mathcal{P}_{uv}^k$ for all $u, v \in V(D)$. For that we apply Lemma 8.2 for each vertex $u \in V(D)$ with $\ell = 2k$ and $p = k$. Thus, we can compute $\widehat{\mathcal{P}}_{uv}^k \subseteq_{rep}^k \mathcal{P}_{uv}^k$ for all $u, v \in V(D)$ in time $\mathcal{O}(8^{k+o(k)} mn \log n)$. Moreover, for every $X \in \widehat{\mathcal{P}}_{uv}^k$ we also compute a directed uv -path P_X using vertices of X . Let

$$\mathcal{Q} = \bigcup_{u, v \in V(D)} \widehat{\mathcal{P}}_{uv}^k.$$

Now for every set $X \in \mathcal{Q}$ and the corresponding uv -path P_X with endpoint, we check if there is a uv -path in D avoiding all vertices of X but u and v . This check can be

done by a standard graph traversal algorithm like BFS/DFS in time $\mathcal{O}(m + n)$. If we succeed in finding a path for at least one $X \in \mathcal{Q}$, we answer YES and return the corresponding directed cycle obtained by merging P_X and another path. Otherwise, if we did not succeed to find such a path for any of the sets $X \in \mathcal{Q}$, this means that there is no directed cycle of length at least k in D . The correctness of the algorithm follows from Lemma 8.1. By Corollary 7.1, the size of \mathcal{Q} is upper bounded by $n^2 \binom{2k}{k} 2^{o(k)} \leq n^2 4^{k+o(k)}$. Thus the overall running time of the algorithm is upper bounded by

$$\mathcal{O}(8^{k+o(k)} m n \log n + 4^{k+o(k)} (n^2 m + n^3)).$$

This concludes the proof. \square

8.2 Faster Algorithm for Long Directed Cycle

In this section we design a faster algorithm for LONG DIRECTED CYCLE. In Section 8.1 we have seen an algorithm for LONG DIRECTED CYCLE where the running time mainly depend on the computation of representative families $\widehat{\mathcal{P}}_{uv}^p \subseteq_{rep}^q \mathcal{P}_{uv}^p$ for $2 \leq p \leq k$ and $q = 2k - p$. We used Theorem 7.8 with $x = \frac{p}{p+q}$ (i.e., Corollary 7.1) to compute representative families. The choice $x = \frac{p}{p+q}$ minimizes the size of representative family. But in fact, we can choose x that minimizes the running time instead.

Now we find out the choice of x which minimizes the computation of $\widehat{\mathcal{P}}_{uv}^p \subseteq_{rep}^q \mathcal{P}_{uv}^p$ for $2 \leq p \leq k$ and $q = 2k - p$. Let $s_{p,q}$ denote the size of $\widehat{\mathcal{P}}_{uv}^p$. We know that the computation of $\widehat{\mathcal{P}}_{uv}^p \subseteq_{rep}^q \mathcal{N}_{uv}^p \subseteq_{rep}^q \mathcal{P}_{uv}^p$ depends on $|\mathcal{N}_{uv}^p|$, which depends on the size of the representative families $\widehat{\mathcal{P}}_{uv}^{p-1}$. That is $|\mathcal{N}_{uv}^p| \leq s_{p-1,q+1} \cdot n$. Thus the value of $s_{p-1,q+1}$ and $s_{p,q}$ are “almost equal” and we denote it by $s_{p-1,q+1} \approx s_{p,q}$. By Theorem 7.3, the running time to compute $\widehat{\mathcal{P}}_{uv}^p \subseteq_{rep}^q \mathcal{N}_{uv}^p \subseteq_{rep}^q \mathcal{P}_{uv}^p$ is,

$$\begin{aligned} & \mathcal{O}(|\mathcal{N}_{uv}^p| \cdot (1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n) \\ &= \mathcal{O}(s_{p,q} \cdot (1-x)^{-q} \cdot 2^{o(p+q)} \cdot n \log n) \\ &= \mathcal{O}(x^{-p} \cdot (1-x)^{-2q} \cdot 2^{o(p+q)} \cdot n \log n) \end{aligned}$$

To minimize the above running time it is enough to minimize the function $f(x) = x^{-p} \cdot (1-x)^{-2q}$. Using methods from calculus we know that the value x^* of x for which $f'(x^*) = 0$ corresponds to a minimum value of the function $f(x)$ if $f''(x^*) > 0$.

The derivative of $f(x)$ is, $f'(x) = -px^{-p-1}(1-x)^{-2q} + 2q \cdot x^{-p}(1-x)^{-2q-1}$. Now consider the value of x for which $f'(x) = 0$.

$$\begin{aligned} -px^{-p-1}(1-x)^{-2q} + 2q \cdot x^{-p}(1-x)^{-2q-1} &= 0 \\ -p(1-x) + 2q \cdot x &= 0 \\ x &= \frac{p}{p+2q} \end{aligned}$$

Set $x^* = \frac{p}{p+2q}$. To prove $f(x)$ is minimized at x^* , it is enough to show that $f''(x^*) > 0$.

$$\begin{aligned} f'(x) &= -px^{-p-1}(1-x)^{-2q} + 2q \cdot x^{-p}(1-x)^{-2q-1} \\ &= x^{-p}(1-x)^{-2q}(-p \cdot x^{-1} + 2q \cdot (1-x)^{-1}) \\ &= f(x) \cdot (-p \cdot x^{-1} + 2q \cdot (1-x)^{-1}) \\ f''(x) &= f(x) \cdot (p \cdot x^{-2} + 2q \cdot (1-x)^{-2}) + f'(x) \cdot (-p \cdot x^{-1} + 2q \cdot (1-x)^{-1}) \\ f''(x^*) &= f(x^*) \cdot (p \cdot (x^*)^{-2} + 2q \cdot (1 - (x^*))^{-2}) > 0 \end{aligned}$$

Hence the run time to compute $\widehat{\mathcal{P}}_{uv}^p \subseteq_{rep}^q \mathcal{P}_{uv}^p$ is minimized when $x = \frac{p}{p+2q}$.

Lemma 8.3. *Let D be a directed graph with n vertices and m edges, and $u \in V(D)$. Let ℓ be a positive integer. Then for every $v \in V(D) \setminus \{u\}$ and integer $2 \leq p \leq \ell$ there is an algorithm that computes a family $\widehat{\mathcal{P}}_{uv}^p \subseteq_{rep}^{\ell-p} \mathcal{P}_{uv}^p$ of size $\binom{2\ell-p}{p}^p \binom{2\ell-p}{2\ell-2p}^{\ell-p} \cdot 2^{o(\ell)}$ in time $\mathcal{O}\left(2^{o(\ell)} \cdot m \log n \cdot \max_{i \in [p]} \left\{ \left(\frac{2\ell-i}{i}\right)^i \left(\frac{2\ell-i}{2\ell-2i}\right)^{2\ell-2i} \right\}\right)$*

Proof. The proof is same as the proof of Lemma 8.2, except the choice of x while applying Theorem 7.8 (instead of Corollary 7.1). As in the proof of Lemma 8.2, we have a dynamic programming table \mathcal{D} where the rows are indexed from integers in $\{2, \dots, p\}$ and the columns are indexed from vertices in $\{v_1, \dots, v_{n-1}\}$. The entry $\mathcal{D}[i, v]$ will store the family $\widehat{\mathcal{P}}_{uv}^i \subseteq_{rep}^{\ell-i} \mathcal{P}_{uv}^i$. We fill the entries in the matrix \mathcal{D} in the increasing order of rows. For $i = 2$, $\mathcal{D}[2, v] = \{\{u, v\}\}$ if $uv \in A(D)$. Assume that we have filled all the entries until the row i . Let

$$\mathcal{N}_{uv}^{i+1} = \bigcup_{w \in N^-(v)} \widehat{\mathcal{P}}_{uw}^i \bullet \{v\}.$$

Due to Claim 8.1, we have that $\mathcal{N}_{uv}^{i+1} \subseteq_{rep}^{\ell-(i+1)} \mathcal{P}_{uv}^{i+1}$. Lemma 7.1 implies that $\widehat{\mathcal{N}}_{uv}^{i+1} =$

$\widehat{\mathcal{P}}_{uv}^{i+1} \subseteq_{rep}^{\ell-i-1} \mathcal{P}_{uv}^{i+1}$. We assign this family to $\mathcal{D}[i+1, v]$.

Now we explain the computation of $\widehat{\mathcal{N}}_{uv}^{i+1} = \widehat{\mathcal{P}}_{uv}^{i+1}$. For any j , to compute $\widehat{\mathcal{N}}_{uv}^j = \widehat{\mathcal{P}}_{uv}^j$, we apply Theorem 7.3 with the value x_j for x , where

$$x_j = \frac{j}{j + 2(\ell - j)} = \frac{j}{2\ell - j}$$

Let $s_{j, \ell-j}$ be the size of the representative family $\widehat{\mathcal{N}}_{uv}^j = \widehat{\mathcal{P}}_{uv}^j$ when we apply Theorem 7.3 with the value x_j . That is $s_{j, \ell-j} = (x_j)^{-j} (1 - x_j)^{-\ell+j} \cdot 2^{o(\ell)}$. Assume that we have computed $\widehat{\mathcal{P}}_{uv}^j$ of size $s_{j, \ell-j}$ and stored it in $\mathcal{D}[j, w]$ for all $j \leq i$ and $w \in \{v_1, \dots, v_{n-1}\}$. Now consider the computation of $\widehat{\mathcal{N}}_{uv}^{i+1} = \widehat{\mathcal{P}}_{uv}^{i+1}$. We apply Theorem 7.3 with value x_{i+1} for x to compute $\widehat{\mathcal{N}}_{uv}^{i+1} \subseteq_{rep}^{\ell-(i+1)} \mathcal{N}_{uv}^{i+1}$. Since $\mathcal{N}_{uv}^{i+1} = \bigcup_{w \in N^-(v)} \widehat{\mathcal{P}}_{uw}^i \bullet \{v\}$, we have that

$$\begin{aligned} |\mathcal{N}_{uv}^{i+1}| &\leq s_{i, \ell-i} \cdot d^-(v) \\ &\leq (x_i)^{-i} (1 - x_i)^{-\ell+i} \cdot 2^{o(\ell)} d^-(v) \end{aligned}$$

By Theorem 7.3, the running time to compute $\widehat{\mathcal{N}}_{uv}^{i+1}$ is,

$$s_{i, \ell-i} \cdot (1 - x_{i+1})^{-\ell+(i+1)} \cdot 2^{o(\ell)} \cdot d^-(v) \cdot \log n \quad (8.1)$$

To analyze the running time further we need the following claim.

Claim 8.2. For any $3 < i < p$, $s_{i, \ell-i} \leq e^2 \cdot (i+1) \cdot s_{i+1, \ell-i-1}$.

Proof. By applying the definition of s_i and x_{i+1} we get the following inequality.

$$\begin{aligned}
\frac{s_{i,\ell-i}}{s_{i+1,\ell-i-1}} &= \frac{x_i^{-i}(1-x_i)^{-\ell+i}}{x_{i+1}^{-(i+1)}(1-x_{i+1})^{-\ell+(i+1)}} \\
&= \left(\frac{2\ell-i}{i}\right)^i \left(\frac{2\ell-i}{2\ell-2i}\right)^{\ell-i} \left(\frac{i+1}{2\ell-(i+1)}\right)^{i+1} \left(\frac{2\ell-2(i+1)}{2\ell-(i+1)}\right)^{\ell-(i+1)} \\
&= \left(\frac{2\ell-i}{2\ell-(i+1)}\right)^\ell \cdot \frac{(i+1)^{i+1}}{i^i} \cdot \frac{(2\ell-2(i+1))^{\ell-(i+1)}}{(2\ell-2i)^{\ell-i}} \\
&\leq \left(1 + \frac{1}{2\ell-(i+1)}\right)^{2\ell-(i+1)} \cdot (i+1) \cdot \left(1 + \frac{1}{i}\right)^i \\
&\leq e^2 \cdot (i+1).
\end{aligned}$$

In the last transition we used that $(1 + 1/x)^x < e$ for every $x > 0$. \square

From Equation 8.1 and Claim 8.2 we have that the running time for computing $\widehat{\mathcal{P}}_{uv}^p$ is bounded by

$$\begin{aligned}
&\mathcal{O}\left(\sum_{i=2}^p \sum_{j=1}^{n-1} s_{i,\ell-i} \cdot d^-(v_j) \cdot (1-x_i)^{-\ell+i} \cdot 2^{o(\ell)} \cdot \log n\right) \\
&= \mathcal{O}\left(2^{o(\ell)} \cdot m \log n \cdot \max_{i \in [p]} \left\{ \left(\frac{2\ell-i}{i}\right)^i \left(\frac{2\ell-i}{2\ell-2i}\right)^{2\ell-2i} \right\}\right)
\end{aligned}$$

The size of the family $\widehat{\mathcal{P}}_{uv}^p \subseteq_{rep}^{\ell-p} \mathcal{N}_{uv}^p \subseteq_{rep}^{\ell-p} \mathcal{P}_{uv}^p$ is,

$$s_{p,\ell-p} = (x_p)^{-p}(1-x_p)^{-\ell+p} \cdot 2^{o(\ell)} = \left(\frac{2\ell-p}{p}\right)^p \left(\frac{2\ell-p}{2\ell-2p}\right)^{\ell-p} \cdot 2^{o(\ell)}.$$

This completes the proof. \square

We now have a faster algorithm to compute the representative family $\widehat{\mathcal{P}}_{uv}^k \subseteq_{rep}^k \mathcal{P}_{uv}^p$. Using Lemma 8.3, we can compute $\widehat{\mathcal{P}}_{uv}^k$, for all $v \in V(D) \setminus \{u\}$ in time

$$\mathcal{O}\left(2^{o(k)} \cdot m \log n \cdot \max_{i \in [p]} \left\{ \left(\frac{4k-i}{i}\right)^i \left(\frac{4k-i}{4k-2i}\right)^{4k-2i} \right\}\right).$$

Simple calculus shows that the maximum is attained for $i = k$. Hence the running time to compute $\widehat{\mathcal{P}}_{uv}^k$ for all $u, v \in V(D)$ is upper bounded by $\mathcal{O}(6.75^{k+o(k)} nm \log n)$.

This yields an improved bound for the running time of our algorithm for LONG DIRECTED CYCLE.

We apply Lemma 8.3 for each $u \in V(D)$ with $\ell = 2k$ and $p = k$. Thus, we can compute $\widehat{\mathcal{P}}_{uv}^k \subseteq_{rep}^k \mathcal{P}_{uv}^p$ for all $u, v \in V(D)$ in time $\mathcal{O}(6.75^{k+o(k)}nm \log n)$. The size of the family $\widehat{\mathcal{P}}_{uv}^k$ for any $u, v \in V(D)$ is upper bounded by $\mathcal{O}(4.5^{k+o(k)})$. Thus, if we now loop over every set in the representative families and run a breadth first search algorithm, just as in the proof of Theorem 8.1, this will take at most $\mathcal{O}(6.75^{k+o(k)}nm \log n + 4.5^{k+o(k)}(n^3 + n^2m))$ time. Hence we arrive at the following theorem.

Theorem 8.2. *There is a $\mathcal{O}(6.75^{k+o(k)}mn^2)$ time algorithm for LONG DIRECTED CYCLE*

Chapter 9

k -Path and k -Tree

In this chapter we study k -PATH, k -TREE and k -SUBGRAPH ISOMORPHISM problems. In the k -PATH problem we are given an undirected n -vertex graph G and integer k . The question is if G contains a path of length k . k -PATH is a special case of the k -SUBGRAPH ISOMORPHISM problem, where for given n -vertex graph G and k -vertex graph F , the question is whether G contains a subgraph isomorphic to F . Another special case of the k -SUBGRAPH ISOMORPHISM is k -TREE, where the given k -vertex graph F is a tree.

Previous Work. k -PATH was studied intensively within the parameterized complexity paradigm. For n -vertex graphs the problem is trivially solvable in time $\mathcal{O}(n^k)$. Monien [97] showed that the problem is fixed parameter tractable. Monien used representative families in set systems for his k -PATH algorithm [97] and Plehn and Voigt extended this algorithm to SUBGRAPH ISOMORPHISM in [109]. This led Papadimitriou and Yannakakis [105] to conjecture that the problem is solvable in polynomial time for $k = \log n$. This conjecture was resolved in a seminal paper of

Reference	Randomized	Deterministic
Monien [97]	-	$\mathcal{O}(k!nm)$
Bodlaender [21]	-	$\mathcal{O}(k!2^k n)$
Alon et al. [4]	$\mathcal{O}(5.44^k n)$	$\mathcal{O}(c^k n \log n)$ for a large c
Kneis et al. [77]	$\mathcal{O}^*(4^k)$	$\mathcal{O}^*(16^k)$
Chen et al. [30]	$\mathcal{O}(4^k k^{2.7} m)$	$4^{k+\mathcal{O}(\log^3 k)} nm$
Koutis [80]	$\mathcal{O}^*(2.83^k)$	-
Williams [119]	$\mathcal{O}^*(2^k)$	-
Björklund et al. [17]	$\mathcal{O}^*(1.66^k)$	-
Fomin et al. [57]	-	$\mathcal{O}(2.851^k n \log^2 n)$
Theorem 9.1	-	$\mathcal{O}(2.619^k n \log n)$

Table 9.1: Results for k -PATH

Alon et al. [4], who introduced the method of color-coding and obtained the first single exponential algorithm for the problem. Actually, the method of Alon et al. can be applied for more general problems, like finding a k -path in directed graphs, or to solve the SUBGRAPH ISOMORPHISM problem in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(t)}$, when the treewidth of the pattern graph is bounded by t . There has been a lot of efforts in parameterized algorithms to reduce the base of the exponent of both deterministic as well as the randomized algorithms for the k -PATH problem, see Table 9.1. After the work of Alon et al. [4], there were several breakthrough ideas leading to faster and faster *randomized* algorithms. Concerning deterministic algorithms, no improvements occurred since 2007, when Chen et al. [33] showed a clever way of applying universal sets to reduce the running time of color-coding algorithm to $\mathcal{O}^*(4^{k+o(k)})$.

The weighted version of k -PATH is known as SHORT CHEAP TOUR. Let G be a graph with maximum edge cost W , then the problem is to find a path of length at least k where the total sum of costs on the edges is minimized. The algorithm of Björklund et al. [17] can be adapted to solve SHORT CHEAP TOUR in time $\mathcal{O}(1.66^k n^{\mathcal{O}(1)} W)$, however, their approach does not seem to be applicable to obtain algorithms with polylogarithmic dependence on W . Williams in [119] observed that a divide-and-color approach from [30] can be used to solve SHORT CHEAP TOUR in time $\mathcal{O}(4^k n^{\mathcal{O}(1)} \log W)$. As it was noted by Williams, the $\mathcal{O}(2^k n^{\mathcal{O}(1)})$ algorithm of his paper does not appear to extend to weighted graphs.

In addition to k -PATH problem, parameterized algorithms for two other variants of k -SUBGRAPH ISOMORPHISM, when F is a tree, and more generally, a graph of treewidth at most t , were studied in the literature. Alon et al. [4] showed that k -SUBGRAPH ISOMORPHISM, when the treewidth of the pattern graph is bounded by t , is solvable in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(t)}$. Cohen et al. gave a randomized algorithm that for an input digraph D decides in time $5.704^k n^{\mathcal{O}(1)}$ if D contains a given out-tree with k vertices [35]. They also showed how to derandomize the algorithm in time $6.14^k n^{\mathcal{O}(1)}$. Amini et al. [5] introduced an inclusion-exclusion based approach in the classical color-coding and gave a randomized $5.4^k n^{\mathcal{O}(t)}$ time algorithm and a deterministic $5.4^{k+o(k)} n^{\mathcal{O}(t)}$ time algorithm for the case when F has treewidth at most t . Koutis and Williams [82] generalized their algebraic approach for k -PATH to k -TREE and obtained a randomized algorithm running in time $2^k n^{\mathcal{O}(1)}$ for k -TREE. Fomin et al. [56], extended this result by providing a randomized algorithm for k -SUBGRAPH ISOMORPHISM running in time $2^k (nt)^{\mathcal{O}(t)}$, when the treewidth of F is at most t .

Our results. We give deterministic algorithms for k -PATH and k -TREE that run in time $\mathcal{O}(2.619^k n \log n)$ and $\mathcal{O}(2.619^k n^{\mathcal{O}(1)})$. The algorithm for k -TREE can be generalized to k -SUBGRAPH ISOMORPHISM for the case when the pattern graph F has treewidth at most t . This algorithm will run in time $\mathcal{O}(2.619^k n^{\mathcal{O}(t)})$. Our approach can also be applied to find directed paths and cycles of length k in time $\mathcal{O}(2.619^k m \log n)$ and $\mathcal{O}(2.619^k n^{\mathcal{O}(1)})$ respectively, where m is the number of edges in the input graph. Another interesting feature of our approach is that due to using weighted representative families, we can handle the weighted version of the problem as well. Our approach provides deterministic $\mathcal{O}(2.619^k n^{\mathcal{O}(1)} \log W)$ time

algorithm for SHORT CHEAP TOUR and partially resolves an open question asked by Williams.

Organization of the chapter In Section 9.1 we prove our result on k -PATH. In section 9.2 we give an algorithm for k -TREE. This algorithm can be generalized to solve k -SUBGRAPH ISOMORPHISM when the treewidth of the pattern graph is a constant.

9.1 k -PATH

In this section we give an algorithm for k -PATH. We start by modifying the graph slightly. We add a new vertex, say s not present in $V(G)$, to G by making it adjacent to every vertex in $V(G)$. Let the modified graph be called G' . It is clear that G has a path of length k if and only if G' has a path of length $k + 1$ starting from s . Hence we redefine the problem k -PATH as follows.

<p>k-PATH</p> <p>Input: An undirected n-vertex and m-edge graph G, $s \in V(G)$ and a positive integer k.</p> <p>Question: Does there exist a simple path of length $k + 1$ in G'?</p>	<p>Parameter: k</p>
--	---

For a given pair of vertices $s, v \in V(G)$, recall that we defined (in Chapter 8)

$$\mathcal{P}_{sv}^i = \left\{ X \mid X \subseteq V(G), v, s \in X, |X| = i \text{ and there is a path from } s \text{ to } v \text{ of length } i \text{ in } G \text{ with all the vertices belonging to } X. \right\}$$

The problem can be reformulated to asking whether there exists $v \in V(G)$ such that \mathcal{P}_{sv}^{k+2} is non-empty. Our algorithm will check whether \mathcal{P}_{uv}^{k+2} is non-empty by computing $\widehat{\mathcal{P}}_{sv}^{k+2} \subseteq_{rep}^0 \mathcal{P}_{sv}^{k+2}$ and checking whether $\widehat{\mathcal{P}}_{sv}^{k+2}$ is non-empty. The correctness of this algorithm is as follows. If \mathcal{P}_{sv}^{k+2} is non-empty then \mathcal{P}_{sv}^{k+2} contains some set A which does not intersect the empty set \emptyset . But then $\widehat{\mathcal{P}}_{sv}^{k+2} \subseteq_{rep}^0 \mathcal{P}_{sv}^{k+2}$ must also contain a set which does not intersect with \emptyset , and hence $\widehat{\mathcal{P}}_{sv}^{k+2}$ must be non-empty as well. Thus, having computed the representative families $\widehat{\mathcal{P}}_{sv}^{k+2}$ all we need to do is to check whether there is a vertex v such that $\widehat{\mathcal{P}}_{sv}^{k+2}$ is non-empty. All that remains is an algorithm that computes the representative families $\widehat{\mathcal{P}}_{sv}^{k+2} \subseteq_{rep}^0 \mathcal{P}_{sv}^{k+2}$ for all $v \in V(G) \setminus \{s\}$.

Now using Lemma 8.3 (by setting $\ell = p = k + 2$) we compute $\widehat{\mathcal{P}}_{sv}^{k+2} \subseteq_{rep}^0 \mathcal{P}_{sv}^{k+2}$ for

all $v \in V(G) \setminus \{s\}$ in time

$$2^{o(k)} \cdot m \log n \cdot \max_{i \in [k+2]} \left\{ \left(\frac{2(k+2) - i}{i} \right)^i \left(\frac{2(k+2) - i}{2(k+2) - 2i} \right)^{2(k+2) - 2i} \right\}.$$

Simple calculus shows that the running time is maximized for $i = (1 - \frac{1}{\sqrt{5}})(k+2)$, and thus the running time to compute $\widehat{\mathcal{P}}_{sv}^{k+2} \subseteq_{rep}^0 \mathcal{P}_{sv}^{k+2}$ for all $v \in V(G) \setminus \{s\}$ together is upper bounded by $\phi^{2k+o(k)} m \log n = \mathcal{O}(2.619^k m \log n)$, where ϕ is the golden ratio $\frac{1+\sqrt{5}}{2}$. Furthermore, in the same time every set in $\widehat{\mathcal{P}}_{sv}^p$ can be ordered in a way that it corresponds to an undirected path in G . A graph G has a path of length $k+1$ starting from s if and only if for some $v \in V(G) \setminus \{s\}$, we have that $\widehat{\mathcal{P}}_{sv}^{k+2} \neq \emptyset$. Thus the running time of this algorithm is upper bounded by $\mathcal{O}(2.619^k m \log n)$. Let us remark that almost the same arguments show that the version of the problem on directed graphs is solvable within the same running time. However on undirected graphs we can speed up the algorithm slightly by using the following standard trick. We need the following result.

Proposition 9.1 ([21]). *There exists an algorithm, that given a graph G and an integer k , in time $\mathcal{O}(k^2 n)$ either finds a simple path of length $\geq k$ or computes a DFS (depth first search) tree rooted at some vertex of G of depth at most k .*

We first apply Proposition 9.1 and in time $\mathcal{O}(k^2 n)$ either find a simple path of length $\geq k$ in G or compute a DFS tree of G of depth at most k . In the former case we simply output the same path. In the later case since all the root to leaf paths are upper bounded by k and there are no cross edges in a DFS tree, we have that the number of edges in G is upper bounded by $\mathcal{O}(k^2 n)$. Now on this G we apply the representative set based algorithm described above. This results in the following theorem.

Theorem 9.1. *k -PATH can be solved in time $\mathcal{O}(2.619^k n \log n)$.*

Our algorithm for k -PATH can be used to solve the weighted version of the problem, i.e, SHORT CHEAP TOUR. In this problem a graph G with maximum edge cost W is given, and the objective is to find a path of length at least k where the total sum of costs on the edges is minimized.

Theorem 9.2. *SHORT CHEAP TOUR can be solved in time $\mathcal{O}(2.619^k n^{\mathcal{O}(1)} \log W)$.*

9.2 k -TREE and k -SUBGRAPH ISOMORPHISM

In this section we consider the following problem.

<p>k-TREE</p> <p>Input: An undirected n-vertex, m-edge graph G and a tree T on k vertices.</p> <p>Question: Does G contains a subgraph isomorphic to T?</p>	<p>Parameter: k</p>
---	---

The algorithm for k -TREE is more involved than for k -PATH. The reason to that is due to the fact that paths poses perfectly balanced separators of size one while trees not. We select a leaf r of T and root the tree at r . For vertices $x, y \in V(T)$ we say that $y \leq x$ if x lies on the path from y to r in T (if $x = r$ we also say that $y \leq x$). For a set C of vertices in T we will say that $x \preceq_C y$ if $x \leq y$ and there is no $z \in C$ such that $x \leq z$ and $z \leq y$. For a pair x, y of vertices such that $y \leq x$ in T we define

$$C^{xy} = \begin{cases} \emptyset & \text{if } xy \in E(T), \\ \text{The unique component } C \text{ of } T \setminus \{x, y\} \text{ s.t } N(C) = \{x, y\} & \text{otherwise.} \end{cases}$$

We also define $T^{uv} = T[C^{uv} \cup \{u, v\}]$. We start by making a few simple observations about sets of vertices in trees.

Lemma 9.1. *For any tree T , a pair $\{x, y\}$ of vertices in $V(T)$ and integer $c \geq 1$ there exists a set W of vertices such that $\{x, y\} \subseteq W$, $|W| = \mathcal{O}(c)$ and every connected component U of $T \setminus W$ satisfies $|U| \leq \frac{|V(T)|}{c}$ and $|N(U)| \leq 2$.*

Proof. We first find a set W_1 of size at most c such that every connected component U of $T \setminus W_1$ satisfies $|U| \leq \frac{|V(T)|}{c}$. Start with $W_1 = \emptyset$ and select a lowermost vertex $u \in V(T)$ such that the subtree rooted at u has at least $\frac{|V(T)|}{c}$ vertices. Add u to W_1 and remove the subtree rooted at u from T . The process must stop after c iterations since each iteration removes $\frac{|V(T)|}{c}$ vertices of T . Each component U of $T \setminus W_1$ satisfies $|U| \leq \frac{|V(T)|}{c}$ because (a) whenever a vertex u is added to W_1 , all components below u have size strictly less than $\frac{|V(T)|}{c}$ and (b) when the process ends the subtree rooted at r has size at most $|U| \leq \frac{|V(T)|}{c}$. Now, insert x and y into W_1 as well.

We build W from W_1 by taking the *least common ancestor closure* of W_1 ; start with $W = W_1$ and as long as there exist two vertices u and v in W such that their least common ancestor w is not in W , add w to W . Standard counting arguments on trees imply that this process will never increase the size of W by more than a factor

2, hence $|W| \leq 2|W_1| = O(c)$.

We claim that every connected component U of $T \setminus W$ satisfies $N(U) \leq 2$. Suppose not and let u be the vertex of u closest to the root. Since $N(U) > 2$ at least two vertices v and w in $N(U)$ are descendants of u . Since U is connected v and w can't be descendants of each other, but then the least common ancestor of v and w is in U , contradicting the construction of W . \square

Observation 9.1. *For any tree T , set $W \subseteq V(T)$ and component U of $T \setminus W$ such that $|N(U)| = 1$, U contains a leaf of T .*

Proof. $T[U \cup N(U)]$ is a tree on at least two vertices and hence it has at least two leaves. At most one of these leaves is in $N(U)$, the other one is also a leaf of T . \square

Lemma 9.2. *Let $W \subseteq V(T)$ be a set of vertices such that for every pair of vertices in W their least common ancestor is also in W . Let X be a set containing one leaf of T from each connected component U of $T \setminus W$ such that $|N(U)| = 1$. Then, for every connected component U such that $|N(U)| = 1$ there exist $x \in W$, $y \in X$ such that $U = C^{xy} \cup \{y\}$. For every other connected component U there exist $x, y \in W$ such that $U = C^{xy}$.*

Proof. It follows from the argument at the end of the proof of Lemma 9.1 that every component U of $T \setminus W$ satisfies $|N(U)| \leq 2$. If $|N(U)| = 2$, let $N(U) = \{x, y\}$. We have that $x \leq y$ or $y \leq x$ since least common ancestor of x and y can not be in U and would therefore be in $N(U)$, contradicting $|N(U)| = 2$. Without loss of generality $y \leq x$. But then $U = C^{xy}$. If $|N(U)| = 1$, let $N(U) = \{x\}$. By Observation 9.1 U contains a leaf y of T . Then $U = C^{xy} \cup \{y\}$. \square

Given two graphs F and H , a graph *homomorphism* from F to H is a map f from $V(F)$ to $V(H)$, that is $f : V(F) \rightarrow V(H)$, such that if $uv \in E(F)$, then $f(u)f(v) \in E(H)$. Furthermore, when the map f is injective, f is called a *subgraph isomorphism*. For every $x, y \in V(T)$ such that $y \leq x$, and every u, v in $V(G)$ we

define

$$\mathcal{F}_{uv}^{xy} = \left\{ F \in \binom{V(G) \setminus \{u, v\}}{|C^{xy}|} : \exists \text{ subgraph isomorphism } f \text{ from } T^{xy} \text{ to } G[F \cup \{u, v\}] \text{ such that } f(x) = u \text{ and } f(y) = v \right\}$$

Let us remind that for a set X and a family \mathcal{A} , we use $\mathcal{A}+X$ to denote $\{A \cup X : A \in \mathcal{A}\}$. For every $x, y \in V(T)$ such that $y \leq x$, and every u in $V(G)$ we define

$$\mathcal{F}_{u*}^{xy} = \bigcup_{v \in V(G) \setminus \{u\}} \mathcal{F}_{uv}^{xy} + \{v\} \quad (9.1)$$

In order to solve the problem it is sufficient to select an arbitrary leaf ℓ of T and determine whether there exists a $u \in V(G)$ such that the family $\mathcal{F}_{u*}^{r\ell}$ is non-empty. We show that the collections of families $\{\mathcal{F}_{uv}^{xy}\}$ and $\{\mathcal{F}_{u*}^{xy}\}$ satisfy a recurrence relation. We will then exploit this recurrence relation to get a fast algorithm for k -TREE.

Lemma 9.3. *For every $x, y \in V(T)$ such that $y \leq x$, every $\widehat{W} = W \cup \{x, y\}$ where $W \subseteq C^{xy}$, such that for every pair of vertices in \widehat{W} their least common ancestor is also in \widehat{W} , every $X \subseteq C^{xy} \setminus W$ such that X contains exactly one leaf of T in each connected component U of $T^{xy} \setminus \widehat{W}$ with $|N(U)| = 1$, the following recurrence holds.*

$$\mathcal{F}_{uv}^{xy} = \bigcup_{\substack{g: \widehat{W} \rightarrow V(G) \\ g(x)=u \wedge g(y)=v}} \left[\left(\prod_{\substack{x', y' \in \widehat{W} \\ y' \preceq_{\widehat{W}} x'}} \mathcal{F}_{g(x')g(y')}^{x'y'} \cdot \prod_{\substack{x' \in \widehat{W}, y' \in X \\ y' \preceq_{\widehat{W}} x'}} \mathcal{F}_{g(x')*}^{x'y'} \right) + g(W) \right] \quad (9.2)$$

Here the union goes over all $O(n^{|\widehat{W}|})$ injective maps g from \widehat{W} to $V(G)$ such that $g(x) = u$ and $g(y) = v$, and by $g(W)$ we mean $\{g(c) : c \in W\}$.

Proof. For the \subseteq direction of the equality consider any subgraph isomorphism f from T^{xy} to $V(G)$ such that $f(x) = u$ and $f(y) = v$. Let g be the restriction of f to W . The map f can be considered as a collection of subgraph isomorphisms with one isomorphism for each $x', y' \in \widehat{W}$ such that $y' \preceq_{\widehat{W}} x$ from $T^{x'y'}$ to G such that $f(x') = g(x')$ and $f(y') = g(y')$, and one isomorphism for each $x' \in \widehat{W}, y' \in X$ such that $y' \preceq_{\widehat{W}} x$ from $T^{x'y'}$ to G such that $f(x') = g(x')$. Taking the union of the

ranges of each of the small subgraph isomorphisms clearly give the range of f . Here we used Lemma 9.2 to argue that for every connected component U of $T^{xy} \setminus \widehat{W}$ we have that $T[U \cup N(U)]$ is in fact on the form $T^{x'y'}$ for some x', y' .

For the reverse direction take any collection of subgraph isomorphisms with one isomorphism f for each $x', y' \in \widehat{W}$ such that $y' \preceq_{\widehat{W}} x$ from $T^{x'y'}$ to G such that $f(x') = g(x')$ and $f(y') = g(y')$, and one isomorphism for each $x' \in \widehat{W}, y' \in X$ such that $y' \preceq_{\widehat{W}} x$ from $T^{x'y'}$ to G such that $f(x') = g(x')$, such that the range of all of these subgraph isomorphisms are pairwise disjoint (except on vertices in \widehat{W}). Since all of these subgraph isomorphisms agree on the set W they can be glued together to a subgraph isomorphism from T^{xy} to G . \square

Our goal is to compute for every $x, y \in V(T)$ such that $y \leq x$ and $u, v \in V(G)$ a family $\hat{\mathcal{F}}_{uv}^{xy}$ such that $\hat{\mathcal{F}}_{uv}^{xy} \subseteq_{rep}^{k-|C^{xy}|} \mathcal{F}_{uv}^{xy}$ and for every $x, y \in V(T)$ such that $y \leq x$ and $u \in V(G)$ a family $\hat{\mathcal{F}}_{u*}^{xy}$ such that $\hat{\mathcal{F}}_{u*}^{xy} \subseteq_{rep}^{k-|C^{xy}|-1} \mathcal{F}_{u*}^{xy}$. We will also maintain the following size invariants.

$$|\hat{\mathcal{F}}_{uv}^{xy}| \leq \left(\frac{2k - |C^{xy}|}{|C^{xy}|} \right)^{|C^{xy}|} \left(\frac{2k - |C^{xy}|}{2k - 2|C^{xy}|} \right)^{k-|C^{xy}|} 2^{o(k)} \quad (9.3)$$

$$|\hat{\mathcal{F}}_{u*}^{xy}| \leq \left(\frac{2k - |C^{xy}| - 1}{|C^{xy}| + 1} \right)^{|C^{xy}|+1} \left(\frac{2k - |C^{xy}| - 1}{2k - 2|C^{xy}| - 2} \right)^{k-|C^{xy}|-1} 2^{o(k)} \quad (9.4)$$

Let the right hand side of equation 9.3 be s_{xy} and the right had side of equation 9.4 be s_{xy}^* . We first compute such families $\hat{\mathcal{F}}_{uv}^{xy}$ for all $x, y \in V(T)$ such that $y \leq x$ and $xy \in E(T)$. Observe that in this case we have

$$\mathcal{F}_{uv}^{xy} = \begin{cases} \{\emptyset\} & \text{if } uv \in E(G), \\ \emptyset & \text{if } uv \notin E(G). \end{cases}$$

For each $x, y \in V(T)$ such that $y \leq x$ and $xy \in E(T)$ and every $u, v \in V(G)$ we set $\hat{\mathcal{F}}_{uv}^{xy} = \mathcal{F}_{uv}^{xy}$. We can now for compute $\hat{\mathcal{F}}_{u*}^{xy}$ for every $x, y \in V(T)$ such that $y \leq x$ and $xy \in E(T)$ and every $u \in V(G)$ by applying Equation 9.1. Clearly the computed families are within the required size bounds.

We now show how to compute a family $\hat{\mathcal{F}}_{uv}^{xy}$ of size s_{xy} for every $x, y \in V(T)$ such that $y \leq x$ and $u, v \in V(G)$ and $|C^{xy}| = t$, assuming that the families $\hat{\mathcal{F}}_{uv}^{xy}$ and $\hat{\mathcal{F}}_{u*}^{xy}$ have been computed for every $x, y \in V(T)$ such that $y \leq x$ and $u, v \in V(G)$ and $|C^{xy}| < t$. We also assume that for each family $\hat{\mathcal{F}}_{uv}^{xy}$ that has been computed, $|\hat{\mathcal{F}}_{uv}^{xy}| \leq s_{xy}$. Similarly we assume that for each family $\hat{\mathcal{F}}_{u*}^{xy}$ that has been computed,

$$|\hat{\mathcal{F}}_{u*}^{xy}| \leq s_{xy}^*.$$

We fix a constant c whose value will be decided later. First apply Lemma 9.1 on T^{xy} , vertex pair $\{x, y\}$ and constant c and obtain a set \widehat{W} such that $\{x, y\} \subseteq \widehat{W}$ and every connected component U of $T \setminus \widehat{W}$ satisfies $|U| \leq \frac{|V(T)|}{c}$ and $|N(U)| \leq 2$. Select a set $X \subseteq V(T^{x,y}) \setminus \widehat{W}$ such that each connected component U of $T \setminus \widehat{W}$ with $|N(U)| = 1$ contains exactly one leaf which is in X . Now, set $W = \widehat{W} \setminus \{x, y\}$ and consider Equation 9.2 for $\hat{\mathcal{F}}_{uv}^{xy}$ for this choice of x, y, W and X . Define

$$\tilde{\mathcal{F}}_{uv}^{xy} = \bigcup_{\substack{g: \widehat{W} \rightarrow V(G) \\ g(x)=u \wedge g(y)=v}} \left[\left(\dot{\prod}_{\substack{x', y' \in \widehat{W} \\ y' \preceq_{\widehat{W}} x'}} \hat{\mathcal{F}}_{g(x')g(y')}^{x'y'} \bullet \dot{\prod}_{\substack{x' \in \widehat{W}, y' \in X \\ y' \preceq_{\widehat{W}} x'}} \hat{\mathcal{F}}_{g(x')*}^{x'y'} \right) + g(W) \right] \quad (9.5)$$

Lemma 9.3 together with Lemmata 7.2 and 7.3 directly imply that $\tilde{\mathcal{F}}_{uv}^{xy} \subseteq_{rep}^{k-|C^{xy}|} \mathcal{F}_{uv}^{xy}$. Furthermore, each family on the right hand side of Equation 9.5 has already been computed, since $C^{x'y'} \subset C^{xy}$ and so $|C^{x'y'}| < t$. For a fixed injective map $g : W \rightarrow V(G)$ we define

$$\tilde{\mathcal{F}}_g^{xy} = \left(\dot{\prod}_{\substack{x', y' \in \widehat{W} \\ y' \preceq_{\widehat{W}} x'}} \hat{\mathcal{F}}_{g(x')g(y')}^{x'y'} \bullet \dot{\prod}_{\substack{x' \in \widehat{W}, y' \in X \\ y' \preceq_{\widehat{W}} x'}} \hat{\mathcal{F}}_{g(x')*}^{x'y'} \right) + g(W) \quad (9.6)$$

It follows directly from the definition of $\tilde{\mathcal{F}}_{uv}^{xy}$ and $\tilde{\mathcal{F}}_g^{xy}$ that

$$\tilde{\mathcal{F}}_{uv}^{xy} = \bigcup_{\substack{g: \widehat{W} \rightarrow V(G) \\ g(x)=u \wedge g(y)=v}} \tilde{\mathcal{F}}_g^{xy}.$$

Our goal is to compute a family $\hat{\mathcal{F}}_{uv}^{xy} \subseteq_{rep}^{k-|C^{xy}|} \tilde{\mathcal{F}}_{uv}^{xy}$ such that $|\hat{\mathcal{F}}_{uv}^{xy}| \leq s_{xy}$. Lemma 7.1 then implies that $\hat{\mathcal{F}}_{uv}^{xy} \subseteq_{rep}^{k-|C^{xy}|} \mathcal{F}_{uv}^{xy}$. To that end, we define the function **reduce**. Given a family \mathcal{F} of sets of size p , the function **reduce** will run the algorithm of Theorem 7.3 on \mathcal{F} with $x = \frac{p}{2k-p}$ and produce a family of size $\left(\frac{2k-p}{p}\right)^p \left(\frac{2k-p}{2k-2p}\right)^{k-p} 2^{o(k)}$ that $k-p$ represents \mathcal{F} .

We will compute for each $g : \widehat{W} \rightarrow V(G)$ such that $g(x) = u$ and $g(y) = v$ a family

$\hat{\mathcal{F}}_g^{xy}$ of size at most s_{xy} such that $\hat{\mathcal{F}}_g^{xy} \subseteq_{rep}^{k-|C^{xy}|} \tilde{\mathcal{F}}_g^{xy}$. We will then set

$$\hat{\mathcal{F}}_{uv}^{xy} = \text{reduce} \left(\bigcup_{\substack{g: \widehat{W} \rightarrow V(G) \\ g(x)=u \wedge g(y)=v}} \hat{\mathcal{F}}_g^{xy} \right). \quad (9.7)$$

To compute $\hat{\mathcal{F}}_g^{xy}$, inspect Equation 9.6. Equation 9.6 shows that $\tilde{\mathcal{F}}_g^{xy}$ basically is a long chain of \bullet operations, specifically

$$\tilde{\mathcal{F}}_g^{xy} = \left(\hat{F}_1 \bullet \hat{F}_2 \bullet \hat{F}_3 \dots \bullet \hat{F}_\ell \right) + g(W) \quad (9.8)$$

We define (and compute) $\hat{\mathcal{F}}_g^{xy}$ as follows

$$\hat{\mathcal{F}}_g^{xy} = \text{reduce} \left(\text{reduce} \left(\dots \text{reduce} \left(\text{reduce} \left(\hat{F}_1 \bullet \hat{F}_2 \right) \bullet \hat{F}_3 \right) \bullet \dots \right) \bullet \hat{F}_\ell \right) + g(W) \quad (9.9)$$

$\hat{\mathcal{F}}_g^{xy} \subseteq_{rep}^{k-|C^{xy}|} \tilde{\mathcal{F}}_g^{xy}$ and thus also $\hat{\mathcal{F}}_{uv}^{xy} \subseteq_{rep}^{k-|C^{xy}|} \tilde{\mathcal{F}}_{uv}^{xy} \subseteq_{rep}^{k-|C^{xy}|} \mathcal{F}_{uv}^{xy}$ follows from Lemma 7.3 and Theorem 7.3. Since the last operation we do in the construction of $\hat{\mathcal{F}}_{uv}^{xy}$ is a call to **reduce**, $|\hat{\mathcal{F}}_{uv}^{xy}| \leq s_{xy}$ follows from Theorem 7.3. To conclude the computation we set

$$\tilde{\mathcal{F}}_{u*}^{xy} = \text{reduce} \left(\bigcup_{v \in V(G) \setminus \{u\}} \hat{\mathcal{F}}_{uv}^{xy} + \{v\} \right) \quad (9.10)$$

Lemma 7.3 and Theorem 7.3 imply that $\tilde{\mathcal{F}}_{u*}^{xy} \subseteq_{rep}^{k-|C^{xy}|-1} \mathcal{F}_{u*}^{xy}$ and that $|\hat{\mathcal{F}}_{u*}^{xy}| \leq s_{xy}^*$.

The algorithm computes the families $\hat{\mathcal{F}}_{u*}^{xy}$ and $\hat{\mathcal{F}}_{uv}^{xy}$ for every $x, y \in V(T)$ such that $y \leq x$. It then selects an arbitrary leaf ℓ of T and checks whether there exists a $u \in V(G)$ such that the family $\hat{\mathcal{F}}_{u*}^{r\ell}$ is non-empty. Since $\hat{\mathcal{F}}_{u*}^{r\ell} \subseteq_{rep}^0 \mathcal{F}_{u*}^{r\ell}$ there is a non-empty $\mathcal{F}_{u*}^{r\ell}$ if and only if there is a non empty $\hat{\mathcal{F}}_{u*}^{r\ell}$. Thus the algorithm can answer that there is a subgraph isomorphism from T to G if some $\hat{\mathcal{F}}_{u*}^{r\ell}$ is non-empty, and that no such subgraph isomorphism exists otherwise.

It remains to bound the running time of the algorithm. Up to polynomial factors, the running time of the algorithm is dominated by the computation of $\hat{\mathcal{F}}_{uv}^{xy}$. This computation consists of $n^{\mathcal{O}(|\widehat{W}|)}$ independent computations of the families $\hat{\mathcal{F}}_g^{xy}$. Each computation of the family $\hat{\mathcal{F}}_g^{xy}$ consists of at most k repeated applications of the operation

$$\hat{\mathcal{F}}^{i+1} = \text{reduce}(\hat{\mathcal{F}}^i \bullet \hat{\mathcal{F}}_{i+1}).$$

Here \mathcal{F}^i is a family of sets of size p , and so $|\mathcal{F}^i| \leq \left(\frac{2k-p}{p}\right)^p \left(\frac{2k-p}{2k-2p}\right)^{k-p} 2^{o(k)} \log n$. On the other hand $\hat{\mathcal{F}}_{i+1}$ is a family of sets of size $p' \leq \frac{k}{c}$ since we used Lemma 9.1 to construct \widehat{W} . Thus,

$$\begin{aligned} |\hat{\mathcal{F}}_{i+1}| &\leq \left(\frac{2k-p'}{p'}\right)^{p'} \left(\frac{2k-p'}{2k-2p'}\right)^{k-p'} 2^{o(k)} \\ &\leq \left(\frac{2k}{p'}\right)^{p'} \left(\frac{2k}{2k-2p'}\right)^{k-p'} 2^{o(k)} \\ &\leq \binom{k}{p'} \cdot 2^{p'} \cdot 2^{o(k)} \\ &\leq \binom{k}{k/c} \cdot 2^{k/c} \cdot 2^{o(k)} \\ &\leq 2^{(\varepsilon+1/c)k} \cdot 2^{o(k)} \end{aligned}$$

Thus $|\hat{\mathcal{F}}^i \bullet \hat{\mathcal{F}}_{i+1}| \leq \left(\frac{2k-p}{p}\right)^p \left(\frac{2k-p}{2k-2p}\right)^{k-p} 2^{(\varepsilon+1/c)k+o(k)}$. Hence, when we apply Theorem 7.3 with $x = \frac{p+p'}{2k-p-p'}$ to compute $\text{reduce}(\hat{\mathcal{F}}^i \bullet \hat{\mathcal{F}}_{i+1})$, this takes time

$$\begin{aligned} &|\hat{\mathcal{F}}^i \bullet \hat{\mathcal{F}}_{i+1}| \left(\frac{2k-p-p'}{2k-2p-2p'}\right)^{k-p-p'} 2^{o(k)} \log n \\ &\leq |\hat{\mathcal{F}}^i \bullet \hat{\mathcal{F}}_{i+1}| \left(\frac{2k-p}{2k-2p}\right)^{k-p} \left(\frac{2k-2p}{2k-2p-2p'}\right)^{k-p-p'} 2^{o(k)} \log n \\ &\leq |\hat{\mathcal{F}}^i \bullet \hat{\mathcal{F}}_{i+1}| \left(\frac{2k-p}{2k-2p}\right)^{k-p} \left(1 + \frac{p'}{k-p-p'}\right)^{k-p-p'} 2^{o(k)} \log n \\ &\leq |\hat{\mathcal{F}}^i \bullet \hat{\mathcal{F}}_{i+1}| \left(\frac{2k-p}{2k-2p}\right)^{k-p} e^{p'} 2^{o(k)} \log n \\ &\leq \left(\frac{2k-p}{p}\right)^p \left(\frac{2k-p}{2k-2p}\right)^{2k-2p} 2^{(\varepsilon+3/c)k+o(k)} \log n \end{aligned}$$

Since there are $n^{\mathcal{O}(|\widehat{W}|)}$ (which is equal to $n^{\mathcal{O}(c)}$, where c is a constant) independent computations of the families $\hat{\mathcal{F}}_g^{xy}$, the total running time is upper bounded by

$$\left(\frac{2k-p}{p}\right)^p \left(\frac{2k-p}{2k-2p}\right)^{2k-2p} 2^{(\varepsilon+3/c)k+o(k)} n^{\mathcal{O}(1)}$$

The maximum value of $\left(\frac{2k-p}{p}\right)^p \left(\frac{2k-p}{2k-2p}\right)^{2k-2p}$ is when $p = (1 - \frac{1}{\sqrt{5}})k$ and the maximum value is ϕ^{2k} , where ϕ is the golden ratio $\frac{1+\sqrt{5}}{2}$. Now we can choose the value of

c in such a way that $\varepsilon + 3/c$ is small enough and the above running time is bounded by $2.619^{k+o(k)}n^{\mathcal{O}(1)}$. This yields the following theorem.

Theorem 9.3. *k -TREE can be solved in time $2.619^{k+o(k)}n^{\mathcal{O}(1)}$.*

The algorithm for k -TREE can be generalized to k -SUBGRAPH ISOMORPHISM for the case when the pattern graph F has treewidth at most t . Towards this we need a result analogous to Lemma 9.1 for trees, which can be proved using the separation properties of graphs of treewidth at most t . This will lead to an algorithm with running time $2.619^{k+o(k)} \cdot n^{\mathcal{O}(t)}$.

Chapter 10

r -Dimensional k -Matching

Given a universe $U := U_1 \uplus \dots \uplus U_r$, and an r -uniform family $\mathcal{F} \subseteq U_1 \times \dots \times U_r$, the r -DIMENSIONAL k -MATCHING ((r, k) -DM) problem asks if \mathcal{F} admits a collection of k mutually disjoint sets. The special case in which $r = 3$ can be viewed as an immediate generalization of the matching problem on bipartite graphs to tripartite, three-uniform hypergraphs. The question of finding the largest 3D-Matching is a classic optimization problem, and the decision version is listed as one of the six fundamental NP-Complete problems in Garey and Johnson [62]. The (r, k) -DM problem may be thought as a restricted version of the more general r -SET k -PACKING ((r, k) -SP) problem, where no restrictions are assumed on the universe. More precisely, given a universe U , and an r -uniform family $\mathcal{F} \subseteq 2^U$, the (r, k) -SP problem asks if \mathcal{F} admits a collection of k mutually disjoint sets. In this chapter we study the weighted version of (r, k) -DM.

WEIGHTED r -DIMENSIONAL k -MATCHING ((r, k) -WDM) **Parameter:** $r \cdot k$
Input: A universe $U := U_1 \uplus \dots \uplus U_r$, a family $\mathcal{F} \subseteq U_1 \times \dots \times U_r$, a non-negative weight function $w : \mathcal{F} \rightarrow \mathbb{N}$, and positive integers k, W .
Question: Does \mathcal{F} have a collection \mathcal{M} of k mutually disjoint sets such that $w(\mathcal{M}) \geq W$?

Prior Work and Our Result. The problem (r, k) -DM, including their weighted versions, have enjoyed substantial attention in the context of exact parameterized algorithms, and there have been several deterministic and randomized approaches to these problems (see Table 10.1). One of the earliest approaches [42] used the color-coding technique [4]. Further, in [46], a kernel was developed, and the color-coding was combined with dynamic programming on the structure of the kernel to obtain an improvement. In [79], Koutis used an algebraic formulation of (r, k) -SP and proposed a randomized algorithm (derandomized with hash families). The randomized approaches saw further improvements in subsequent work [80, 82, 17], also based on algebraic techniques. The common theme in these developments

Reference	Weighted?	Algorithm	Running Time
Chen <i>et al.</i> [28]	No	D	$\mathcal{O}^*((rk)^{O(rk)})$
Downey <i>et al.</i> [42]	Yes	D	$\mathcal{O}^*((rk)^{O(rk)})$
Fellows <i>et al.</i> [46]	Yes	D	$\mathcal{O}^*(2^{O(rk)})$
Liu <i>et al.</i> [83]	Yes	D	$\mathcal{O}^*(12.8^{(r-1)k})$
Koutis [79]	No	D	$\mathcal{O}^*(2^{O(rk)})$
	No	R	$\mathcal{O}^*(10.874^{rk})$
Chen <i>et al.</i> [34]	No	D	$\mathcal{O}^*(5.44^{(r-1)k})$
Chen <i>et al.</i> [31]	Yes	D	$\mathcal{O}^*(4^{rk+o(rk)})$
	Yes	R	$\mathcal{O}^*(4^{(r-1)k+o(rk)})$
Chen <i>et al.</i> [27]	Yes	D	$\mathcal{O}^*(4^{(r-1)k+o(rk)})$
Koutis [80]	No	R	$\mathcal{O}^*(2^{rk})$
Koutis <i>et al.</i> [82]	No	R	$\mathcal{O}^*(2^{(r-1)k})$
Björklund <i>et al.</i> [17]	No	R	$\mathcal{O}^*(2^{(r-2)k})$
Goyal <i>et al.</i> [65]	Yes	D	$\mathcal{O}^*(2.851^{(r-1)k})$
Theorem 10.1	Yes	D	$\mathcal{O}^*(2.619^{(r-1)k})$

Table 10.1: Algorithms for (r, k) -DM, with D denoting deterministic algorithms and R denoting randomized algorithms.

is to express a parameterized problem in an algebraic framework by associating monomials with the combinatorial structures that are sought, ultimately arriving at a multilinear monomial testing problem or a polynomial identity testing problem. In a recent development [34], a derandomization method was proposed for these algebraic approaches, leading to deterministic algorithms that solve (r, k) -DM in time $\mathcal{O}^*(5.44^{(r-1)k})$.

Prior to our work, the running time of the best deterministic algorithm for (r, k) -DM was $\mathcal{O}^*(4^{(r-1)k+o(rk)})$ [27]. This algorithm, based on the randomized divide-and-conquer technique [31], also achieved the previous best running times for the weighted version of (r, k) -DM. Many algorithms have been designed for the special cases of $(3, k)$ -DM (see Table 10.2). In particular, the approach in [32] uses a clever combination of dynamic programming (embedded in a color coding framework) and iterative expansion, derandomized using hash families. However, the specialized algorithms achieve neither the previous best deterministic running times for $(3, k)$ -DM, nor the previous best running times for the weighted versions of these problems. These are achieved by the above mentioned algorithms of Chen *et al.* [27], which, in the special cases of $(3, k)$ -DM, run in times $\mathcal{O}^*(16^{k+o(k)})$.

In [65], we gave an algorithm for (r, k) -WDM running in time $\mathcal{O}^*(2.851^{(r-1)k})$ and an algorithm for $(3, k)$ -DM running in time $\mathcal{O}^*(8.042^k)$. In this thesis we give an algorithm for (r, k) -WDM running in time $\mathcal{O}^*(2.619^{(r-1)k})$ by incorporating the time-size trade off of representative family. This algorithm runs in time $\mathcal{O}^*(6.86^k)$ for $(3, k)$ -WDM.

Algorithm. We give a dynamic programming algorithm for the problem (r, k) -

Reference	Weighted?	Algorithm	Running Time
Wang <i>et al.</i> [117]	Yes	D	$\mathcal{O}^*(432.082^k)$
Chen <i>et al.</i> [32]	No	D	$\mathcal{O}^*(21.907^k)$
Liu <i>et al.</i> [84]	No	D	$\mathcal{O}^*(21.254^k)$
	No	R	$\mathcal{O}^*(12.488^k)$
Wang <i>et al.</i> [118]	No	D	$\mathcal{O}^*(43.615^k)$
Goyal <i>et al.</i> [65]	No	D	$\mathcal{O}^*(8.042^k)$
Theorem 10.1	Yes	D	$\mathcal{O}^*(6.86^k)$

Table 10.2: Algorithms for $(3, k)$ -DM, with D denoting deterministic algorithms and R denoting randomized algorithms.

WDM. Let $(U_1, \dots, U_r, \mathcal{F}, w : \mathcal{F} \rightarrow \mathbb{N}, k, W)$ be an instance of (r, k) -WDM. Recall that $\mathcal{F} \subseteq U_1 \times \dots \times U_r$, and the problem involves finding k mutually disjoint sets in \mathcal{F} whose weight is at least W . Before explaining the algorithm consider the following lemma regarding the computation of the representative families, which is useful for designing our algorithm.

Lemma 10.1. *Let k, p', q' be integers and let $p = p' + q'$. Let \mathcal{P} be a p' -family of size $y^{-p'}(1-y)^{-(k-p')}2^{o(k)}$ over a universe U of size n , where $y = \frac{p'}{2k-p'}$, and let $S \subseteq U$ of size q' . Let $w : \mathcal{P} \rightarrow \mathbb{N}$ be a non-negative weight function with maximum value at most W . Then we can compute $\widehat{\mathcal{P} \bullet \{S\}} \subseteq_{\maxrep}^{k-p'-q'} \mathcal{P} \bullet \{S\}$ of size*

$$\left(\frac{2k-p}{p}\right)^p \left(\frac{2k-p}{2(k-p)}\right)^{k-p} 2^{o(k)}$$

in time

$$\mathcal{O}\left(2^{o(k)} \log n \cdot \log W \cdot \max_{j \in [p]} \left\{ \left(\frac{2k-j}{j}\right)^j \left(\frac{2k-j}{2k-2j}\right)^{2k-2j} \right\}\right)$$

Proof. Let $S = \{s_1, \dots, s_{q'}\}$. Remove from \mathcal{P} the sets that are not disjoint from S . First, we compute $\mathcal{P}_1 = \widehat{\mathcal{P} \bullet \{s_1\}} \subseteq_{\maxrep}^{k-p'-1} \mathcal{P} \bullet \{s_1\}$; then, we compute $\mathcal{P}_2 = \widehat{\mathcal{P}_1 \bullet \{s_2\}} \subseteq_{\maxrep}^{k-p'-2} \mathcal{P}_1 \bullet \{s_2\}$, and so on. For all $j \in \{1, \dots, q'\}$, we compute $\mathcal{P}_j = \widehat{\mathcal{P}_{j-1} \bullet \{s_j\}} \subseteq_{\maxrep}^{k-p'-j} \mathcal{P}_{j-1} \bullet \{s_j\}$ by applying Theorem 7.3 with the value x_j for x , where

$$x_j = \frac{p' + j}{2k - (p' + j)}$$

We output $\mathcal{P}_{q'}$ as the $(k - p' - q')$ -representative family for $\mathcal{P} \bullet \{S\}$. Using induction on i , we prove that for $i \in \{0, \dots, q'\}$, $\mathcal{P}_i \subseteq_{\maxrep}^{k-p'-i} \mathcal{P} \bullet \{\{s_1, \dots, s_i\}\}$ where $\mathcal{P}_0 = \mathcal{P}$. The statement is trivially true for $i = 0$. Now, suppose the statement is true for all values of $j < i$. We need to show that $\mathcal{P}_i \subseteq_{\maxrep}^{k-p'-i} \mathcal{P} \bullet \{\{s_1, \dots, s_i\}\}$. Let $X \in \mathcal{P} \bullet \{\{s_1, \dots, s_i\}\}$, and $Y \subseteq U$ such that $|Y| = k - p' - i$ and $X \cap Y = \emptyset$. We need to show that there exists $X^* \in \mathcal{P}_i$ such that $X^* \cap Y = \emptyset$ and $w(X^* \setminus S) \geq w(X \setminus S)$. Consider the sets $X_i = X \setminus \{s_i\}$ and $Y_i = Y \cup \{s_i\}$. Note that $X_i \in \mathcal{P} \bullet \{\{s_1, \dots, s_{i-1}\}\}$ and $X_i \cap Y_i = \emptyset$. By induction hypothesis, $\mathcal{P}_{i-1} \subseteq_{\maxrep}^{k-p'-(i-1)} \mathcal{P} \bullet \{\{s_1, \dots, s_{i-1}\}\}$, and thus there exists $X_i^* \in \mathcal{P}_{i-1}$ such that $X_i^* \cap Y_i = \emptyset$ and $w(X_i^* \setminus S) \geq w(X_i \setminus S)$. Let $X' = X_i^* \cup \{s_i\}$. Note that $X' \in \mathcal{P}_{i-1} \bullet \{\{s_i\}\}$ and $X' \cap Y = \emptyset$. Hence, there exists $X^* \in \mathcal{P}_i = \widehat{\mathcal{P}_{i-1} \bullet \{\{s_i\}\}}$ as desired.

The cardinality of $(k - p' - q')$ -representative family $\mathcal{P}_{q'}$ of $\mathcal{P} \bullet \{S\}$ follows from Theorem 7.3. Let s_j be the size of the representative family $\mathcal{P}_j = \widehat{\mathcal{P}_{j-1} \bullet \{s_j\}}$ when we apply Theorem 7.3 with the value x_j . That is $s_j = (x_j)^{-p'-j} (1 - x_j)^{-k+p'+j} \cdot 2^{o(k)}$. By the assumption of the cardinality of \mathcal{P} , we have that $|\mathcal{P}| = s_0$. Now we have that $|\mathcal{P}_{j+1}| = \widehat{\mathcal{P}^{j-1} \bullet \{s_j\}} = s_j$. By Theorem 7.3, the running time to compute \mathcal{P}_{j+1} is bounded by,

$$s_j \cdot (1 - x_{j+1})^{-k+(j+1)} \cdot 2^{o(k)} \log n \cdot \log W \quad (10.1)$$

We need the following claim to analyse the running time and its proof is similar to the proof of Claim 8.2.

Claim 10.1. *For any $3 < p' + i < p$, $s_i \leq e^2 \cdot (p' + i + 1) \cdot s_{i+1}$.*

From Equation 10.1 and Claim 10.1 we have that the running time for computing

$\mathcal{P}_{q'}$ is bounded by

$$\begin{aligned} & \mathcal{O} \left(\sum_{i=1}^{q'} s_i \cdot (1 - x_i)^{-k+p'+i} \cdot 2^{o(k)} \cdot \log n \cdot \log W \right) \\ &= \mathcal{O} \left(2^{o(k)} \log n \cdot \log W \cdot \max_{j \in [p]} \left\{ \left(\frac{2k-j}{j} \right)^j \left(\frac{2k-j}{2k-2j} \right)^{2k-2j} \right\} \right) \end{aligned}$$

This completes the proof. □

Now we define some notation which is helpful to design the algorithm and its correctness. For a set $S \in \mathcal{F}$, we let $S[i]$ denote the element in $S \cap U_i$, that is, $S[i]$ is the element of S that is from U_i . We sometimes refer to the element $S[i]$ as the i^{th} coordinate of S . The dynamic programming approach involves iterating over the elements in U_r . To this end, let $U_r := \{c_1, c_2, \dots, c_n\}$ (for the discussion in this section, the index i of each element c_i is an arbitrary and fixed choice). Moreover, let $\mathcal{M} := \{S_1, S_2, \dots, S_t\}$ be an rD -Matching, and let $\mathcal{M}_r := \{S_i[r] \mid i \in [t]\} \subseteq U_r$ and $\mathcal{M}_{[r-1]} := \{S_i[j] \mid i \in [t], j \in [r-1]\}$. We define the *maximum last index* of \mathcal{M} , denoted by $\lambda(\mathcal{M})$, as the largest index i for which $c_i \in \mathcal{M}_r$. For the empty matching \emptyset , $\lambda(\emptyset) = 0$. In the i^{th} iteration of our algorithm, we would like to store representative sub family of all matchings whose maximum last index is at most i . We let $\mathcal{Q}^{(i)} := \{\mathcal{M} \mid \lambda(\mathcal{M}) \leq i\}$ and $\mathcal{Q}_j^{(i)} := \{\mathcal{M} \mid \mathcal{M} \in \mathcal{Q}^{(i)}, |\mathcal{M}| = j\}$. Notice that the $\mathcal{Q}_j^{(i)}$'s constitute a partition of the set $\mathcal{Q}^{(i)}$ based on matching size; in other words, $\mathcal{Q}^{(i)} := \bigcup_{j=0}^n \mathcal{Q}_j^{(i)}$. Let $\mathcal{X}_j^{(i)} := \{\mathcal{M}_{[r-1]} \mid \mathcal{M} \in \mathcal{Q}_j^{(i)}\}$. We assign a weight function $w' : \mathcal{X}_j^{(i)} \rightarrow \mathbb{N}$ as follows. For every $\mathcal{M}_{[r-1]} \in \mathcal{X}_j^{(i)}$, $w'(\mathcal{M}_{[r-1]}) = \max\{w(\mathcal{M}^*) \mid \mathcal{M}^* \in \mathcal{Q}_j^{(i)}, \mathcal{M}_{[r-1]} = \mathcal{M}_{[r-1]}^*\}$.

We are now ready to describe our algorithm, Algorithm 1, whose outline is given below. The heart of Algorithm 1 consists of two modules – **Compute Partial Solutions** $\mathcal{L}^{(i)}$ and **Prune Partial Solutions** $\mathcal{L}^{(i)}$. In $\mathcal{L}^{(i)}$, we store carefully chosen rD -Matchings whose maximum last index is at most i . To use the representative families more efficiently, we would like to split the elements of the matchings into two parts. We will first collect the parts of the matching that come from $(\bigcup_{i=1}^{r-1} U_i)$, and then store separately a map that completes the first part to the complete matching. This will allow us to apply the dynamic programming approach. let

$$\gamma_j := \{(\mathcal{M}, \mathcal{M}_{[r-1]}) \mid \mathcal{M} \text{ is a matching of size } j \text{ in } \mathcal{F}\}.$$

In γ_j , we are merely storing the associations of $\mathcal{M}_{[r-1]}$ with the matchings that they “came from”. Observe that γ_j might (by definition) contain multiple entries with the same second coordinate. On the other hand, when the algorithm stores the associations in γ_j , we will see that it is enough to maintain one maximum weighted

Algorithm 1: Algorithm for (r, k) -WDM

```
1  $\mathcal{L}^{(0)} \leftarrow \{\emptyset\}$ 
2 for  $i \in \{1, 2, \dots, n\}$  do
3   Compute Partial solutions  $\mathcal{L}^{(i)}$ 
4   Prune Partial Solutions  $\mathcal{L}^{(i)}$ 
5 if  $\exists \mathcal{M} \in \mathcal{L}^{(n)}$  such that  $|\mathcal{M}| = k$  and  $w(\mathcal{M}) \geq W$  then
6   return  $\mathcal{M}$ .
7 else
8   returnNo
```

Algorithm 2: Compute Partial Solutions $\mathcal{L}^{(i)}$

```
1  $\mathcal{L}_j^{(i-1)} \leftarrow \{\mathcal{M} \mid \mathcal{M} \in \mathcal{L}^{(i-1)}, |\mathcal{M}| = j\}$ 
2  $\mathcal{R}_j^{(i-1)} \leftarrow \{\mathcal{M}_{[r-1]} \mid \mathcal{M} \in \mathcal{L}_j^{(i-1)}\}$ 
3  $\gamma_j := \{(\mathcal{M}, \mathcal{M}_{[r-1]}) \mid \mathcal{M} \text{ is a matching in } \mathcal{L}_j^{(i-1)}\}$ 
4 for  $j \in \{1, \dots, k\}$ ,  $S \in \mathcal{F}$  such that  $S[r] = c_i$  do
5   Compute  $\mathcal{R}_j(S) \subseteq_{\text{maxrep}}^{(r-1)(k-j)} \mathcal{R}_{j-1}^{(i-1)} \bullet \{S_{[r-1]}\}$  using Lemma 10.1
6  $\mathcal{L}^{(i)} \leftarrow \bigcup_{j=1}^k (\{\tilde{\gamma}_j(\mathcal{M}_{[r-1]}) \cup S \mid \mathcal{M}_{[r-1]} \cup S_{[r-1]} \in \mathcal{R}_j(S), S[r] = c_i\}) \cup \mathcal{L}^{(i-1)}$ 
```

entry for each $\mathcal{M}_{[r-1]}$. To this end, we define the function $\tilde{\gamma}_j$ as follows. Let \preceq be an arbitrary total order on the set of all matchings in \mathcal{F} . For a set $S \subseteq \bigcup_{i=1}^{r-1} U_i$, we define $\tilde{\gamma}(S)$ as the smallest matching \mathcal{M} (with respect to \preceq) among the maximum weighted matchings in the set $\{\mathcal{M}' \mid (\mathcal{M}', S) \in \gamma_j\}$. If $\tilde{\gamma}_j(S)$ is \mathcal{M} , then we say that \mathcal{M} is the matching associated with S . Finally, for a set S and a matching \mathcal{M} , we abuse notation and say that \mathcal{M} is disjoint from S (notationally, $\mathcal{M} \cap S = \emptyset$) to mean that $T \cap S = \emptyset$ for all $T \in \mathcal{M}$.

The module **Compute Partial Solutions** $\mathcal{L}^{(i)}$, whose pseudocode is given in Algorithm 2, computes subfamily of matchings of size i from $\mathcal{L}^{(i-1)}$ and \mathcal{F} using Lemma 10.1 to perform an initial pruning of $\mathcal{L}^{(i)}$ (while computing it from $\mathcal{L}^{(i-1)}$ and \mathcal{F}). In Algorithm 2, the collection of sets corresponding to matchings in $\mathcal{L}_j^{(i-1)}$ are stored in $\mathcal{R}_j^{(i-1)}$. By computing $\mathcal{R}_j(S) \subseteq_{\text{maxrep}}^{(r-1)(k-j)} \mathcal{R}_{j-1}^{(i-1)} \bullet \{S_{[r-1]}\}$ for any set S such that $S[r] = c_i$, we are computing a representative family of $\mathcal{L}_j^{(i-1)} \bullet S$, which can be recovered from $\mathcal{R}_j(S)$ using the map $\tilde{\gamma}_j$. for $\mathcal{X}_j^{(i-1)} \bullet \{S\}$; then, in the last step, the matchings are recovered from $\mathcal{R}_j(S)$ via the map $\tilde{\gamma}_j$.

The pseudocode of the second module, **Prune Partial Solutions** $\mathcal{L}^{(i)}$, is given in Algorithm 3. In this module, we compute a representative family for $\mathcal{L}^{(i)}$. We first partition the set $\mathcal{L}^{(i)}$ — the part denoted by $\mathcal{L}_j^{(i)}$ contains all matchings from $\mathcal{L}^{(i)}$ of size j . Note that this is simply done to ensure uniformity of size. Next, we associate with every matching $\mathcal{M} \in \mathcal{L}_j^{(i)}$, a set that consists of the first $(r-1)$

Algorithm 3: Prune Partial Solutions $\mathcal{L}^{(i)}$

```

1  $\mathcal{R}_0^{(0)} \leftarrow \{\emptyset\}$ 
2  $\mathcal{R}_j^{(0)} \leftarrow \emptyset$ , for all  $1 \leq j \leq k$ 
3  $\mathcal{L}_j^{(i)} \leftarrow \{\mathcal{M} \mid \mathcal{M} \in \mathcal{L}^{(i)}, |\mathcal{M}| = j\}$ 
4  $\mathcal{P}_j^{(i)} \leftarrow \{\mathcal{M}_{[r-1]} \mid \mathcal{M} \in \mathcal{L}_j^{(i)}\}$ 
5  $\gamma_j := \{(\mathcal{M}, \mathcal{M}_{[r-1]}) \mid \mathcal{M} \text{ is a matching in } \mathcal{L}_j^{(i)}\}$ 
6 for  $j \in \{0, 1, \dots, k\}$  do
7    $w' \leftarrow \{(\mathcal{M}_{[r-1]}, w(\tilde{\gamma}_j(\mathcal{M}_{[r-1]}))) \mid \mathcal{M}_{[r-1]} \in \mathcal{P}_j^{(i)}\}$ 
8   Compute  $\mathcal{R}_j^{(i)} \subseteq_{\maxrep}^{(r-1)(k-j)} \mathcal{P}_j^{(i)}$  using Theorem 7.3 with  $x = \frac{(r-1)j}{2(r-1)k - (r-1)j}$ 
9    $\mathcal{L}_j^{(i)} \leftarrow \{\tilde{\gamma}_j(\mathcal{M}_{[r-1]}) \mid \mathcal{M}_{[r-1]} \in \mathcal{R}_j^{(i)}\}$ 
10  $\mathcal{L}^{(i)} \leftarrow \bigcup_{j=0}^k \mathcal{L}_j^{(i)}$ 

```

indices of every set in \mathcal{M} . Recall that this is denoted by $\mathcal{M}_{[r-1]}$. The collection of sets that correspond to matchings in $\mathcal{L}_j^{(i)}$ is denoted by $\mathcal{P}_j^{(i)}$. We use γ_j to store the associations between the sets and the original matchings. Note that γ_j might have multiple pairs with the same second index and the same weight w for the first index, but this will be irrelevant (it would simply mean that $\mathcal{M}_{[r-1]}$ can be “pulled back” to multiple matchings, each of which would be equally valid). We then define a weight function $w' : \mathcal{P}_j^{(i)} \rightarrow \mathbb{N}$. For all $\mathcal{M}_{[r-1]} \in \mathcal{P}_j^{(i)}$, we set $w'(\mathcal{M}_{[r-1]}) = w(\tilde{\gamma}(\mathcal{M}_{[r-1]}))$. Now, the central step of this module is to compute a max $(r-1)(k-j)$ -representative family $\mathcal{R}_j^{(i)}$ for $\mathcal{P}_j^{(i)}$. Once we have the representative family, we revise $\mathcal{L}_j^{(i)}$ to only include the matchings associated with the sets in $\mathcal{R}_j^{(i)}$.

The correctness of the algorithm relies on the fact that at each step, instead of the complete family of partial solutions $\mathcal{Q}^{(i)}$, it suffices to store only a representative family for $\mathcal{Q}^{(i)}$. Also, we will show that the family computed by the algorithm, $\mathcal{L}^{(i)}$, is indeed a representative family for $\mathcal{Q}^{(i)}$. We next prove the following lemma.

Lemma 10.2. *For all $0 \leq i \leq n$, and $0 \leq j \leq k-1$, the set $\mathcal{P}_j^{(i)}$ is a max $((r-1)k - (r-1)j)$ -representative family for $\mathcal{X}_j^{(i)}$, where $\mathcal{P}_j^{(0)} = \mathcal{R}_j^{(0)}$*

Proof. To prove this lemma, we need to show that for all $Y \subseteq U_1 \cup U_2 \cup \dots \cup U_{r-1}$ such that $|Y| \leq (r-1)(k-j)$, if there exists a set $Z \in \mathcal{X}_j^{(i)}$ such that $Y \cap Z = \emptyset$, then there also exists $Z^* \in \mathcal{P}_j^{(i)}$ such that $Z^* \cap Y = \emptyset$ and $w'(Z^*) \geq w'(Z)$. Recall that in this situation, we say that Z^* is a max $(r-1)(k-j)$ -representative for Z with respect to Y . The proof is by induction on i . The base case is when $i = 0$.

Observe that:

$$\mathcal{P}_j^{(0)} = \mathcal{X}_j^{(0)} = \begin{cases} \{\emptyset\} & \text{if } j = 0 \\ \emptyset & \text{Otherwise.} \end{cases}$$

Hence $\mathcal{P}_j^{(0)}$ is an $(r-1)(k-j)$ -representative family for $\mathcal{X}_j^{(0)}$ for all $0 \leq j \leq k-1$. The induction hypothesis states that $\mathcal{P}_j^{(i-1)}$ is a max $(r-1)(k-j)$ -representative family for $\mathcal{X}_j^{(i-1)}$, for all $0 \leq j \leq k-1$. We will now show that $\mathcal{P}_j^{(i)}$ is a max $(r-1)(k-j)$ -representative family for $\mathcal{X}_j^{(i)}$, for all $0 \leq j \leq k-1$. Note that the case when $j = 0$ is easily handled, since $\emptyset \in \mathcal{P}_0^{(i)}$. For the rest of this proof we assume that $j \in \{1, \dots, k-1\}$. Let $Y \subseteq U_1 \cup U_2 \cup \dots \cup U_{r-1}$ such that $|Y| \leq (r-1)(k-j)$, and suppose there exists a set $Z \in \mathcal{X}_j^{(i)}$ such that $Z \cap Y = \emptyset$. Let \mathcal{M}^Z be the matching associated with Z and hence $w'(Z) = w(\mathcal{M}^Z)$. Since \mathcal{M}^Z is derived from an element of $\mathcal{X}_j^{(i)}$, note that $\lambda(\mathcal{M}^Z) \leq i$. We distinguish two cases, depending on whether \mathcal{M}^Z contains a set with c_i as the r^{th} coordinate.

Case 1. \mathcal{M}^Z contains a set with c_i as the last coordinate.

Let $S \in \mathcal{M}^Z$ be such that $c_i \in S$. Define the smaller matching $\mathcal{M}^{Z \setminus S} := \mathcal{M}^Z \setminus S$. Note that $|\mathcal{M}^{Z \setminus S}| = j-1$ and $\lambda(\mathcal{M}^{Z \setminus S}) \leq i-1$. Hence, $\mathcal{M}^{Z \setminus S} \in \mathcal{Q}_{j-1}^{(i-1)}$ and $\mathcal{M}_{[r-1]}^{Z \setminus S} \in \mathcal{X}_{j-1}^{(i-1)}$. Let $A = \mathcal{M}_{[r-1]}^{Z \setminus S}$. By definition of w' , $w'(A) \geq w(\mathcal{M}^{Z \setminus S})$. Now consider the set $Y^S = Y \cup S_{[r-1]}$. Note that $|Y^S| \leq (r-1)(k-j+1)$, since $|Y| \leq (r-1)(k-j)$. It is also easy to check that $A \cap Y^S = \emptyset$. By the induction hypothesis, we have that $\mathcal{P}_{j-1}^{(i-1)}$ contains an $(r-1)(k-j+1)$ -representative of A with respect to Y^S . Let us denote this representative by B . Note that $B \cap Y^S = \emptyset$ and $w'(B) \geq w'(A)$ by definition. Since $\mathcal{R}_{j-1}^{(i-1)}$ is an $(r-1)(k-j+1)$ -representative family for $\mathcal{P}_{j-1}^{(i-1)}$, there exists $C \in \mathcal{R}_{j-1}^{(i-1)}$ such that $C \cap Y^S = \emptyset$ and $w'(C) \geq w'(B) \geq w'(A)$. Since $C \cap S_{[r-1]} = \emptyset$, $C \cup S_{[r-1]} \in \mathcal{R}_{j-1}^{(i-1)} \bullet S_{[r-1]}$. Note that $C \cup S_{[r-1]}$ is disjoint from Y . Since $\mathcal{R}_j(S) \subseteq_{\text{maxrep}}^{(r-1)(k-j)} \mathcal{R}_{j-1}^{(i-1)} \bullet \{S_{[r-1]}\}$, there exists $D \cup S_{[r-1]} \in \mathcal{R}_j(S)$ such that $D \cup S_{[r-1]}$ is disjoint from Y and $w'(D) \geq w'(C) \geq w'(A)$. Let $\mathcal{M}^{D'}$ be the matching associated with $D \cup S_{[r-1]}$. Since $D \cup S_{[r-1]} \in \mathcal{R}_j(S)$, we have that $\mathcal{M}^{D'} \in \mathcal{L}_j^{(i)}$, and thus $D \cup S_{[r-1]} \in \mathcal{P}_j^{(i)}$. Hence $D \cup S_{[r-1]}$ is the required $(r-1)(k-j)$ -representative

with respect to Y .

Case 2. \mathcal{M}^Z contains no set with c_i as the last coordinate.

In this case, we have that $\lambda(\mathcal{M}^Z) \leq i - 1$. Clearly, \mathcal{M}^Z is contained in $\mathcal{Q}_j^{(i-1)}$ and consequently, $Z = \mathcal{M}_{[r-1]}^Z \in \mathcal{X}_j^{(i-1)}$. By the induction hypothesis, let B be the max $(r - 1)(k - j)$ -representative for Z in $\mathcal{P}_j^{(i-1)}$ with respect to Y . So $B \cap Y = \emptyset$ and $w'(B) \geq w'(\mathcal{M}_{[r-1]}^Z) \geq w(\mathcal{M}^Z) = w'(Z)$. Since $B \in \mathcal{P}_j^{(i-1)}$, by the definition of a representative set and Step 15, there exists $C \in \mathcal{R}_j^{(i-1)}$ such that $C \cap Y = \emptyset$ and $w'(C) \geq w'(B) \geq w'(Z)$. Since $C \in \mathcal{R}_j^{(i-1)}$, the matching associate with C , say \mathcal{M}^C , belongs to $\mathcal{L}_j^{(i-1)}$, and $w(\mathcal{M}^C) = w'(C)$. Further, since $\mathcal{L}_j^{(i)} \supseteq \mathcal{L}_j^{(i-1)}$, we have that \mathcal{M}^C is also present in $\mathcal{L}_j^{(i)}$. This implies that $C = \mathcal{M}_{[r-1]}^C \in \mathcal{P}_j^{(i)}$. Hence C is the required $(r - 1)(k - j)$ -representative with respect to Y . \square

Now, by the transitivity of representative family (Lemma 7.1), we get the following lemma.

Lemma 10.3. *For all $0 \leq i \leq n$, and $0 \leq j \leq k - 1$, the set $\mathcal{R}_j^{(i)}$ is a max $((r - 1)k - (r - 1)j)$ -representative family for $\mathcal{X}_j^{(i)}$.*

Observe that any solution constructed by Algorithm 1 is always a valid matching. Therefore, if there is no matching of size k , Algorithm 1 always returns NO. On the other hand, if given a YES-instance, we now show that Algorithm 1 always finds a rD -Matching of size k with weight at least W .

Lemma 10.4. *Let $(U_1, \dots, U_r, \mathcal{F}, w : \mathcal{F} \rightarrow \mathbb{N}, k, W)$ be a YES-instance of (r, k) -WDM. Then, Algorithm 1 successfully computes a rD -Matching of size k with weight at least W .*

Proof. Let $(U_1, \dots, U_r, \mathcal{F}, w : \mathcal{F} \rightarrow \mathbb{N}, k, W)$ be a YES-instance of (r, k) -WDM. Let \mathcal{M} be a k -sized rD -Matching in \mathcal{F} such that $w(\mathcal{M}) \geq W$. Recall that $\mathcal{Q}_j^{(i)}$ is the set of all rD -Matchings of size j with maximum last index at most i , and $\mathcal{X}_j^{(i)}$ contains the projections of these matchings on their first $(r - 1)$ coordinates. Therefore, $\mathcal{M} \in \mathcal{Q}_k^{(n)}$ and $\mathcal{M}_{[r-1]} \in \mathcal{X}_k^{(n)}$. By Lemma 10.2, we have that $\mathcal{R}_k^{(n)}$ is a max 0-representative family for $\mathcal{X}_k^{(n)}$. Since $\mathcal{R}_k^{(n)} \subseteq_{maxrep}^0 \mathcal{X}_k^{(n)}$, we have that $\mathcal{R}_k^{(n)}$

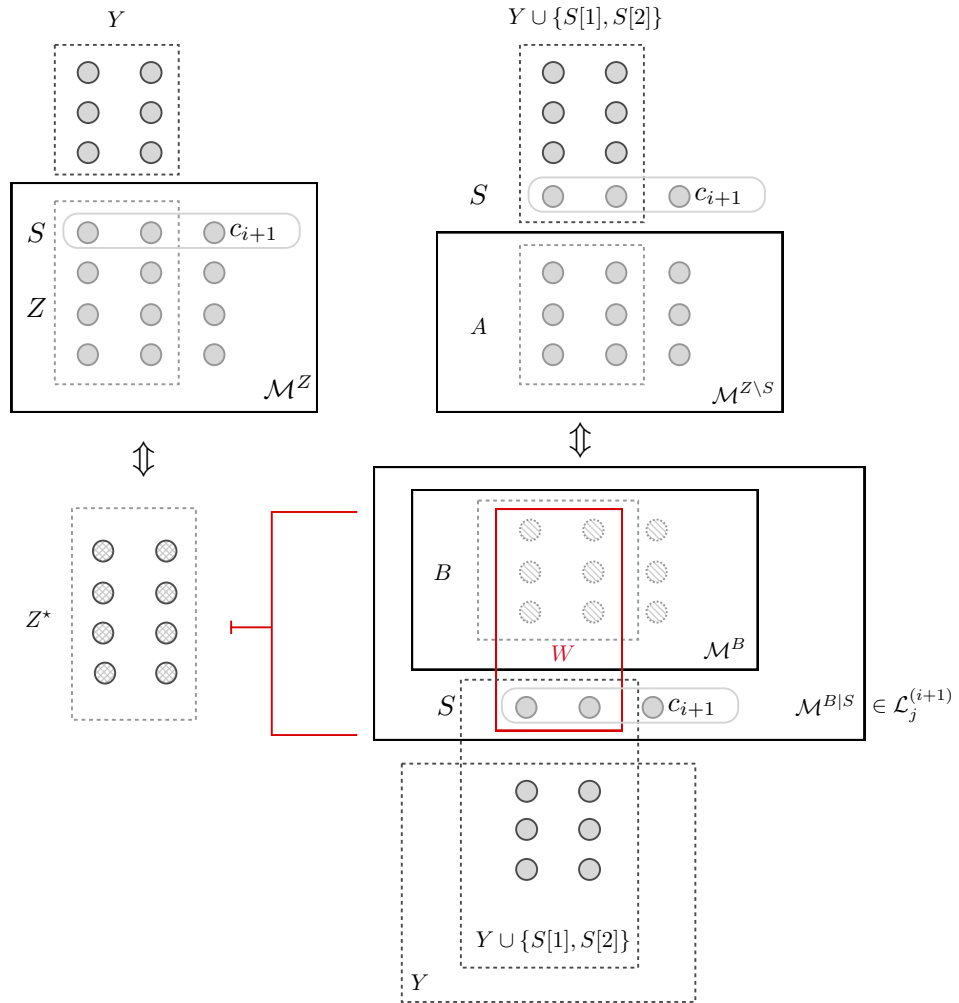


Figure 10.1: A schematic view of the proof of Lemma 10.2 when $r = 3$. It is meant to be read in clockwise order. Note that B is a representative for A with respect to $Y \cup \{S[1], S[2]\}$, and Z^* is a representative for $B \cup S$ with respect to Y . This eventually implies that Z^* is a representative for Z with respect to Y .

contains a 0-representative Z^* for $\mathcal{M}_{[r-1]}$ with respect to \emptyset . So we have $w'(Z^*) \geq w'(\mathcal{M}_{[r-1]})$. Let \mathcal{M}^* be the matching associated with Z^* . Therefore $w(\mathcal{M}^*) = w'(Z^*) \geq w'(\mathcal{M}_{[r-1]}) = w(\mathcal{M})$. Therefore, in Step 10 of Algorithm 3, we have that $\mathcal{L}^{(n)}$ contains the matching \mathcal{M}^* , which is then returned as output in Step 6 of Algorithm 1. \square

Lemma 10.5. *The running time of Algorithm 1 is bounded by $\mathcal{O}(2.619^{(r-1)k} |\mathcal{F}| \cdot n \log(n + W))$.*

Proof. Let $1 \leq i \leq n$ be and $1 \leq j \leq k$ be fixed. First, consider the running time of Step 5 in Algorithm 2. Since $\mathcal{R}_{j-1}^{(i-1)}$ is derived from $\mathcal{L}_{j-1}^{(i-1)}$ in the previous iteration of the module Algorithm 3, by Theorem 7.3, we have that

$$|\mathcal{R}_{j-1}^{(i-1)}| = \left(\frac{(r-1)(2k-j')}{(r-1)j'} \right)^{(r-1)j'} \left(\frac{(r-1)(2k-j')}{2(r-1)(k-j')} \right)^{(r-1)(k-j')} 2^{o(rk)}$$

where $j' = j-1$. Note that the cardinality of each set in $\mathcal{R}_{j-1}^{(i-1)}$ is $(r-1)(j-1)$. The cardinality of $\mathcal{R}_{j-1}^{(i-1)}$ satisfy the premise of Lemma 10.1. Let $k' = (r-1)k$. Thus, due to Lemma 10.1, the running time of Step 5 in Algorithm 2 is bounded by

$$\mathcal{O} \left(2^{o(rk)} \log(n + W) \cdot \max_{j'' \in [(r-1)j]} \left\{ \left(\frac{2k' - j''}{j''} \right)^{j''} \left(\frac{2k' - j''}{2k' - 2j''} \right)^{2k' - 2j''} \right\} \right)$$

Now we consider the running time of Step 8 of Algorithm 3 for the fixed i and j . Let $x_j = \frac{(r-1)j}{2(r-1)k - (r-1)j} = \frac{j}{2k-j}$ and $s_j = x_j^{-(r-1)j} (1 - x_j)^{-(r-1)(k-j)} 2^{o(k)}$ for all j . Since the construction of $\mathcal{R}_j(S)$ in iteration i uses Lemma 10.1, the size of $|\mathcal{R}_j(S)|$ is bounded by s_j . Since $\mathcal{L}_j^{(i-1)}$ is a pruned sub family of matching obtained by applying Theorem 7.3 in Step 8 of Algorithm 3 in the iteration $i-1$, $|\mathcal{L}_j^{(i-1)}|$ is bounded by s_j . This implies that the cardinality of \mathcal{P}_j in iteration i is bounded by $\mathcal{O}(|\mathcal{F}|s_j)$. Thus by Theorem 7.3, the running time of Step 8 of Algorithm 3 in the iteration i

is bounded by,

$$\mathcal{O}(s_j \cdot (1-x_j)^{-(r-1)(k-j)} \cdot \log(n+W)) = \mathcal{O}(x_j^{-(r-1)j} (1-x_j)^{-2(r-1)(k-j)} 2^{o(k)} \cdot \log(n+W)).$$

Thus the total running time of Algorithm 1 is bounded by

$$\mathcal{O} \left(2^{o(rk)} |\mathcal{F}| \cdot n \log(n+W) \max_{j'' \in [k']} \left\{ \left(\frac{2k' - j''}{j''} \right)^{j''} \left(\frac{2k' - j''}{2k' - 2j''} \right)^{2k' - 2j''} \right\} \right).$$

The above running time is maximized when $j'' = \left(1 - \frac{1}{\sqrt{5}}\right) k'$. Thus the total running time is bounded by $\mathcal{O}(2.619^{(r-1)k} |\mathcal{F}| \cdot n \log(n+W))$. \square

Thus we have the following theorem.

Theorem 10.1. *(r, k) -WDM can be solved in deterministic time $\mathcal{O}(2.619^{(r-1)k} |\mathcal{F}| \cdot n \log(n+W))$.*

Chapter 11

Representative Family

computation for product family

We have seen many dynamic programming algorithms using fast computation of representative families. It is therefore very tempting to ask whether it is possible to compute representative families faster for families that arise naturally in dynamic programs, than for general families. A class of families which often arises in dynamic programs is the class of *product* families. A family \mathcal{F} is the *product* of \mathcal{A} and \mathcal{B} if $\mathcal{F} = \{A \cup B : A \in \mathcal{A}, B \in \mathcal{B} \wedge A \cap B = \emptyset\}$, that is, $\mathcal{F} = \mathcal{A} \bullet \mathcal{B}$. Product families naturally appear in dynamic programs where sets represent partial solutions and two partial solutions can be combined if they are disjoint. For an example, in the k -PATH problem partial solutions are vertex sets of paths starting at a particular root vertex v , and two such paths may be combined to a longer path if and only if they are disjoint (except for overlapping at v). Many other examples exist—essentially product families can be thought of as a *subset convolution* [10, 11], and the wide applicability of the fast subset convolution technique of Bjorklund et al [16] is largely due to the frequent demand to compute product families in dynamic programs.

In this chapter we give an algorithm for the computation of representative family for a product family in set systems. We give an algorithm which given an integer q and families \mathcal{A}, \mathcal{B} of sets of sizes p_1 and p_2 over a universe of size n , computes a q -representative family \mathcal{F}' of \mathcal{F} . The running time of our algorithm is *sublinear* in $|\mathcal{F}|$ for many choices of \mathcal{A}, \mathcal{B} and q which occur naturally in several dynamic programming algorithms. For example, let q, p_1, p_2 be integers. Let $k = q + p_1 + p_2$ and suppose that we have families \mathcal{A} and \mathcal{B} , which are $(k - p_1)$ and $(k - p_2)$ -representative families. Then the sizes of these families are roughly $|\mathcal{A}| = \binom{k}{p_1}$ and $|\mathcal{B}| = \binom{k}{p_2}$. In particular, when $p_1 = p_2 = \lceil k/2 \rceil$ both families are of size roughly 2^k , and thus the cardinality of \mathcal{F} is approximately 4^k . On the other hand, for any choice of p_1, p_2 , and k , our algorithm outputs a $(k - p_1 - p_2)$ -representative family

of \mathcal{F} of size roughly $\binom{k}{p_1+p_2}$ in time $3.8408^k n^{\mathcal{O}(1)}$. For many choices of p_1 , p_2 and q our algorithm runs significantly faster than $3.8408^k n^{\mathcal{O}(1)}$. The expression capturing the running time dependence on p_1 , p_2 and q can be found in Theorem 11.1 and Corollary 11.1. This algorithm considerably outperforms the naive approach where one first computes \mathcal{F} from \mathcal{A} and \mathcal{B} , and then computes a q -representative family \mathcal{F}' from \mathcal{F} . Following is the main theorem in this chapter.

Theorem 11.1. *Let \mathcal{L}_1 be a p_1 -family of sets and \mathcal{L}_2 be a p_2 -family of sets over a universe U of size n . Let $w : 2^U \rightarrow \mathbb{N}$ be an additive weight function. Let $\mathcal{L} = \mathcal{L}_1 \bullet \mathcal{L}_2$ and $p = p_1 + p_2$. For any $0 < x_1, x_2 < 1$, there exist $\widehat{\mathcal{L}} \subseteq_{\minrep}^{k-p_1-p_2} \mathcal{L}$ of size $x_1^{-p}(1-x_1)^{-(k-p)} \cdot 2^{\mathcal{O}(k)} \cdot \log n$ and it can be computed in time*

$$\mathcal{O} \left(\frac{z(n, k, W)}{x_1^p(1-x_1)^q} + \frac{z(n, k, W)}{x_2^{p_1}(1-x_2)^{p_2}} + \frac{|\mathcal{L}_1| \cdot z(n, k, W)}{x_1^{p_2}(1-x_1)^q(1-x_2)^{p_2}} + \frac{|\mathcal{L}_2| \cdot z(n, k, W)}{x_1^{p_1}(1-x_1)^q x_2^{p_1}} \right),$$

where $z(n, k, W) = 2^{\mathcal{O}(k)} n \log n \cdot \log W$ and W is the maximum weight defined by w .

Proof. We set $p = p_1 + p_2$ and $q = k - p$. To obtain the desired construction we first define an auxiliary graph and then use it to obtain the q -representative for the product family \mathcal{L} . Recall the definition separating collections from Chapter 7 We first obtain two families of separating collections.

- Apply Lemma 7.4 for $0 < x_1 < 1$ and construct a n - p - q -separating collection $(\mathcal{F}, \chi_{\mathcal{F}}, \chi'_{\mathcal{F}})$ of size $2^{\mathcal{O}(\frac{p+q}{\log \log(p+q)})} \cdot \frac{1}{x_1^p(1-x_1)^q} \cdot (p+q)^{\mathcal{O}(1)} \log n$ in time linear in the size of \mathcal{F} .
- Apply Lemma 7.4 for $0 < x_2 < 1$ and construct a n - p_1 - p_2 -separating collection $(\mathcal{H}, \chi_{\mathcal{H}}, \chi'_{\mathcal{H}})$ of size $2^{\mathcal{O}(\frac{p_1+p_2}{\log \log(p_1+p_2)})} \cdot \frac{1}{x_2^{p_1}(1-x_2)^{p_2}} \cdot (p_1+p_2)^{\mathcal{O}(1)} \log n$ in time linear in the size of \mathcal{H} .

Now we construct a graph $G = (V, E)$ where the vertex set V contains a vertex each for sets in $\mathcal{F} \uplus \mathcal{H} \uplus \mathcal{L}_1 \uplus \mathcal{L}_2$. For clarity of presentation we name the vertices by the corresponding set. Thus, the vertex set $V = \mathcal{F} \uplus \mathcal{H} \uplus \mathcal{L}_1 \uplus \mathcal{L}_2$. The edge set $E = E_1 \uplus E_2 \uplus E_3 \uplus E_4$, where each E_i for $i \in \{1, 2, 3, 4\}$ is defined as follows (see

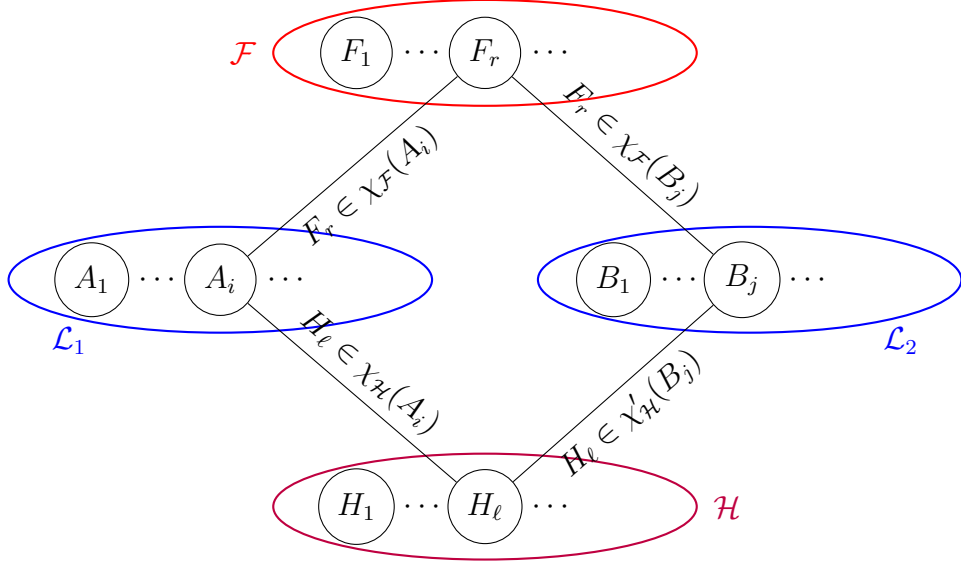


Figure 11.1: Graph constructed from $\mathcal{L}_1, \mathcal{L}_2, \mathcal{F}$ and \mathcal{H}

Figure 11.1).

$$\begin{aligned}
 E_1 &= \left\{ (A, F) \mid A \in \mathcal{L}_1, F \in \chi_{\mathcal{F}}(A) \right\} \\
 E_2 &= \left\{ (B, F) \mid B \in \mathcal{L}_2, F \in \chi_{\mathcal{F}}(B) \right\} \\
 E_3 &= \left\{ (A, H) \mid A \in \mathcal{L}_1, H \in \chi_{\mathcal{H}}(A) \right\} \\
 E_4 &= \left\{ (B, H) \mid B \in \mathcal{L}_2, H \in \chi'_{\mathcal{H}}(B) \right\}
 \end{aligned}$$

Thus G is essentially a 4-partite graph.

Algorithm. The construction of $\widehat{\mathcal{L}}$ is as follows. For a set $F \in \mathcal{F}$, we call a pair of sets (A, B) *cyclic*, if $A \in \mathcal{L}_1$, $B \in \mathcal{L}_2$ and there exists $H \in \mathcal{H}$ such that $FAHB$ forms a cycle of length four in G . Let $\mathcal{J}(F)$ denote the family of cyclic pairs for a set $F \in \mathcal{F}$ and

$$w_F = \min_{(A,B) \in \mathcal{J}(F)} w(A) + w(B).$$

We obtain the family $\widehat{\mathcal{L}}$ by adding $A \cup B$ for every set $F \in \mathcal{F}$ such that $(A, B) \in \mathcal{J}(F)$ and $w(A) + w(B) = w_F$. Indeed, if the family $\mathcal{J}(F)$ is empty then we do not add

any set to $\widehat{\mathcal{L}}$ corresponding to F . The procedure to find the smallest weight $A \cup B$ for any F is as follows. We first mark the vertices of $N_G(F)$ (the neighbors of F). Now we mark the neighbors of $\mathcal{P} = (N_G(F) \cap \mathcal{L}_1)$ in \mathcal{H} . For every marked vertex $H \in \mathcal{H}$, we associate a set A of minimum weight such that $A \in (\mathcal{P} \cap N_G(H))$. This can be done sequentially as follows. Let $\mathcal{P} = \{S_1, \dots, S_\ell\}$. Now iteratively visit the neighbors of S_i in \mathcal{H} , $i \in [\ell]$, and for each vertex of \mathcal{H} store the smallest weight vertex $S \in \mathcal{P}$ it has seen so far. After this we have a marked set of vertices in \mathcal{H} such that with each marked vertex H in \mathcal{H} we stored a smallest weight marked vertex in \mathcal{L}_1 which is a neighbor of H . Now for each marked vertex B in \mathcal{L}_2 , we go through the neighbors of B in the marked set of vertices in \mathcal{H} and associate (if possible) a second vertex (which is a minimum weighted marked neighbor from \mathcal{L}_2) with each marked vertex in \mathcal{H} . We obtain a pair of sets $(A, B) \in \mathcal{J}(F)$ such that $w(A) + w(B) = w_F$. This can be easily done by keeping a variable that stores a minimum weighted $A \cup B$ seen after every step of marking procedure. Since for each $F \in \mathcal{F}$ we add at most one set to $\widehat{\mathcal{L}}$, the size of $\widehat{\mathcal{L}}$ follows.

Correctness. We first show that $\widehat{\mathcal{L}} \subseteq \mathcal{L}$. Towards this we only need to show that for every $A \cup B \in \widehat{\mathcal{L}}$ we have that $A \cap B = \emptyset$. Observe that if $A \cup B \in \widehat{\mathcal{L}}$ then there exists a $F \in \mathcal{F}$, $H \in \mathcal{H}$ such that $FAHB$ forms a cycle of length four in the graph G . So $H \in \chi_{\mathcal{H}}(A)$ and $H \in \chi'_{\mathcal{H}}(B)$. This means $A \subseteq H$ and $B \cap H = \emptyset$. So we conclude A and B are disjoint and hence $\widehat{\mathcal{L}} \subseteq \mathcal{L}$. We also need to show that if there exist pairwise disjoint sets $A \in \mathcal{L}_1, B \in \mathcal{L}_2, C \in \binom{U}{q}$, then there exist $\widehat{A} \in \mathcal{L}_1, \widehat{B} \in \mathcal{L}_2$ such that $\widehat{A} \cup \widehat{B} \in \widehat{\mathcal{L}}$, $\widehat{A}, \widehat{B}, C$ are pairwise disjoint and $w(\widehat{A}) + w(\widehat{B}) \leq w(A) + w(B)$. By the property of separating collections $(\mathcal{F}, \chi_{\mathcal{F}}, \chi'_{\mathcal{F}})$ and $(\mathcal{H}, \chi_{\mathcal{H}}, \chi'_{\mathcal{H}})$, we know that there exists $F \in \chi_{\mathcal{F}}(A) \cap \chi_{\mathcal{F}}(B) \cap \chi'_{\mathcal{F}}(C)$, $H \in \chi_{\mathcal{H}}(A) \cap \chi'_{\mathcal{H}}(B)$. This implies that $FAHB$ forms a cycle of length four in the graph G . Hence in the construction of $\widehat{\mathcal{L}}$, we should have chosen $\widehat{A} \in \mathcal{L}_1$ and $\widehat{B} \in \mathcal{L}_2$ corresponding to F such that $w(\widehat{A}) + w(\widehat{B}) \leq w(A) + w(B)$ and added to $\widehat{\mathcal{L}}$. So we know that $F \in \chi_{\mathcal{F}}(\widehat{A}) \cap \chi_{\mathcal{F}}(\widehat{B})$. Now we claim that \widehat{A}, \widehat{B} and C are pairwise disjoint. Since $\widehat{A} \cup \widehat{B} \in \widehat{\mathcal{L}}$, $\widehat{A} \cap \widehat{B} = \emptyset$.

Finally, since $F \in \chi_{\mathcal{F}}(\widehat{A}) \cap \chi_{\mathcal{F}}(\widehat{B})$ and $F \in \chi'_{\mathcal{F}}(C)$, we get $\widehat{A}, \widehat{B} \subseteq F$ and $F \cap C = \emptyset$ which implies C is disjoint from \widehat{A} and \widehat{B} . This completes the correctness proof.

Running Time Analysis. We first consider the time T_G to construct the graph G . We can construct \mathcal{F} in time $2^{\mathcal{O}(\frac{k}{\log \log k})} \cdot \frac{1}{x_1^p(1-x_1)^q} \cdot (p+q)^{\mathcal{O}(1)} \cdot n \log n$. We can construct \mathcal{H} in time $2^{\mathcal{O}(\frac{p}{\log \log p})} \cdot \frac{1}{x_2^{p_1}(1-x_2)^{p_2}} \cdot (p_1+p_2)^{\mathcal{O}(1)} \cdot n \log n$. Now to add edges in the graph we do as follows. For each vertex in $\mathcal{L}_1 \cup \mathcal{L}_2$, we query the data structure created, spending the query time mentioned in Lemma 7.4, and add edges to the vertices in $\mathcal{F} \cup \mathcal{H}$ from it. So the running time to construct G is,

$$T_G \leq 2^{\mathcal{O}(\frac{k}{\log \log k})} k^{\mathcal{O}(1)} n \log n \left(\frac{1}{x_1^p(1-x_1)^q} + \frac{1}{x_2^{p_1}(1-x_2)^{p_2}} + \frac{|\mathcal{L}_1|}{x_1^{p_2}(1-x_1)^q} + \frac{|\mathcal{L}_2|}{x_1^{p_1}(1-x_1)^q} + \frac{|\mathcal{L}_1|}{(1-x_2)^{p_2}} + \frac{|\mathcal{L}_2|}{x_2^{p_1}} \right).$$

Now we bound the time T_C taken to construct $\widehat{\mathcal{L}}$ from G . To do the analysis we see how many times a vertex A in $\mathcal{L}_1 \cup \mathcal{L}_2$ is visited. It is exactly equal to the product of the degree of A to \mathcal{F} (denoted by $\text{degree}_{\mathcal{F}}(A)$) and the degree of A to \mathcal{H} (denoted by $\text{degree}_{\mathcal{H}}(A)$). Also note that two weights can be compared in $\mathcal{O}(\log W)$ time. Then

$$\begin{aligned} T_C &\leq \log W \left(\sum_{A \in \mathcal{L}_1} \text{degree}_{\mathcal{F}}(A) \cdot \text{degree}_{\mathcal{H}}(A) + \sum_{A \in \mathcal{L}_2} \text{degree}_{\mathcal{F}}(A) \cdot \text{degree}_{\mathcal{H}}(A) \right) \\ &\leq \log W \left(\sum_{A \in \mathcal{L}_1} \Delta_{(\chi_{\mathcal{F}}, p_1)}(n, p, q) \cdot \Delta_{(\chi_{\mathcal{H}}, p_1)}(n, p_1, p_2) + \right. \\ &\quad \left. \sum_{A \in \mathcal{L}_2} \Delta_{(\chi_{\mathcal{F}}, p_2)}(n, p, q) \cdot \Delta_{(\chi'_{\mathcal{H}}, p_2)}(n, p_1, p_2) \right) \\ &\leq 2^{\mathcal{O}(\frac{k}{\log \log k})} k^{\mathcal{O}(1)} \log^2 n \log W \left(\frac{|\mathcal{L}_1|}{x_1^{p_2}(1-x_1)^q(1-x_2)^{p_2}} + \frac{|\mathcal{L}_2|}{x_1^{p_1}(1-x_1)^q x_2^{p_1}} \right). \end{aligned}$$

So the total running time T is,

$$\begin{aligned} T &= T_G + T_C \\ &\leq 2^{\mathcal{O}(\frac{k}{\log \log(k)})} k^{\mathcal{O}(1)} n \log n \cdot \log W \left(\frac{1}{x_1^p (1-x_1)^q} + \frac{1}{x_2^{p_1} (1-x_2)^{p_2}} \right. \\ &\quad \left. + \frac{|\mathcal{L}_1|}{x_1^{p_2} (1-x_1)^q (1-x_2)^{p_2}} + \frac{|\mathcal{L}_2|}{x_1^{p_1} (1-x_1)^q x_2^{p_1}} \right). \end{aligned}$$

This completes the proof of the theorem. \square

Now we give a ready to use corollary for Theorem 11.1.

Corollary 11.1. *Let \mathcal{L}_1 be a p_1 -family of sets and \mathcal{L}_2 be a p_2 -family of sets over a universe U of size n . Furthermore, let $w : 2^U \rightarrow \mathbb{N}$ be an additive weight function, $|\mathcal{L}_1| = \binom{k}{p_1} \cdot 2^{o(k)}$, $|\mathcal{L}_2| = \binom{k}{p_2} \cdot 2^{o(k)}$, $\mathcal{L} = \mathcal{L}_1 \bullet \mathcal{L}_2$, $p = p_1 + p_2$ and $q = k - p$. There exists $\widehat{\mathcal{L}} \subseteq_{\minrep}^q \mathcal{L}$ of size $\binom{k}{p} \cdot 2^{o(k)}$ and it can be computed in time $\mathcal{O}(3.8408^k 2^{o(k)} n \log n \cdot \log W)$, where W is the maximum weight defined by w .*

Proof. We apply Theorem 11.1 for $0 < x_1, x_2 < 1$ and find $\mathcal{L}' \subseteq_{\minrep}^q \mathcal{L}$ of cardinality $x_1^{-p} (1-x_1)^{-q} 2^{o(k)} \cdot \log n$ in time,

$$T_1 = \mathcal{O} \left(\frac{z(n, k, W)}{x_1^p (1-x_1)^q} + \frac{z(n, k, W)}{x_2^{p_1} (1-x_2)^{p_2}} + \frac{z(n, k, W) \cdot |\mathcal{L}_1|}{x_1^{p_2} (1-x_1)^q (1-x_2)^{p_2}} + \frac{z(n, k, W) \cdot |\mathcal{L}_2|}{x_1^{p_1} (1-x_1)^q x_2^{p_1}} \right).$$

Now we apply Corollary 7.1 and get $\widehat{\mathcal{L}} \subseteq_{\minrep}^q \mathcal{L}'$ of cardinality $\binom{k}{p} \cdot 2^{o(k)}$ in time

$$T_2 = \mathcal{O} \left(x_1^{-p} (1-x_1)^{-q} \left(\frac{k}{q} \right)^q 2^{o(k)} \cdot \log^2 n \cdot \log W \right).$$

Due to Lemma 7.1, $\widehat{\mathcal{L}} \subseteq_{\minrep}^q \mathcal{L}$. Now we choose x_1, x_2 such that $T_1 + T_2$ is minimized.

Let $z(n, k, W) = 2^{o(k)} n \log n \cdot \log W$ So the total running time T to construct $\widehat{\mathcal{L}}$ is,

$$\begin{aligned}
T &= \min_{x_1, x_2} (T_1 + T_2) \\
&= \min_{x_1, x_2} \mathcal{O} \left(\frac{z(n, k, W)}{x_2^{p_1} (1 - x_2)^{p_2}} + \frac{z(n, k, W) \cdot \binom{k}{p_1}}{x_1^{p_2} (1 - x_1)^q (1 - x_2)^{p_2}} + \right. \\
&\quad \left. \frac{z(n, k, W) \cdot \binom{k}{p_2}}{x_1^{p_1} (1 - x_1)^q x_2^{p_1}} + \frac{z(n, k, W) \cdot \left(\frac{k}{q}\right)^q}{x_1^p (1 - x_1)^q} \right).
\end{aligned}$$

The above running time is upper bounded by $\mathcal{O}(3.8408^k 2^{o(k)} n \log n \cdot \log W)$. This completes the proof. \square

Chapter 12

Multilinear Monomial Detection

In this chapter we give an algorithm for parameterized version of multilinear monomial detection, using fast computation of representative family of a product family.

The problem MULTILINEAR MONOMIAL DETECTION is defined as follows.

MULTILINEAR MONOMIAL DETECTION (k -MLD)

Parameter: k

Input: An arithmetic circuit C over \mathbb{Z}^+ representing a polynomial $P(X)$ over \mathbb{Z}^+ .

Question: Does $P(X)$ construed as a sum of monomials contain a multilinear monomial of degree k ?

It is well known that we can replace any arithmetic circuit C with an equivalent circuit with fan-in two for all the internal nodes with quadratic blow up in the size. For an example, by replacing each node of in-degree greater than 2, with at most $s(C)$ many nodes of the same label and in-degree 2, we can convert a circuit C to a circuit C' of size $s(C') = s(C)^2$. So from now onwards we always assume that we are given a circuit of this form. We assume W be the maximum weight defined by w .

MULTILINEAR MONOMIAL DETECTION is the central problem in the algebraic approach of Koutis and Williams for designing fast parameterized algorithms [80, 81, 82, 119]. The idea behind the approach is to translate a given problem into the language of algebra by reducing it to the problem of deciding whether a constructed polynomial has a multilinear monomial of degree k . As it is mentioned implicitly by Koutis in [80], k -MLD can be solved in time $(2e)^k n^{\mathcal{O}(1)}$, where n is the input length, by making use of color coding. The color coding technique of Alon, Yuster and Zwick [4] is a fundamental and widely used technique in the design of parameterized algorithms. It appeared that most of the problems solvable by making use of color coding can be reduced to a multilinear monomial testing. Williams [119] gave a *randomized* algorithm solving k -MLD in time $2^k n^{\mathcal{O}(1)}$. The algorithms based

on the algebraic method of Koutis-Williams provide a dramatic improvement for a number of fundamental problems [19, 17, 56, 67, 80, 81, 82, 119].

The advantage of the algebraic approach over color coding is that for a number of parameterized problems, the algorithms based on this approach have much better exponential dependence on the parameter. On the other hand color coding based algorithms admit direct derandomization [4] and are able to handle integer weights with running time overhead poly-logarithmic in the weights. Obtaining deterministic algorithms matching the running times of the algebraic methods, but sharing these nice features of color coding remain a challenging open problem.

Our deterministic algorithm for k -MLD is the first non-trivial step towards resolving this problem. In fact, our algorithm solves a weighted version of k -MLD, where the elements of X are assigned weights and the task is to find a k -multilinear term with minimum weight. In the weighted version of k -MLD in addition to an arithmetic circuit C over variables $X = \{x_1, x_2, \dots, x_n\}$ representing a polynomial $P(X)$ over \mathbb{Z}^+ , we are also given an additive weight function $w : 2^X \rightarrow \mathbb{N}$ as an oracle. The task is that if there exists a k -multilinear term then find one with minimum weight. We call the weighted variant by k -WMLD. The running time of our deterministic algorithm is $\mathcal{O}(3.8408^k 2^{o(k)} s(C) n \log n \cdot \log W)$, where $s(C)$ is the size of the circuit and W is the maximum weight of an element from X .

Theorem 12.1. *k -WMLD can be solved in time $\mathcal{O}(3.8408^k 2^{o(k)} s(C) n \log n \cdot \log W)$.*

Proof. An arithmetic circuit C over \mathbb{Z}^+ with all leaves labelled from $X \cup \mathbb{Z}^+$ will represent sum of monomials with positive integer coefficients. With each multilinear term $\prod_{j=1}^k x_{i_j}$ we associate a set $\{x_{i_1}, \dots, x_{i_k}\} \subseteq X$. With any polynomial we can associate a family of subsets of X which corresponds to the set of multilinear terms in it. Since C is a directed acyclic graph, there exists a topological ordering $\pi = v_1, \dots, v_n$, such that all the nodes corresponding to variables appear before any other gate and for every directed arc uv we have that $u <_\pi v$. For a node v_i of the circuit let $P_i(X)$ be the multivariate polynomial represented by the subcircuit containing all the nodes w such that $w \leq_\pi v_i$. At every node we keep a family $\mathcal{F}_{v_i}^j$ of j -multilinear term, where $j \in \{1, \dots, k\}$. Let $\mathcal{F}_{v_i} = \bigcup_{x=1}^k \mathcal{F}_{v_i}^x$. Given a circuit C , if we compute associated family of subsets of X for each node we can answer the question of having a k -multilinear term of minimum weight in the polynomial computed by C . But the size of the family of subsets could be exponential in n , the number of variables. That is, the size of $\mathcal{F}_{v_i}^j$ could be $\binom{n}{j}$. So instead of storing all

subsets, we store a representative family for the associated family of subsets of each node. That is, we store $\widehat{\mathcal{F}}_{v_i}^j \subseteq_{\text{minrep}}^{k-j} \mathcal{F}_{v_i}^j$. The correctness of this step follows from the definition of $k - j$ -representative family.

We make a dynamic programming algorithm to detect a multilinear monomial of order k as follows. Our algorithm goes from left to right following the ordering given by π and computes \mathcal{F}_{v_i} from the families previously computed. The algorithm computes an appropriate representative family corresponding to each node of C . We show that we can compute a representative family \mathcal{F}_v associated with any node v , where the number of subsets with p elements in \mathcal{F}_v is at most $\binom{k}{p} 2^{\sigma(k)}$. When v is an input node then the associated family contains only one set. That is, if v is labelled with x_i then $\mathcal{F}_v = \{\{x_i\}\}$ and if v is labelled from \mathbb{Z}^+ then $\mathcal{F}_v = \{\emptyset\}$. When v is not an input node, then we have two cases.

Addition Gate. $v = v_1 + v_2$

Due to the left to right computation in the topological order, we have a representative families \mathcal{F}_{v_1} and \mathcal{F}_{v_2} for v_1 and v_2 respectively, where the number of subsets with p elements in \mathcal{F}_{v_1} as well as in \mathcal{F}_{v_2} will be at most $\binom{k}{p} 2^{\sigma(k)}$. The representative family corresponding to v will be the representative family of $\mathcal{F}_{v_1} \cup \mathcal{F}_{v_2}$. We partition $\mathcal{F}_{v_1} \cup \mathcal{F}_{v_2}$ based on the size of subsets in it. Let $\mathcal{F}_{v_1} \cup \mathcal{F}_{v_2} = \bigsqcup_{p \leq k} \mathcal{H}_p$, where \mathcal{H}_p contains all subsets of size p in $\mathcal{F}_{v_1} \cup \mathcal{F}_{v_2}$. Note that $|\mathcal{H}_p| \leq 2 \binom{k}{p} 2^{\sigma(k)}$. Now using Corollary 7.1, we can compute all $\widehat{\mathcal{H}}_p \subseteq_{\text{minrep}}^{k-p} \mathcal{H}_p$ in time

$$\mathcal{O} \left(2^{\sigma(k)} \log n \cdot \log W \cdot \sum_{p < k} \left\{ 2 \binom{k}{p} \cdot \left(\frac{k}{k-p} \right)^{k-p} \right\} \right)$$

where W is the maximum weight defined by weight function w . The above running time is upper bounded by $\mathcal{O}(2.851^k 2^{\sigma(k)} \log n \log W)$. We output $\bigcup_{p \leq k} \widehat{\mathcal{H}}_p$ as the representative family corresponding to the node v . By Corollary 7.1, $|\widehat{\mathcal{H}}_p| \leq \binom{k}{p} 2^{\sigma(k)}$ and hence the number of subsets with p elements in the repre-

representative family corresponding to v is at most $\binom{k}{p}2^{o(k)}$.

Multiplication Gate. $v = v_1 \times v_2$

Similar to the previous case we have a representative families \mathcal{F}_{v_1} and \mathcal{F}_{v_2} for v_1 and v_2 respectively, where the number of subsets with p elements in \mathcal{F}_{v_1} as well as in \mathcal{F}_{v_2} , is at most $\binom{k}{p}2^{o(k)}$. Here, the representative family corresponding to v will be the representative family of $\mathcal{F}_{v_1} \bullet \mathcal{F}_{v_2}$. The idea is to get representative families using Corollary 11.1 for different values of p_1 and p_2 . We have that

$$\mathcal{F}_{v_1} \bullet \mathcal{F}_{v_2} = \bigcup_{p_1, p_2} \mathcal{F}_{v_1}^{p_1} \bullet \mathcal{F}_{v_2}^{p_2},$$

where $\mathcal{F}_{v_i}^{p_i}$ contains all the subsets of size p_i in \mathcal{F}_{v_i} . We know that $|\mathcal{F}_{v_i}^{p_i}| \leq \binom{k}{p_i}2^{o(k)}$. Now by using Corollary 11.1, we compute $\widehat{\mathcal{F}_{v_1}^{p_1} \bullet \mathcal{F}_{v_2}^{p_2}} \subseteq_{\minrep}^{k-p_1-p_2} \mathcal{F}_{v_1}^{p_1} \bullet \mathcal{F}_{v_2}^{p_2}$ of size $\binom{k}{p_1+p_2} \cdot 2^{o(k)}$ for all p_1, p_2 such that $p_1 + p_2 \leq k$. Let $q = k - p_1 - p_2$, then all these computation can be done in time

$$\sum_{p_1, p_2} \mathcal{O}(3.8408^k 2^{o(k)} n \log n \cdot \log W) = \mathcal{O}(3.8408^k 2^{o(k)} n \log n \cdot \log W).$$

We output $\bigcup_{p_1, p_2} \widehat{\mathcal{F}_{v_1}^{p_1} \bullet \mathcal{F}_{v_2}^{p_2}}$ as the representative family corresponding to the node v . Note that the number of sets of size p in $\bigcup_{p_1, p_2} \widehat{\mathcal{F}_{v_1}^{p_1} \bullet \mathcal{F}_{v_2}^{p_2}}$ is bounded by $k \cdot \binom{k}{p}2^{o(k)} \leq \binom{k}{p}2^{o(k)}$.

Now we output a minimum weight set of size k (if exists) among the representative family corresponding to the root node, otherwise we output NO. Since there are $s(C)$ nodes in C , the total running time is bounded by $\mathcal{O}(3.8408^k 2^{o(k)} s(C) n \log n \cdot \log W)$. This completes the proof. \square

Part III

Representative Family in Linear Matroids

Chapter 13

Computing Representative Family in Linear Matroids

The notion of representative family can be extended to matroids. Let $M = (E, \mathcal{I})$ be a matroid and let $\mathcal{S} = \{S_1, \dots, S_t\}$ be a p -family of subsets of E . A subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is q -representative for \mathcal{S} if for every set $Y \subseteq E$ of size at most q , if there is a set $X \in \mathcal{S}$ disjoint from Y with $X \cup Y \in \mathcal{I}$, then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y and $\widehat{X} \cup Y \in \mathcal{I}$. In other words, if a set Y of size at most q can be extended to an independent set of size $|Y| + p$ by adding a subset from \mathcal{S} , then it also can be extended to an independent set of size $|Y| + p$ by adding a subset from $\widehat{\mathcal{S}}$ as well. We can easily show that the Lemmata 7.1, 7.2 and 7.3 also holds for representative families in matroids. In this chapter we give a fast algorithm for computing representative families and in subsequent chapters of this Part we show how they can be used to obtain improved parameterized and exact exponential algorithms for several fundamental and well studied problems. We prove the following theorem.

Theorem 13.1. *Let $M = (E, \mathcal{I})$ be a linear matroid of rank $p+q = k$ given together with its representation matrix A_M over a field \mathbb{F} . Let $\mathcal{S} = \{S_1, \dots, S_t\}$ be a family of independent sets of size p . Then a q -representative family $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ for \mathcal{S} with at most $\binom{p+q}{p}$ sets can be found in $\mathcal{O}\left(\binom{p+q}{p} t p^\omega + t \binom{p+q}{q}^{\omega-1}\right)$ operations over \mathbb{F} . Here, $\omega < 2.373$ is the matrix multiplication exponent.*

Actually, we will prove a variant of Theorem 13.1 which allows sets to have weights. The notion of weighted variant of representative family can be extend to matroids as well.

Definition 13.1 (Min/Max q -Representative Family). Given a matroid $M = (E, \mathcal{I})$, a family \mathcal{S} of subsets of E and a non-negative weight function $w : \mathcal{S} \rightarrow \mathbb{N}$, we say that a subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is min q -representative (max q -representative) for \mathcal{S} if the following holds: for every set $Y \subseteq E$ of size at most q , if there is a set $X \in \mathcal{S}$ disjoint from Y with $X \cup Y \in \mathcal{I}$, then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y with

1. $\widehat{X} \cup Y \in \mathcal{I}$; and
2. $w(\widehat{X}) \leq w(X)$ ($w(\widehat{X}) \geq w(X)$).

We use $\widehat{\mathcal{S}} \subseteq_{\minrep}^q \mathcal{S}$ ($\widehat{\mathcal{S}} \subseteq_{\maxrep}^q \mathcal{S}$) to denote a min q -representative (max q -representative) family for \mathcal{S} .

In this chapter we prove the following theorem.

Theorem 13.2. Let $M = (E, \mathcal{I})$ be a linear matroid of rank $p + q = k$, $\mathcal{S} = \{S_1, \dots, S_t\}$ be a p -family of independent sets and $w : \mathcal{S} \rightarrow \mathbb{N}$ be a non-negative weight function. Then there exists $\widehat{\mathcal{S}} \subseteq_{\minrep}^q \mathcal{S}$ ($\widehat{\mathcal{S}} \subseteq_{\maxrep}^q \mathcal{S}$) of size $\binom{p+q}{p}$. Moreover, given a representation A_M of M over a field \mathbb{F} , we can find $\widehat{\mathcal{S}} \subseteq_{\minrep}^q \mathcal{S}$ ($\widehat{\mathcal{S}} \subseteq_{\maxrep}^q \mathcal{S}$) of size at most $\binom{p+q}{p}$ in $\mathcal{O}\left(\binom{p+q}{p} t p^\omega + t \binom{p+q}{q} \omega^{-1}\right)$ operations over \mathbb{F} .

The proof for Theorem 13.2 is obtained by making the known exterior algebra based proof of Lovász [87, Theorem 4.8] algorithmic. For our proof we also need the following well-known generalized Laplace expansion of determinants. For a matrix $A = (a_{ij})$, the row set and the column set are denoted by $\mathbf{R}(A)$ and $\mathbf{C}(A)$ respectively. For $I \subseteq \mathbf{R}(A)$ and $J \subseteq \mathbf{C}(A)$, $A[I, J] = (a_{ij} \mid i \in I, j \in J)$ means the submatrix (or minor) of A with the row set I and the column set J . For $I \subseteq [n]$ let $\bar{I} = [n] \setminus I$ and $\sum I = \sum_{i \in I} i$.

Proposition 13.1 (Generalized Laplace expansion). For an $n \times n$ matrix A and $J \subseteq \mathbf{C}(A) = [n]$, it holds that

$$\det(A) = \sum_{I \subseteq [n], |I|=|J|} (-1)^{\sum I + \sum J} \det(A[I, J]) \det(A[\bar{I}, \bar{J}])$$

We refer to [99, Proposition 2.1.3] for a proof of the above identity. We always assume that the number of rows in the representation matrix A_M of M over a field \mathbb{F} is equal to $\text{rank}(M)=\text{rank}(A_M)$. Otherwise, using Gaussian elimination we can obtain a matrix of the desired kind in polynomial time. See [93, Proposition 3.1] for details.

Proof of Theorem 13.2. We only show how to find $\widehat{\mathcal{S}} \subseteq_{\text{minrep}}^q \mathcal{S}$ in the claimed running time. The proof for $\widehat{\mathcal{S}} \subseteq_{\text{maxrep}}^q \mathcal{S}$ is analogous, and for that case we only point out the places where the proof differs. If $t \leq \binom{k}{p}$, then we can take $\widehat{\mathcal{S}} = \mathcal{S}$. Clearly, in this case $\widehat{\mathcal{S}} \subseteq_{\text{minrep}}^q \mathcal{S}$. So from now onwards we always assume that $t > \binom{k}{p}$. For the proof we view the representation matrix A_M as a vector space over \mathbb{F} and each set $S_i \in \mathcal{S}$ as a subspace of this vector space. For every element $e \in E$, let x_e be the corresponding k -dimensional column in A_M . Observe that each $x_e \in \mathbb{F}^k$. For each subspace $S_i \in \mathcal{S}$, $i \in \{1, \dots, t\}$, we associate a vector $\vec{s}_i = \bigwedge_{j \in S_i} x_j$ in $\mathbb{F}^{\binom{k}{p}}$ as follows. In exterior algebra terminology, the vector \vec{s}_i is a wedge product of the vectors corresponding to elements in S_i . For a set $S \in \mathcal{S}$ and $I \in \binom{[k]}{p}$, we define $s[I] = \det(A_M[I, S])$.

We also define

$$\vec{s}_i = (s_i[I])_{I \in \binom{[k]}{p}}.$$

Thus the entries of the vector \vec{s}_i are the values of $\det(A_M[I, S_i])$, where I runs through all the p sized subsets of rows of A_M .

Let $H_{\mathcal{S}} = (\vec{s}_1, \dots, \vec{s}_t)$ be the $\binom{k}{p} \times t$ matrix obtained by taking \vec{s}_i as columns. Now we define a weight function $w' : \mathbf{C}(H_{\mathcal{S}}) \rightarrow \mathbb{R}^+$ on the set of columns of $H_{\mathcal{S}}$. For the column \vec{s}_i corresponding to $S_i \in \mathcal{S}$, we define $w'(\vec{s}_i) = w(S_i)$. Let \mathcal{W} be a set of columns of $H_{\mathcal{S}}$ that are linearly independent over \mathbb{F} , the size of \mathcal{W} is equal to the $\text{rank}(H_{\mathcal{S}})$ and is of minimum total weight with respect to the weight function w' . That is, \mathcal{W} is a minimum weight column basis of $H_{\mathcal{S}}$. Since the row-rank of a matrix is equal to the column-rank, we have that $|\mathcal{W}| = \text{rank}(H_{\mathcal{S}}) \leq \binom{k}{p}$. We define $\widehat{\mathcal{S}} = \{S_{\alpha} \mid \vec{s}_{\alpha} \in \mathcal{W}\}$. Let $|\widehat{\mathcal{S}}| = \ell$. Because $|\mathcal{W}| = |\widehat{\mathcal{S}}|$, we have that $\ell \leq \binom{k}{p}$. Without loss of generality, let $\widehat{\mathcal{S}} = \{S_i \mid 1 \leq i \leq \ell\}$ (else we can rename these

sets) and $\mathcal{W} = \{\vec{s}_1, \dots, \vec{s}_\ell\}$. The only thing that remains to show is that indeed $\widehat{\mathcal{S}} \subseteq_{\text{minrep}}^q \mathcal{S}$.

Let $S_\beta \in \mathcal{S}$ be such that $S_\beta \notin \widehat{\mathcal{S}}$. We show that if there is a set $Y \subseteq E$ of size at most q such that $S_\beta \cap Y = \emptyset$ and $S_\beta \cup Y \in \mathcal{I}$, then there exists a set $\widehat{S}_\beta \in \widehat{\mathcal{S}}$ disjoint from Y with $\widehat{S}_\beta \cup Y \in \mathcal{I}$ and $w(\widehat{S}_\beta) \leq w(S_\beta)$. Let us first consider the case $|Y| = q$. Since $S_\beta \cap Y = \emptyset$, it follows that $|S_\beta \cup Y| = p + q = k$. Furthermore, since $S_\beta \cup Y \in \mathcal{I}$, we have that the columns corresponding to $S_\beta \cup Y$ in A_M are linearly independent over \mathbb{F} ; that is, $\det(A_M[\mathbf{R}(A_M), S_\beta \cup Y]) \neq 0$.

Recall that, $\vec{s}_\beta = (s_\beta[I])_{I \in \binom{[k]}{p}}$, where $s_\beta[I] = \det(A_M[I, S_\beta])$. Similarly we define $y[L] = \det(A_M[L, Y])$ and

$$\vec{y} = (y[L])_{L \in \binom{[k]}{q}}.$$

Let $\Sigma J = \sum_{j \in S_\beta} j$. Define

$$\gamma(\vec{s}_\beta, \vec{y}) = \sum_{I \in \binom{[k]}{p}} (-1)^{\Sigma I + \Sigma J} s_\beta[I] \cdot y[\bar{I}].$$

Since $\binom{k}{p} = \binom{k}{k-p} = \binom{k}{q}$ the above formula is well defined. Observe that by Proposition 13.1, we have that $\gamma(\vec{s}_\beta, \vec{y}) = \det(A_M[\mathbf{R}(A_M), S_\beta \cup Y]) \neq 0$. We also know that \vec{s}_β can be written as a linear combination of vectors in $\mathcal{W} = \{\vec{s}_1, \vec{s}_2, \dots, \vec{s}_\ell\}$. That is, $\vec{s}_\beta = \sum_{i=1}^\ell \lambda_i \vec{s}_i$, $\lambda_i \in \mathbb{F}$, and for some i , $\lambda_i \neq 0$. Thus,

$$\begin{aligned} \gamma(\vec{s}_\beta, \vec{y}) &= \sum_I (-1)^{\Sigma I + \Sigma J} s_\beta[I] \cdot y[\bar{I}] \\ &= \sum_I (-1)^{\Sigma I + \Sigma J} \left(\sum_{i=1}^\ell \lambda_i s_i[I] \right) y[\bar{I}] \\ &= \sum_{i=1}^\ell \lambda_i \left(\sum_I (-1)^{\Sigma I + \Sigma J} s_i[I] y[\bar{I}] \right) \\ &= \sum_{i=1}^\ell \lambda_i \det(A_M[\mathbf{R}(A_M), S_i \cup Y]) \quad (\text{by Proposition 13.1}) \end{aligned}$$

Define

$$\mathbf{sup}(S_\beta) = \left\{ S_i \mid S_i \in \widehat{\mathcal{S}}, \lambda_i \det(A_M[\mathbf{R}(A_M), S_i \cup Y]) \neq 0 \right\}.$$

Since $\gamma(\vec{s}_\beta, \vec{y}) \neq 0$, we have that $(\sum_{i=1}^\ell \lambda_i \det(A_M[\mathbf{R}(A_M), S_i \cup Y])) \neq 0$ and thus $\mathbf{sup}(S_\beta) \neq \emptyset$. Observe that for all $S \in \mathbf{sup}(S_\beta)$ we have that $\det(A_M[\mathbf{R}(A_M), S \cup Y]) \neq 0$ and thus $S \cup Y \in \mathcal{I}$. We now show that $w(S) \leq w(S_\beta)$ for all $S \in \mathbf{sup}(S_\beta)$.

Claim 13.1. *For all $S \in \mathbf{sup}(S_\beta)$, $w(S) \leq w(S_\beta)$.*

Proof. For a contradiction assume that there exists a set $S_j \in \mathbf{sup}(S_\beta)$ such that $w(S_j) > w(S_\beta)$. Let \vec{s}_j be the vector corresponding to S_j and $\mathcal{W}' = (\mathcal{W} \cup \{\vec{s}_j\}) \setminus \{\vec{s}_\beta\}$. Since $w(S_j) > w(S_\beta)$, we have that $w(\vec{s}_j) > w(\vec{s}_\beta)$ and thus $w'(\mathcal{W}) > w'(\mathcal{W}')$. Now we show that \mathcal{W}' is also a column basis of $H_{\mathcal{S}}$. This will contradict our assumption that \mathcal{W} is a minimum weight column basis of $H_{\mathcal{S}}$. Recall that $\vec{s}_\beta = \sum_{i=1}^\ell \lambda_i \vec{s}_i$, $\lambda_i \in \mathbb{F}$. Since $S_j \in \mathbf{sup}(S_\beta)$, we have that $\lambda_j \neq 0$. Thus \vec{s}_j can be written as linear combination of vectors in \mathcal{W}' . That is,

$$\vec{s}_j = \lambda_j \vec{s}_\beta + \sum_{i=1, i \neq j}^\ell \lambda'_i \vec{s}_i. \quad (13.1)$$

Also every vector $\vec{s}_\gamma \notin \mathcal{W}$ can be written as a linear combination of vectors in \mathcal{W}

$$\vec{s}_\gamma = \sum_{i=1}^\ell \delta_i \vec{s}_i, \quad \delta_i \in \mathbb{F}. \quad (13.2)$$

By substituting (13.1) into (13.2), we conclude that every vector can be written as linear combination of vectors in \mathcal{W}' . This shows that \mathcal{W}' is also a column basis of $H_{\mathcal{S}}$, a contradiction proving the claim. \square

Claim 13.1 and the discussions preceding above it show that we could take any set $S \in \mathbf{sup}(S_\beta)$ as the desired $\widehat{S}_\beta \in \widehat{\mathcal{S}}$. Also, since $\det(A_M[\mathbf{R}(A_M), S \cup Y]) \neq 0$, we have that $S \cap Y = \emptyset$. This shows that indeed $\widehat{\mathcal{S}} \subseteq_{\minrep}^q \mathcal{S}$ for each Y of size q . This completes the proof for the case $|Y| = q$.

Suppose that $|Y| = q' < q$. Since M is a matroid of rank $k = p + q$, there exists a superset $Y' \in \mathcal{I}$ of Y of size q such that $S_\beta \cap Y' = \emptyset$ and $S_\beta \cup Y' \in \mathcal{I}$. This implies that there exists a set $\widehat{S} \in \widehat{\mathcal{S}}$ such that $\det(A_M[\mathbf{R}(A_M), \widehat{S} \cup Y']) \neq 0$ and $w(\widehat{S}) \leq w(S)$. Thus the columns corresponding to $\widehat{S} \cup Y$ are linearly independent.

We now consider the running time of the algorithm. To make the above proof algorithmic we need to

- (a) compute determinants and
- (b) apply fast Gaussian elimination to find a minimum weight column basis.

It is well known that one can compute the determinant of a $n \times n$ matrix in time $\mathcal{O}(n^\omega)$ [25]. For a rectangular matrix A of size $d \times n$ (with $d \leq n$), Bodlaender et al. [23] outline an algorithm computing a minimum weight column basis in time $\mathcal{O}(nd^{\omega-1})$. Thus given a p -family of independent sets \mathcal{S} we can construct the matrix $H_{\mathcal{S}}$ as follows. For every set S_i , we first compute \vec{s}_i . To do this we compute $\det(A_M[I, S_i])$ for every $I \in \binom{[k]}{p}$. This can be done in time $\mathcal{O}\left(\binom{p+q}{p} p^\omega\right)$. Thus, we can obtain the matrix $H_{\mathcal{S}}$ in time $\mathcal{O}\left(\binom{p+q}{p} t p^\omega\right)$. Given matrix $H_{\mathcal{S}}$ we can find a minimum weight column basis \mathcal{W} of $H_{\mathcal{S}}$ in time $\mathcal{O}(t \binom{p+q}{p}^{\omega-1})$. Given \mathcal{W} , we can easily recover $\widehat{\mathcal{S}}$. Thus, we can compute $\widehat{\mathcal{S}} \subseteq_{\minrep}^q \mathcal{S}$ in $\mathcal{O}\left(\binom{p+q}{p} t p^\omega + t \binom{p+q}{q}^{\omega-1}\right)$ field operations. This concludes the proof for finding $\widehat{\mathcal{S}} \subseteq_{\minrep}^q \mathcal{S}$. To find $\widehat{\mathcal{S}} \subseteq_{\maxrep}^q \mathcal{S}$, the only change we need to do in the algorithm for finding $\widehat{\mathcal{S}} \subseteq_{\minrep}^q \mathcal{S}$ is to find a *maximum weight column basis* \mathcal{W} of $H_{\mathcal{S}}$. This concludes the proof. \square

In Theorem 13.2 we assumed that $\text{rank}(M) = p + q$. However, one can obtain a similar result even when $\text{rank}(M) > p + q$ by computing the representation matrix of a k -truncation of $M = (E, \mathcal{I})$ using Lemma 4.2.

Theorem 13.3 ([85]). *Let $M = (E, \mathcal{I})$ be a linear matroid of rank n and let $\mathcal{S} = \{S_1, \dots, S_t\}$ be a family of independent sets, each of size b . Let A be an $n \times |E|$ matrix representing M over a field \mathbb{F} , where $\mathbb{F} = \mathbb{F}_{p^\ell}$ or \mathbb{F} is \mathbb{Q} . Then there is deterministic algorithm which computes a representative set $\widehat{\mathcal{S}} \subseteq_{rep}^q \mathcal{S}$ of size at most $nb \binom{b+q}{b}$, using $\mathcal{O}\left(\binom{b+q}{b} t b^3 n^2 + t \binom{b+q}{b}^{\omega-1} (bn)^{\omega-1}\right) + (n + |E|)^{\mathcal{O}(1)}$ operations over the field \mathbb{F} .*

A proof of Theorem [13.3](#) can be found in [\[85\]](#).

Chapter 14

Minimum Equivalent Graph

In this chapter we show that representative families can be used to design exact exponential algorithms as well. For a given digraph D , a subdigraph D' of D is said to be an *equivalent* subdigraph of D if for any pair of vertices $u, v \in V(D)$ if there is a directed path in D from u to v then there is also a directed path from u to v in D' . That is, reachability of vertices in D and D' is same. In this section we study a problem where given a digraph D the objective is to find an equivalent subdigraph of D of D with as few arcs as possible. Equivalently, the objective is to remove the maximum number of arcs from a digraph D without affecting its reachability. More precisely the problem we study is as follows.

MINIMUM EQUIVALENT GRAPH (MEG)

Input: A directed graph D

Task: Find an equivalent subdigraph of D with the minimum number of arcs.

Previous Work. MEG is a classical NP-hard problem generalizing the HAMILTONIAN CYCLE problem, see Chapter 12 of the book [7] for an overview of combinatorial and algorithmic results on MEG. The algorithmic studies of MEG can be traced to the work of Moyles and Thompson [98] from 1969, who gave a (non-trivial) branching algorithm solving MEG in time $\mathcal{O}(n!)$. In 1975, Hsu in [70] discovered a mistake in the algorithm of Moyles and Thompson, and designed a different branching algorithm for this problem. Martello [90] and Martello and Toth [91] gave another branching based algorithm with running time $\mathcal{O}(2^m)$.

As it was already observed by Moyles and Thompson [98] the hardest instances of MEG are strong digraphs. A digraph is strong if for every pair of vertices $u \neq v$, there are directed paths from u to v and from v to u . MEG restricted to strong digraphs is known as the MINIMUM SCSS (strongly connected spanning subgraph) problem. It is known that the MEG problem reduces in linear time to MINIMUM SCSS, see e.g. [36].

Our Result. We show that MEG is solvable in time $\mathcal{O}(2^{4\omega n} mn)$, where n is the

number of vertices and m is the number of arcs in D .

14.1 Algorithm for MEG

The following proposition is due to Moyles and Thompson [98], see also [7, Sections 2.3], reduces the problem of finding a minimum equivalent subdigraph of an arbitrary D to a strong digraph.

Proposition 14.1. *Let D be a digraph on n vertices with strongly connected components C_1, \dots, C_r . Given a minimum equivalent subdigraph C'_i for each C_i , $i \in [r]$, one can obtain a minimum equivalent subdigraph D' of D containing each of C'_i in $\mathcal{O}(n^\omega)$ time.*

Observe that for a strong digraph D any equivalent subdigraph is also strong. By Proposition 14.1, MEG reduces to the following problem.

MINIMUM STRONGLY CONNECTED SPANNING SUBGRAPH (MINIMUM SCSS)
Input: A strongly connected directed graph D
Task: Find a strong spanning subdigraph of D with the minimum number of arcs.

A digraph T is an *out-tree* (an *in-tree*) if T is an oriented tree with just one vertex s of in-degree zero (out-degree zero). The vertex s is the root of T . If an out-tree (in-tree) T is a spanning subdigraph of D , T is called an *out-branching* (an *in-branching*). We use the notation B_s^+ (B_s^-) to denote an out-branching (in-branching) rooted at s of the digraph.

It is known that a digraph is strong if and only if it contain an out-branching and an in-branching rooted at some vertex $v \in V(D)$ [7, Proposition 12.1.1].

Proposition 14.2. *Let D be a strong digraph on n vertices, let v be an arbitrary vertex of $V(D)$, and $\ell \leq n - 2$ be a natural number. Then there exists a strong spanning subdigraph of D with at most $2n - 2 - \ell$ arcs if and only if D contains an in-branching B_v^- and an out-branching B_v^+ with root v so that $|A(B_v^+) \cap A(B_v^-)| \geq \ell$ (that is, they have at least ℓ common arcs).*

Proposition 14.2 implies that the MINIMUM SCSS problem is equivalent to finding, for an arbitrary vertex $v \in V(D)$, an out-branching B_v^+ and an in-branching B_v^-

that maximizes $|A(B_v^+) \cap A(B_v^-)|$. For our exact algorithm for MINIMUM SCSS we implement this equivalent version using representative sets.

Let D be a strong digraph and $s \in V(D)$ be a fixed vertex. By D_s^- we denote the digraph $D - \text{Out}_D(s)$, i.e, the digraph obtained from D by deleting the arcs in $\text{Out}_D(s)$. Similarly, by D_s^+ we denote the digraph $D - \text{In}_D(s)$.

First we construct four matroids. Recall that $U(D)$ denote the underlying undirected graph of D . The first two matroids $M_1 = (E_1, \mathcal{I}_1)$, $M_2 = (E_2, \mathcal{I}_2)$ are the graphic matroids on $U(D)$. Observe that

$$A(D_s^+) = \bigsqcup_{v \in V(D_s^+)} \text{In}_{D_s^+}(v) \text{ and } A(D_s^-) = \bigsqcup_{v \in V(D_s^-)} \text{Out}_{D_s^-}(v).$$

Thus the arcs of D_s^+ can be partitioned into sets of in-arcs and similarly the arcs of D_s^- into sets of out-arcs. Let $E_3 = A(D_s^+)$ and $E_4 = A(D_s^-)$. The other two matroids are the following partition matroids $M_3 = (E_3, \mathcal{I}_3)$, $M_4 = (E_4, \mathcal{I}_4)$, where

$$\mathcal{I}_3 = \{I \mid I \subseteq A(D_s^+), \text{ for every } v \in V(D_s^+) = V(D), |I \cap \text{In}_{D_s^+}(v)| \leq 1\},$$

and

$$\mathcal{I}_4 = \{I \mid I \subseteq A(D_s^-), \text{ for every } v \in V(D_s^-) = V(D), |I \cap \text{Out}_{D_s^-}(v)| \leq 1\}.$$

Let $n = |V(D)|$. We define the matroid $M = (E, \mathcal{I})$ as the direct sum $M = M_1 \oplus M_2 \oplus M_3 \oplus M_4$. By Proposition 4.3, the matroids M_3 and M_4 are representable over a field of size $\mathcal{O}(n^2)$ and these representations can be constructed in polynomial time. Since D is strongly connected and $\text{In}_{D_s^+}(s) = \emptyset$, the rank of M_3 is $n - 1$. By similar arguments, we have that the rank of M_4 is $n - 1$. By Proposition 4.4, we know that graphic matroid is representable over any field of size at least 2. So the matroids M_1 and M_2 are also representable over a field of size $\mathcal{O}(n^2)$. Also note that the rank of both M_1 and M_2 are $n - 1$, because $U(D)$ is connected. Hence, by Proposition 4.2, the matroid M is representable over a field of size $\mathcal{O}(n^2)$. The rank of the matroid M is $4n - 4$.

Let us note that for each arc $e \in A(D)$ which is not incident with s , we have four elements in the matroid M , corresponding to the copies of e in M_i , $i \in \{1, \dots, 4\}$. We denote these elements by e_i , $i \in \{1, \dots, 4\}$. For every edge $e \in A(D)$ incident with s , we have three corresponding elements. We denote them by e_1, e_2, e_3 , or e_1, e_2, e_4 , depending on the case when e is in-arc or out-arc for s .

For $i \in \{1, \dots, n-1\}$, we define

$$\mathcal{B}^{4i} = \left\{ W \mid W \in \mathcal{I}, |W| = 4i, \right. \\ \left. \forall e \in A(D) \text{ either } W \cap \{e_1, e_2, e_3, e_4\} = \emptyset \text{ or } \{e_1, e_2, e_3, e_4\} \subseteq W \right\}.$$

For $W \in \mathcal{I}$, by A_W we denote the set of arcs $e \in A(D)$ such that $\{e_1, e_2, e_3, e_4\} \cap W \neq \emptyset$. Now we are ready to state the lemma that relates representative sets and the MINIMUM SCSS problem.

Lemma 14.1. *Let D be a strong digraph on n vertices and $\ell \leq n-2$ be a natural number. Then there exists a strong spanning subdigraph D' of D with at most $2n-2-\ell$ arcs if and only if there exists a set $\widehat{F} \in \widehat{\mathcal{B}}^{4\ell} \subseteq_{\text{rep}}^{n'-4\ell} \mathcal{B}^{4\ell}$ such that there is an out-branching D_s^+ containing $A_{\widehat{F}}$ rooted at s and an in-branching D_s^- containing $A_{\widehat{F}}$ rooted at s . Here, $n' = 4n-4$.*

Proof. (\Rightarrow) Let $s \in V(D)$ be a fixed vertex and M is the matroid constructed as above with respect the vertex s . Let D' be a strong spanning subdigraph of D with at most $2n-2-\ell$ arcs. Thus, by Proposition 14.2 we have that there exists an out-branching B_s^+ and an in-branching B_s^- in D' such that $|A(B_s^+) \cap A(B_s^-)| \geq \ell$. Observe that the arcs in $A(B_s^+) \cap A(B_s^-)$ form an out-forest (in-forest). Let F' be an arbitrary subset of $A(B_s^+) \cap A(B_s^-)$ containing exactly ℓ arcs. Take $X' = A(B_v^+) \setminus F'$ and $Y' = A(B_v^-) \setminus F'$. Observe that X' and Y' need not be disjoint. Clearly, $|X'| = |Y'| = n-1-\ell$.

In matroid M , one can associate with D' an independent set $I_{D'}$ of size $4n-4$ as follows:

$$I_{D'} = \bigcup_{e \in F'} \{e_1, e_2, e_3, e_4\} \bigcup_{e \in X'} \{e_1, e_3\} \bigcup_{e \in Y'} \{e_2, e_4\}.$$

By our construction, we have that $I_{D'}$ is an independent set in \mathcal{I} and $|I_{D'}| = 4\ell + 4(n-1-\ell) = n'$. Let $F = \bigcup_{e \in F'} \{e_1, e_2, e_3, e_4\}$, $X = \bigcup_{e \in X'} \{e_1, e_3\}$ and $Y = \bigcup_{e \in Y'} \{e_2, e_4\}$. Then notice that $F \in \mathcal{B}^{4\ell}$ and $F \subset I_{D'}$. This implies that there exists a set $\widehat{F} \in \widehat{\mathcal{B}}^{4\ell} \subseteq_{\text{rep}}^{n'-4\ell} \mathcal{B}^{4\ell}$ such that $I_{\widehat{D}} = \widehat{F} \cup X \cup Y \in \mathcal{I}$. Consider the following four sets.

1. Let $W_1 = \{e_1 \mid e \in X \cup A_{\widehat{F}}\}$ then we have that $W_1 \subseteq I_{\widehat{D}}$ and thus $W_1 \in \mathcal{I}_1$.
This together with the fact that $|W_1| = n - 1$ implies that $X' \cup A_{\widehat{F}}$ forms a spanning tree in $U(D)$.
2. Let $W_2 = \{e_2 \mid e \in Y \cup A_{\widehat{F}}\}$. Similar to the first case, then $Y' \cup A_{\widehat{F}}$ forms a spanning tree in $U(D)$.
3. Let $W_3 = \{e_3 \mid e \in X \cup A_{\widehat{F}}\}$ then we have that $W_3 \subseteq I_{\widehat{D}}$ and thus $W_3 \in \mathcal{I}_3$.
This together with the fact that $|W_1| = |W_3| = n - 1$ and that $X' \cup A_{\widehat{F}}$ is a spanning tree in $U(D)$ implies that $X' \cup A_{\widehat{F}}$ forms an out-branching rooted at s in D_s^+ .
4. Let $W_4 = \{e_4 \mid e \in Y' \cup A_{\widehat{F}}\}$. Similar to the previous case, then $Y' \cup A_{\widehat{F}}$ forms an in-branching rooted at s in D_s^- .

Thus we have shown that D_s^+ and D_s^- are the required out-branching and in-branching respectively.

(\Leftarrow) Suppose there exists a set $\widehat{F} \in \widehat{\mathcal{B}}^{4\ell} \subseteq_{rep}^{n'-4\ell} \mathcal{B}^{4\ell}$ such that there is an out-branching D_s^+ containing $A_{\widehat{F}}$ rooted at s and an in-branching D_s^- containing $A_{\widehat{F}}$ rooted at s . Note that $|A_{\widehat{F}}| = \ell$. Then by Proposition 14.2, there exists a strong spanning subdigraph of D with at most $2n - 2 - \ell$ arcs. This concludes the proof of the lemma. \square

Define $\mathcal{T}_{rm}(t, p, q)$ be the time required to compute a family $\widehat{\mathcal{S}} \subseteq_{rep}^q \mathcal{S}$ of size $\binom{p+q}{q}$, in a linear matroid M of rank $p + q$. By Theorem 13.1, $\mathcal{T}_{rm}(t, p, q)$ is bounded by $\mathcal{O}\left(\binom{p+q}{p} t p^\omega + t \binom{p+q}{q}^{\omega-1}\right)$ multiplied by the time required to perform operations over the field in which the linear matroid M is representable.

Lemma 14.2. *Let D be a strong digraph on n vertices and $\ell \leq n - 2$ be a natural number. Then in time $\mathcal{O}\left(\max_{i \in [\ell]} \binom{n'}{4i}^\omega m \log n\right)$ we can compute $\widehat{\mathcal{B}}^{4\ell} \subseteq_{rep}^{n'-4\ell} \mathcal{B}^{4\ell}$ of size $\binom{n'}{4\ell}$. Here, $n' = 4n - 4$.*

Proof. We describe a dynamic programming based algorithm. Let \mathcal{D} be an array of size ℓ . The entry $\mathcal{D}[i]$ will store the family $\widehat{\mathcal{B}}^{4i} \subseteq_{rep}^{n'-4i} \mathcal{B}^{4\ell}$. We fill the entries in the

array \mathcal{D} in the increasing order of its index, that is, from $0, \dots, \ell$. For the base case define $\widehat{\mathcal{B}}^0 = \{\emptyset\}$ and let $W = \{\{e_1, e_2, e_3, e_4\} \mid e \in A(D)\}$. Given that $\mathcal{D}[i]$ is filled for all $i' \leq i$, we fill $\mathcal{D}[i+1]$ as follows. Define $\mathcal{N}^{4(i+1)} = \left(\widehat{\mathcal{B}}^{4i} \bullet W\right) \cap \mathcal{I}$.

Claim 14.1. *For all $0 \leq i \leq \ell - 1$, $\mathcal{N}^{4(i+1)} \subseteq_{rep}^{n'-4(i+1)} \mathcal{B}^{4(i+1)}$.*

Proof. Let $S \in \mathcal{B}^{4(i+1)}$ and Y be a set of size $n' - 4(i+1)$ such that $S \cap Y = \emptyset$ and $S \cup Y \in \mathcal{I}$. We will show that there exists a set $\widehat{S} \in \mathcal{N}^{4(i+1)}$ such that $\widehat{S} \cap Y = \emptyset$ and $\widehat{S} \cup Y \in \mathcal{I}$. This will imply the desired result.

Let $e \in A(D)$ such that $\{e_1, e_2, e_3, e_4\} \subseteq S$. Define $S^* = S \setminus \{e_1, e_2, e_3, e_4\}$ and $Y^* = Y \cup \{e_1, e_2, e_3, e_4\}$. Since $S \cup Y \in \mathcal{I}$ we have that $S^* \in \mathcal{I}$ and $Y^* \in \mathcal{I}$. Observe that $S^* \in \mathcal{B}^{4i}$, $S^* \cup Y^* \in \mathcal{I}$ and the size of Y^* is $n' - 4i$. This implies that there exists \widehat{S}^* in $\widehat{\mathcal{B}}^{4i} \subseteq_{rep}^{n'-4i} \mathcal{B}^{4i}$ such that $\widehat{S}^* \cup Y^* \in \mathcal{I}$. Thus $\widehat{S}^* \cup \{e_1, e_2, e_3, e_4\} \in \mathcal{I}$ and also in $\widehat{\mathcal{B}}^{4i} \bullet W$ and thus in $\mathcal{N}^{4(i+1)}$. Taking $\widehat{S} = \widehat{S}^* \cup \{e_1, e_2, e_3, e_4\}$ suffices for our purpose. This completes the proof of the claim. \square

We fill the entry for $\mathcal{D}[i+1]$ as follows. Observe that $\mathcal{N}_{uv}^{4(i+1)} = (\mathcal{D}[i, w] \bullet W) \cap \mathcal{I}$. We already have computed the family corresponding to $\mathcal{D}[i]$. By Theorem 13.1, $|\widehat{\mathcal{B}}^{4i}| \leq \binom{n'}{4i}$ and thus $|\mathcal{N}^{4(i+1)}| \leq 4m \binom{n'}{4i}$. Furthermore, we can compute $\mathcal{N}^{4(i+1)}$ in time $\mathcal{O}\left(mn \binom{n'}{4i}\right)$. Using Theorem 13.1, we can compute $\widehat{\mathcal{N}}^{4(i+1)} \subseteq_{rep}^{n'-4(i+1)} \mathcal{N}^{4(i+1)}$ in time $\mathcal{T}_{Im}(t, 4i+4, n' - 4(i+1))$, where $t = 4m \binom{n'}{4i}$.

By Claim 14.1 we know that $\mathcal{N}^{4(i+1)} \subseteq_{rep}^{n'-4(i+1)} \mathcal{B}^{4(i+1)}$. Thus Lemma 7.1 implies that $\widehat{\mathcal{N}}^{4(i+1)} = \widehat{\mathcal{B}}^{4(i+1)} \subseteq_{rep}^{n'-4(i+1)} \mathcal{B}^{4(i+1)}$. We assign this family to $\mathcal{D}[i+1]$. This completes the description and the correctness of the dynamic programming. The field size for uniform matroids are upper bounded by $\mathcal{O}(n^2)$ and thus we can perform all the field operations in time $\mathcal{O}(\log n)$. Thus, the running time of this algorithm is upper bounded by

$$\mathcal{O}\left(\sum_{i=1}^{\ell} \mathcal{T}_{Im}\left(4m \binom{n'}{4(i-1)}, 4i, n' - 4i\right)\right) = \mathcal{O}\left(\max_{i \in [\ell]} \binom{n'}{4i}^{\omega} m \log n\right).$$

This completes the proof. \square

Lemma 14.3. MINIMUM SCSS *can be solved in time* $\mathcal{O}(2^{4\omega n} m \log n)$.

Proof. Let us fix $n' = 4n - 4$. Proposition 14.2 implies that the MINIMUM SCSS problem is equivalent to finding, for an arbitrary vertex $s \in V(D)$, an out-branching B_v^+ and an in-branching B_v^- that maximizes $|A(B_v^+) \cap A(B_v^-)|$. We guess the value of $|A(B_v^+) \cap A(B_v^-)|$ and let this be ℓ . By Lemma 14.1, there exists a strong spanning subdigraph D' of D with at most $2n - 2 - \ell$ arcs if and only if there exists a set $\widehat{F} \in \widehat{\mathcal{B}}^{4\ell} \subseteq_{rep}^{n'-4\ell} \mathcal{B}^{4\ell}$ such that D has a strong spanning subdigraph \bar{D} with $A_{\widehat{F}} \subseteq A(\bar{D})$. Recall that for $X \in \mathcal{I}$, by A_X we denote the set of arcs $e \in A(D)$ such that $\{e_1, e_2, e_3, e_4\} \cap X \neq \emptyset$. Now using Lemma 14.2 we compute $\widehat{\mathcal{B}}^{4\ell} \subseteq_{rep}^{n'-4\ell} \mathcal{B}^{4\ell}$ of size $\binom{n'}{4\ell}$ in time $\mathcal{O}\left(\max_{i \in [\ell]} \binom{n'}{4i}^\omega m \log n\right)$.

For every $\widehat{F} \in \widehat{\mathcal{B}}^{4\ell}$ we test whether $A_{\widehat{F}}$ can be extended to an out-branching in D_s^+ and to an in-branching in D_s^- . We can do it in $\mathcal{O}(n(n+m))$ -time by putting weights 0 to the arcs of $A_{\widehat{F}}$ and weights 1 to all remaining arcs and then by running the classical algorithm of Edmonds [45]. Since $\ell \leq n - 2$, the running time of this algorithm is upper bounded by $\mathcal{O}(2^{4\omega n} m \log n)$. This concludes the proof. \square

Finally, we are ready to prove the main result of this section

Theorem 14.1. MINIMUM EQUIVALENT GRAPH *can be solved in* $\mathcal{O}(2^{4\omega n} m \log n)$ *time.*

Proof. Given an arbitrary digraph D we first find its strongly connected components C_1, \dots, C_s . Now on each C_i , we apply Lemma 14.3 and obtain a minimum equivalent subdigraph C'_i . After this we apply Proposition 14.1 and obtain a minimum equivalent subdigraph of D . Since all the steps except Lemma 14.3 takes polynomial time we get the desired running time. This completes the proof. \square

A weighted variant of MINIMUM EQUIVALENT GRAPH has also been studied in literature. More precisely the problem is defined as follows.

MINIMUM WEIGHT EQUIVALENT GRAPH (MWEG)

Input: A directed graph D and a weight function $w : A(D) \rightarrow \mathbb{N}$.

Task: Find a minimum weight equivalent subdigraph of D .

MWEG can be solved along the same line as MEG but to do this we need to use the notion of min q -representative family and use Theorem 13.2 instead of Theorem 13.1. These changes give us the following theorem.

Theorem 14.2. MINIMUM WEIGHT EQUIVALENT GRAPH *can be solved in $\mathcal{O}(2^{4\omega n} m \log n \cdot \log W)$ time. Here, W is the maximum value assigned by the weight function $w : A(D) \rightarrow \mathbb{N}$.*

Chapter 15

Editing to Connected f -Degree Graph

A subgraph F of a graph G is a factor of G , if F is a spanning subgraph of G . When a factor F is described in terms of its degrees, it is called a *degree-factor*. For example, one of the most fundamental notions in Graph Theory is 1-factor (or a perfect matching), the case when a factor F has all of its degrees equal to 1. Another example is r -factor, a regular spanning subgraph of degree r . More generally, for a function $f: V(G) \rightarrow \mathbb{N}$, subgraph F is a f -factor of G if for every $v \in V(G)$, $d_F(v)$, the degree of v in F is exactly $f(v)$. The study of degree factors is one of the mainstays in combinatorics with long history dating back to 1847 to the works of Kirkman [74], and Petersen [106]. We refer to surveys [2, 110], as well as the book of Lovász and Plummer [88] for an extensive overview of degree factors.

Another broad set of degree-factor problems is obtained by requesting the factor to be connected. The most famous examples are another old classical Graph Theory notions, namely Hamiltonian cycle, which is a connected 2-factor, and Eulerian subgraph, which is a connected even-degree factor. We refer to the survey of Kouider and Vestergaard [78] on connected factors, as well as to the book of Fleischner [47] for a thorough study of Eulerian graphs and related topics.

A natural algorithmic problem concerning (connected) f -factors is for a given graph G and a function f , to decide if G contains a (connected) f -factor. While deciding if a given graph contains an f -factor can be done in polynomial time for any function f [6], deciding the existence of even a connected 2-factor (Hamiltonian cycle) is NP-complete. In this chapter we study parameterized complexity of the following algorithmic generalization of the problem of finding a connected f -factor.

<p>EDITING TO CONNECTED f-DEGREE GRAPH (ECG) Parameter: k</p> <p>Input: An undirected graph G, a function $f : V(G) \rightarrow \{1, 2, \dots, d\}$ and an integer k</p> <p>Question: Does there exist a connected graph F such that for every vertex v, $d_F(v) = f(v)$, and the size of the symmetric difference $E(G) \Delta E(F) \leq k$?</p>

The main result of this chapter is the following theorem

Theorem 15.1. *ECG is solvable in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ deterministically.*

The proof of our theorem is based on (i) color-coding and (ii) fast computation of representative family over a linear matroid. Our proof requires the usage representative family in some non-standard matroids. In particular, we use fast representative family computations over direct sum matroid of ℓ -elongation of co-graphic matroid associated with G and uniform matroid over $E(G)$. We believe that this combination could be useful for designing parameterized algorithms for other edge editing problems. To the best of our knowledge this is the first uses of elongation of matroids in designing parameterized algorithms.

In previous chapters we have seen many algorithms using representative families and these algorithms can modified to solve the weighted version the problems. So it would be natural to suggest that the nice properties of matroids would help us with the “weighted” version of ECG as well.

<p>EDITING TO CONNECTED f-DEGREE GRAPH WITH COSTS Parameter: $k+d$</p> <p>Input: A graph G, functions $f : V(G) \rightarrow \{1, 2, \dots, d\}$ and $c : \binom{V(G)}{2} \rightarrow \mathbb{N}$ and $k, C \in \mathbb{N}$</p> <p>Question: Does there exist a connected graph F such that for every vertex v, $d_F(v) = f(v)$, $E(G) \Delta E(F) \leq k$, and $c(E(G) \Delta E(F)) \leq C$?</p>
--

However, in spite of our attempts, we could not extend the results of Theorem 21.1 to EDITING TO CONNECTED f -DEGREE GRAPH WITH COSTS. The following Theorem explains why our initial attempts failed.

Theorem 15.2. *EDITING TO CONNECTED f -DEGREE GRAPH WITH COSTS (parameterized by $k + d$) is W[1]-hard even when the input graph G is a tree and costs are restricted to be 0 or 1.*

Previous work. It was shown by Mathieson and Szeider [94] that the problem of deleting k vertices to transform an input graph into an r -regular graph, where $r \geq 3$ is W[1]-hard parameterized by k . For edge-modification problems to a graph with certain degrees, Mathieson and Szeider have shown in [94] that EDITING TO f -DEGREE GRAPH, the case when the requirement that the resulting graph F is connected is omitted, is solvable in polynomial time. As with the f -factor problem, the situation changes drastically when one adds the requirement that the resulting

graph F is connected. A simple reduction from Hamiltonian cycle shows that in this case deciding if a graph can be edited into a connected 2-degree graph, i.e. a cycle, by changing at most k adjacencies, is NP-complete [64].

Golovach in [64] have shown that when parameterized by the maximum vertex degree d in the resulting graph plus the number of editing operations k , the problem EDITING TO CONNECTED f -DEGREE GRAPH is FPT. In the same paper, it was shown that the variant when the resulting graph F is regular, the problem is FPT parameterized by k . However, prior to our work the complexity status of EDITING TO CONNECTED f -DEGREE GRAPH remained open. Thus Theorem 21.1 resolves the problem in affirmative. However, we still do not know the kernelization status of this problem and leave it as an interesting open problem.

Our Approach. Any solution to our problem is of the form $D \cup A \in \binom{V(G)}{2}$ where $D \subseteq E(G)$ and $A \subseteq \overline{E(G)}$. The sets D and A are called a deletion set and an addition set, respectively, corresponding to the solution $D \cup A$. We start by characterizing our solution in terms of deletion set D (called nice deletion set), satisfying certain properties and has an accompanying map ψ . This map together with other properties of nice deletion set allows us to recover the addition set A in polynomial time. Thus, our whole effort is in finding this nice deletion set D . This viewpoint is useful as this allows us to “forget about the addition set” and concentrate on finding a nice deletion set. However, this is not useful for designing the dynamic programming algorithm. To achieve this we view the solution $D \cup A$ as a system of “alternating walks and alternating even closed walks”. An alternating walks and closed walks are essentially normal walks and closed walks with edges from $D \cup A$ and they do not have consecutive edges from either D or A . We take this view point to construct the dynamic programming algorithm as this allows us to go from one state to other using one edge (either of an addition set or of a deletion set). The number of states can be upper bounded by $2^{\mathcal{O}(k)}$. However, the number of sets $D' \cup A'$ that could satisfy the prerequisite of being in a particular table entry could be as large as $n^{\mathcal{O}(k)}$ and thus this would not lead to an FPT algorithm. However, we follow this template for our algorithm and use some more structural properties to prune this deletion set.

Our first observation towards an FPT algorithm is that after we delete the edges in D then the number of components can at most be $|D| - k + 1$. This allows us to show that in fact we can think of D being an independent in a independent set in the matroid $M_G^*(\ell)$, ℓ -elongation of the co-graphic matroid, M_G^* , associated with G , where $\ell = |E(G)| - |V(G)| + k - |D| + 1$ (we refer to preliminaries for the definition). Next we show that for addition set A , all we need is to store some form of disjointness and that can be captured using uniform matroid over the universe $\overline{E(G)}$. Let $U_{m',k-k'}$ be a uniform matroid with ground set $\overline{E(G)}$, where $m' = |\overline{E(G)}|$. From the definition of $U_{m',k-k'}$, any set A of size at most $k - k'$ is independent in $U_{m',k-k'}$. We have already explained that we view the deletion set D as an independent set in $M_G^*(\ell)$ where $\ell = |E(G)| - |V(G)| + k - k' + 1$. Thus, to see the solution set $D \cup A$ as an independent set in a single matroid, we consider direct sum of $M_G^*(\ell)$ and $U_{m',k-k'}$. That is, let $M = M_G^*(\ell) \oplus U_{m',k-k'}$. In M , a set I is an independent set if

and only if $I \cap E(G)$ is an independent set in $M_G^*(\ell)$ and $I \cap \overline{E(G)}$ is an independent set in $U_{m', k-k'}$. This ensures that any solution $D \cup A$ is an independent set in M . By viewing any solution of the problem as an independent set in a matroid M (which is linear), we can use the q -representative families to prune the table. However, we still need to take care of a technical requirement in the definition of a nice deletion set. Towards this, we show that for every deletion set D there exists a set of edges $W_D \subseteq E(G)$ (disjoint from D) of size at most $6k$ such that if these edges are not picked then we can satisfy that technical requirement. To achieve this we apply color coding technique and this can be derandomized using universal sets.

Organization of the chapter. In Section 15.1 we give an brief overview of our algorithm and in Section 15.2 we explain our algorithm formally.

15.1 An overview of our algorithm

Any solution to our problem is of the form $D \cup A \in \binom{V(G)}{2}$ where $D \subseteq E(G)$ and $A \subseteq \overline{E(G)}$. The sets D and A are called a deletion set and an addition set, respectively, corresponding to the solution $D \cup A$.

15.1.1 Characterizing the solution.

Starting point of our algorithm is a characterization of solution in terms of a deletion set D satisfying certain properties (such deletion sets are called nice deletion set). This, allows us to focus on finding nice deletion sets. To describe the main steps and ideas involved in our algorithm we first give a semi-informal definition of a nice deletion set. Towards this we need the definition of following two sets .

$$\begin{aligned} \text{def}(G, f) &= \{v \mid v \in V(G), f(v) > d_G(v)\} - \text{set of deficient vertices} \\ \mathbf{S}(G, f) &= \{v(i) \mid v \in V(G), f(v) > d_G(v), i \in \{1, \dots, f(v) - d_G(v)\}\} \end{aligned}$$

The second set specifies for every vertex $v \in \text{def}(G, f)$ how many edges in addition set must be adjacent to v . Let $\psi : \mathbf{S}(G, f) \rightarrow \mathbf{S}(G, f)$ be a bijection. Given ψ , we define a multiset E_ψ as follows. For each $u(i) \in \mathbf{S}(G, f)$ we add (u, v) to E_ψ if $\psi(u(i)) = v(j)$ for some $j \geq i$. Essentially, the map ψ will allow us to get our addition set A . We say that ψ is a *proper deficiency map* if ψ is involution, for any $u \in V(G)$ $(u, u) \notin E_\psi$, E_ψ is a set and not a multiset and $E_\psi \cap E(G) = \emptyset$. Finally, a set $D \subseteq E(G)$ is called *nice deletion set* if

- (i) for all $v \in V(G)$, $d_{G-D}(v) \leq f(v)$ and $|\mathbf{S}(G - D, f)| = 2(k - |D|)$;
- (ii) $G - D$ has at most $k - |D| + 1$ connected components;

- (iii) each connected component in $G - D$ contains a vertex v such that $d_{G-D} < f(v)$ (i.e, for each connected component F in $G - D$, $V(F) \cap \text{def}(G - D, f) \neq \emptyset$); and
- (iv) there exists a proper deficiency map $\psi : \mathcal{S}(G - D, f) \rightarrow \mathcal{S}(G - D, f)$.

Let $D \subseteq E(G)$. Then there exists $A \subseteq \overline{E(G)}$, $|A| = k - |D|$ such that $A \cup D$ is a solution to ECG if and only if D is a nice deletion set. Furthermore, given a nice deletion set we can find an addition set, if exists, in polynomial time. Thus, our problem reduces to finding a nice deletion set $D \subseteq E(G)$ and an accompanying proper deficiency map ψ on $\mathcal{S}(G - D, f)$, if it exists, using dynamic programming (DP).

15.1.2 Towards the states of DP algorithm.

The next question that arises is: how are we going to find a nice deletion set D ? Throughout this section we will work with a hypothetical deletion set D . We start with coloring the vertices of G , *green* and *red* in the following way. We color $v \in V(G)$ green if $d_G(v) > f(v)$, otherwise we color v red. Let $E_r = E(G[\text{Red}])$ and $E_g = E(G) \setminus E_r$. One first does a quick sanity check. That is, if $\sum_{\{v: d_G(v) \neq f(v)\}} |d_G(v) - f(v)| > 2k$ then we output NO, because in this case any solution to ECG requires more than k edge edits (addition/deletion operations). Now we guess the size $k' \leq k$ of D such that $2k' \geq \sum_{v: d_G(v) > f(v)} d_G(v) - f(v)$. Since D is our hypothetical deletion set, we have that for any $v \in \text{Green}$, the number of edges in D which are incident to v is at least $d_G(v) - f(v)$. Now we guess the number k_1 of edges in D which are incident to only green vertices and the number k_2 of edges in D which are incident to at least one vertex in Red. Note that $k_1 + k_2 = k'$. Also note that the number of ways we can guess (k', k_1, k_2) is at most k^2 . Now for every $v \in \text{Green}$, we guess the number of edges in D which are incident to v . In particular, we guess a function $\Phi : \text{Green} \rightarrow \mathbb{N}$ such that for all $v \in \text{Green}$ we have that $\Phi(v) \geq d_G(v) - f(v)$. The number of possible functions $\Phi(v)$ is upper bounded by $\mathcal{O}(4^k k)$. From now onwards we will assume that we are given a Φ . In other words we have guessed the function Φ corresponding to the hypothetical solution D . We say that a solution $D \cup A$ to ECG satisfies the function Φ if for every vertex $v \in \text{Green}$ the number of edges incident to v in D is exactly equal to $\Phi(v)$.

We start with an intuitive explanation of the structure of the solution that helps us in designing partial solution for the DP algorithm we are trying to give for our problem. Given $D \cup A$, we first define a notion of an alternating walk. An *alternating walk* is a sequence of vertices u_1, u_2, \dots, u_ℓ such that consecutive pairs $((u_i, u_{i+1}), (u_{i+1}, u_{i+2}))$ either belong to $D \times A$ or $A \times D$. That is, an edge from D is followed by an edge from A or vice-versa. In an *alternating even length closed walk*, $u_1 = u_\ell$ and ℓ is even. One might wonder about the definition of *alternating odd length closed walk*. For our purposes we will think of them as alternating walks that start and end at the same vertex. Essentially, these will be alternating walks that start and end with the same vertex and the first and the last edge either both belong to D or both belong

to A . From now onwards whenever we say an alternating closed walk, we mean an alternating even length closed walk. For any intermediate vertex in the alternating walk or in the alternating closed walk, one of the edges incident to it belongs to D and the other edge belongs to A . Thus the degree of any intermediate vertex is not disturbed either by an alternating walk or alternating closed walk. Our reasons to define these notions are as follows. Let $D \cup A$ be a solution of ECG that satisfies Φ . We can think of edges in $D \cup A$ forming a *family*, \mathcal{P} , of *alternating walks and alternating closed walks* with the following properties.

- For every vertex $v \in V(G)$ and a set $Z \in \{D, A\}$, we define $\text{apdeg}(\mathcal{P}, Z, v)$ as the number of edges from Z that are incident to v and appear as (i) the first edge in alternating walks from \mathcal{P} that start with v ; and (ii) the last edge in alternating walks from \mathcal{P} that end in v . Note that, if there is an alternating walk that both starts and ends in v and the start edge as well as the last edge belong to Z then this path contributes two to $\text{apdeg}(\mathcal{P}, Z, v)$. For every vertex $v \in \text{Green}$, $\text{apdeg}(\mathcal{P}, D, v) = d_G(v) - f(v)$ and $\text{apdeg}(\mathcal{P}, A, v) = 0$. Furthermore, for every vertex $v \in \text{Red}$, $\text{apdeg}(\mathcal{P}, D, v) = 0$ and $\text{apdeg}(\mathcal{P}, A, v) = f(v) - d_G(v)$.
- Every vertex $v \in \text{Green}$, the number of times it appear as an intermediate vertex in an alternating walk or in an alternating closed walk of \mathcal{P} is exactly equal to $\Phi(v) - (d_G(v) - f(v))$.

This path system view allows us to make a dynamic programming algorithm where we can move from one state to another using one edge addition or deletion. In particular, the algorithm works by constructing all alternating walks P_1, \dots, P_η first and then construct alternating closed walks $P_{\eta+1}, \dots, P_\alpha$. Given a partially constructed path system we try to append an edge to the current path we are constructing by adding an edge to it; or declaring that we are finished with the current path system and move to obtain a new path. During this process we also keep a partial proper deficiency map ψ' such that $E_{\psi'}$ is the addition edges in the current partial solution. Thus, a state in the dynamic programming algorithm is given by our current guesses and a subset of domain of partial proper deficiency map. It can be shown that the number of states is upper bounded by $2^{\mathcal{O}(k)}$. However, the number of sets $D' \cup A'$ that could satisfy the prerequisite of being in a particular table entry could be as large as $n^{\mathcal{O}(k)}$ and thus this would not lead to an FPT algorithm. However, this is indeed a template for our algorithm. Next we observe that we can prune the table size to $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ and thus lead to an FPT algorithm.

15.1.3 Pruning the DP table entry and an FPT algorithm.

We need to prune the DP table in a way that we do not change the answer to the given instance (G, f, k) . Towards this we show that if some subset we have stored in a DP table entry could lead to a nice deletion set then we do have at least one such set after the pruning operation. Our guessing of k_1, k_2 and ϕ allows us to satisfy the

property (i) of nice deletion set. The property (ii) of nice deletion set imply that D is an independent set in the matroid $M_G^*(\ell)$. That is, ℓ -elongation of the co-graphic matroid, M_G^* , associated with G , where $\ell = |E(G)| - |V(G)| + k - |D| + 1$ (we refer to preliminaries for the definition). Thus, by only considering those D which are independent sets in $M_G^*(\ell)$ we ensure that property (ii) of the nice deletion set is satisfied. Now consider the property (iv) of the nice deletion set, i.e, there exists a proper deficiency map $\psi : S(G - D, f) \rightarrow S(G - D, f)$. Our objective is to get a set $D \cup A$ such that there is a proper deficiency map ψ over $S(G - D, f)$ such that $E_\psi = A$, along with other properties as well. Let $D_1 \cup A_1, D_2 \cup A_2$ be two partial solutions belonging to the same equivalence class where $D_1, D_2 \subseteq E(G)$ and $A_1, A_2 \subseteq \overline{E(G)}$. Suppose $D' \subseteq E(G), A' \subseteq \overline{E(G)}, (D_1 \cup D') \cup (A_1 \cup A')$ is a solution and $A_2 \cap A' = \emptyset$. Since $D_1 \cup A_1, D_2 \cup A_2$ belongs to same equivalence class and $A_2 \cap A'$ is disjoint, there is a proper deficiency map ψ' over $S(G - (D_2 \cup D'), f)$ such that $E_{\psi'} = A_2 \cup A'$. To take care of the disjointness property between the current addition set and the future addition set while doing the DP, we *view the the addition set A of the solution as an independent set in a uniform matroid over the universe $\overline{E(G)}$* . Let $U_{m', k-k'}$ be a uniform matroid with ground set $\overline{E(G)}$, where $m' = |\overline{E(G)}|$. From the definition of $U_{m', k-k'}$, any set A of size at most $k - k'$ is independent in $U_{m', k-k'}$. We have already explained that we view the deletion set D as an independent set in $M_G^*(\ell)$ where $\ell = |E(G)| - |V(G)| + k - k' + 1$. Thus, to see the solution set $D \cup A$ as an independent set in a single matroid, we consider direct sum of $M_G^*(\ell)$ and $U_{m', k-k'}$. That is, let $M = M_G^*(\ell) \oplus U_{m', k-k'}$. In M , a set I is an independent set if and only if $I \cap E(G)$ is an independent set in $M_G^*(\ell)$ and $I \cap \overline{E(G)}$ is an independent set in $U_{m', k-k'}$. This ensures that any solution $D \cup A$ is an independent set in M . By viewing any solution of the problem as an independent set in a matroid M (which is linear), we can use the q -representative families to prune the table. However, we still need to ensure that property (iii) of nice deletion set is satisfied. In what follows we explain how we achieve this.

We show that for every deletion set D there exists a set of edges $W_D \subseteq E(G)$ (disjoint from D) of size at most $6k$ such that if these edges are not picked then we can guarantee that each connected component in $G - D$ contains a vertex v such that $d_{G-D} < f(v)$. To achieve this we apply color coding. That is, if we randomly color each edge orange with probability $6/7$ and black with probability $1/7$, then with probability at least $(1/7)^k (6/7)^{6k}$, all the edges of the deletion set D will be colored with orange and all the edges in W_D will be colored with black. If we repeat our algorithm $(7^{7k}/6^{6k})$ times we can increase the success probability to a constant. This step can be derandomized using universal sets.

15.2 Algorithm

For a given graph G , a solution to ECG comprises a *deletion set* $D \subseteq E(G)$ and an *addition set* $A \subseteq \overline{E(G)}$. That is, $F = G - D + A$. In particular, we denote the solution set as a pair (D, A) , where $D \subseteq E(G)$ and $A \subseteq \overline{E(G)}$.

15.2.1 Structural Characterization

In this subsection we give a structural characterization of solution that is central to our parameterized algorithm. In particular we give a necessary and sufficient conditions on $D \subseteq E(G)$ for being a deletion set of an optimum solution to ECG. Furthermore, we show that if we have D that satisfies the desired conditions then we can obtain the corresponding addition set A in polynomial time. We start with few definitions that will setup a useful language to speak about different aspects of the sets we will be considering throughout the paper.

Definition 15.1. *Let G be a graph and $f : V(G) \rightarrow \{1, 2, \dots, d\}$ be a function. We call a vertex $v \in V(G)$ **deficient** with respect to G and f , if $f(v) > d_G(v)$. We define **deficiency vertex set** of G and f , denoted by $\text{def}(G, f)$, as the set of deficient vertices with respect to G and f . We define **deficiency set** of G and f , denoted by $\mathbf{S}(G, f)$, as a set containing $\max\{0, f(v) - d_G(v)\}$ vertices corresponding to each vertex $v \in V(G)$. Notice that $\mathbf{S}(G, f)$ contains non-zero vertices corresponding to a vertex v only if $f(v) > d_G(v)$ and the cardinality of $\mathbf{S}(G, f)$ is equal to $\sum_{\{v: f(v) > d_G(v)\}} f(v) - d_G(v)$. In particular these sets are defined as:*

$$\begin{aligned} \text{def}(G, f) &= \{v \mid v \in V(G), f(v) > d_G(v)\} \\ \mathbf{S}(G, f) &= \{v(i) \mid v \in V(G), f(v) > d_G(v), i \in [f(v) - d_G(v)]\} \end{aligned}$$

Let $W \subseteq V(G) \times \mathbb{N}^+$. Recall that we denote a pair $(v, i) \in V(G) \times \mathbb{N}^+$ (or in W) by $v(i)$. Let $\psi : W \rightarrow W$ be a bijection. Given ψ , we define a multiset E_ψ as follows. For each $u(i) \in W$ we add (u, v) to E_ψ if $\psi(u(i)) = v(j)$ for some $j \geq i$.

Definition 15.2. *Let G be a graph and $W \subseteq V(G) \times \mathbb{N}^+$. A bijection $\psi : W \rightarrow W$ is called a **proper deficiency map** if it satisfies the following properties.*

1. **Involution:** *For any $x \in W$, $\psi(\psi(x)) = x$.*
2. **Non-Loop Property:** *For any $u \in V(G)$, $(u, u) \notin E_\psi$*
3. **Simple Edge Property:** *E_ψ is a set, not a multiset. That is, there does not exist $u, v \in V(G)$ such that $\psi(u(i_1)) = v(j_1)$ and $\psi(u(i_2)) = v(j_2)$ for some $i_1 \neq i_2$ and $j_1 \neq j_2$.*
4. **Non-Edge Property:** *$E_\psi \cap E(G) = \emptyset$.*

In general we will have proper deficiency map over the domain $\mathbf{S}(G, f)$ or some set related to this. Finally, we define a notion of *nice deletion set*.

Definition 15.3. *Let (G, f, k) be an instance of ECG and let $D \subseteq E(G)$. We say that D is a **nice deletion set** if the following properties are satisfied.*

- (i) *For all $v \in V(G)$, $d_{G-D}(v) \leq f(v)$.*
- (ii) *$|\mathbf{S}(G - D, f)| = 2(k - |D|)$.*
- (iii) *$G - D$ has at most $k - |D| + 1$ connected components.*
- (iv) *Each connected component in $G - D$ contains a vertex v such that $d_{G-D} < f(v)$ (i.e, for each connected component F in $G - D$, $V(F) \cap \text{def}(G - D, f) \neq \emptyset$).*
- (v) *There exists a proper deficiency map $\psi : \mathbf{S}(G - D, f) \rightarrow \mathbf{S}(G - D, f)$.*

Lemma 15.1. *Let (G, f, k) be an instance of ECG and let $D \subseteq E(G)$. Then there exists $A \subseteq \overline{E(G)}$, $|A| = k - |D|$ such that $A \cup D$ is a solution to ECG if and only if D is a nice deletion set. Moreover, given a nice deletion set $D \subseteq E(G)$ we can find $A \subseteq \overline{E(G)}$ such that $|A| = k - |D|$ and $D \cup A$ is a solution to ECG in polynomial time.*

Proof. (\Rightarrow) Let $A \subseteq \overline{E(G)}$, $|A| = k - |D|$ such that $A \cup D$ is a solution to ECG. We need to show that $D \subseteq E(G)$ is a nice deletion set. Since $A \cup D$ is a solution to ECG, we have that $d_{G-D}(v) \leq f(v)$ for all $v \in V(G)$, satisfying condition (i). Furthermore, $A \cup D$ being a solution also implies that $\sum_{\{v : f(v) > d_{G-D}(v)\}} f(v) - d_{G-D}(v) = 2|A| = 2(k - |D|)$. Hence $|\mathbf{S}(G - D, f)| = 2(k - |D|)$, satisfying condition (ii). Since $G - D + A$ is a connected graph, $G - D$ can have at most $|A| + 1 = k - |D| + 1$ connected components, satisfying condition (iii) in the definition. The graph $G - D + A$ is connected and thus each connected component F in $G - D$ contains a vertex $v \in V(F)$ such that $(v, u) \in A$ for some $u \in V(G)$. Since $D \cup A$ is a solution to ECG, $d_{G-D+A}(v) = f(v)$ and hence $d_{G-D}(v) < f(v)$ (because $(v, u) \in A$), satisfying condition (iv). Finally, we show that D satisfies the last property. Let $A = \{e_1, e_2, \dots, e_r\} \subseteq \overline{E(G)}$ where $r = k - |D|$. Since $D \cup A$ is a solution to ECG,

we have that for any vertex v , there are exactly $f(v) - d_{G-D}(v)$ edges in A which are adjacent to v . Now we define a bijection $\psi : \mathcal{S}(G - D, f) \rightarrow \mathcal{S}(G - D, f)$ as follows: $\psi(u(i)) = v(j)$ if $(u, v) = e_\ell$ such that there are exactly $i - 1$ edges from $\{e_1, \dots, e_{\ell-1}\}$ are incident on u and there are exactly $j - 1$ edges from $\{e_1, \dots, e_{\ell-1}\}$ are incident on v . That is, we traverse the edges e_1, \dots, e_r from left to right and find the i^{th} edge incident to u , say $e_\ell = (u, v)$, and then we assign it $v(j)$, if there are exactly j edges incident to v present among $\{e_1, \dots, e_{\ell-1}\}$.

Claim 15.1. $\psi : \mathcal{S}(G - D, f) \rightarrow \mathcal{S}(G - D, f)$ is a proper deficiency map.

Proof. By the definition of ψ , if $\psi(u(i)) = v(j)$ then $\psi(v(j)) = u(i)$, and so ψ is an involution. Since $G - D + A$ is a simple graph, for any $u \in V(G)$, $(u, u) \notin E_\psi$. Now we need to show that E_ψ is not a multiset. Suppose not, then there exists $u, v \in V(G)$ such that $\psi(u(i_1)) = v(j_1)$ and $\psi(u(i_2)) = v(j_2)$ for some $i_1 \neq i_2$ and $j_1 \neq j_2$. This implies that there exist $e_i, e_j \in A$, $i \neq j$ such that $e_i = e_j = (u, v)$. This contradicts the fact that $G - D + A$ is a simple graph. Since A is disjoint from $E(G)$, $E_\psi \cap E(G) = \emptyset$. This completes the proof. \square

(\Leftarrow) Let $D \subseteq E(G)$ be a nice deletion set. We need to show that we can find $A \subseteq \overline{E(G)}$ such that $|A| = k - |D|$ and $G - D + A$ is a solution to ECG. Properties (i) and (ii) imply that $d_{G-D}(v) \leq f(v)$ for all $v \in V(G)$ and $|\mathcal{S}(G - D, f)| = 2(k - |D|)$. Due to the property (v) in the definition of nice deletion set, we know that there exists a proper deficiency map $\psi : \mathcal{S}(G - D, f) \rightarrow \mathcal{S}(G - D, f)$. Define $A_1 = E_\psi$. By the definition of E_ψ , $|A_1| = |E_\psi| = |\mathcal{S}(G - D, f)|/2 = k - |D|$. Also note that $A_1 \cap E(G) = \emptyset$ because ψ is a proper deficiency map. Now consider the graph $G_1 = G - D + A_1$. Note that G_1 is simple graph because ψ is a proper deficiency map. Also by the definition of E_ψ , $d_{G-D+A_1}(v) = f(v)$ for all $v \in V(G)$. So G_1 satisfies the degree constraints. If G_1 is connected then $D \cup A_1$ is a solution to ECG. Thus, we assume that G_1 is not connected. In what follows we give an iterative procedure that finds the desired A . Suppose we are in i^{th} iteration and we

have a set A_i such that $G - D + A_i$ satisfies all degree constraints but $G - D + A_i$ is not connected. Then in the next iteration we find another addition set of size $k - |D|$, say A_{i+1} , such that $G - D + A_{i+1}$ satisfies all degree constraints and $G - D + A_{i+1}$ has strictly less number of connected components than in $G - D + A_i$. The procedure is started with $A_1 = E_\psi$. Let $i \geq 1$ and we have A_i such that $G - D + A_i$ satisfies all degree constraints but $G - D + A_i$ is not connected. Since $|A_i| = k - |D|$ and $G - D$ has at most $k - |D| + 1$ connected components, $G_i = G - D + A_i$ has a component F such that there is an edge $(u_1, v_1) \in A_i$ with the property that $u_1, v_1 \in V(F)$ and (u_1, v_1) is not a bridge in F . Let F' be another connected component in G_i . Since each connected component in $G - D$ contains a vertex $w \in V(G)$ such that $d_{G-D}(w) < f(w)$, there exists an edge $(u_2, v_2) \in A_i$ such that $u_2, v_2 \in V(F')$. Now let $A_{i+1} = (A_i \setminus \{(u_1, v_1), (u_2, v_2)\}) \cup \{(u_1, u_2), (v_1, v_2)\}$. Observe that $G_{i+1} = G - D + A_{i+1}$ is a simple graph with strictly less connected component than in G_i and $d_{G_{i+1}}(v) = f(v)$ for all $v \in V(G)$. Observe that when the procedure stops we would have found the desired A .

Given a nice deletion set D , we can find the desired A using the iterative procedure described in the reverse direction of the proof. Clearly, this procedure can be implemented in polynomial time. This completes the proof of the lemma. \square

15.2.2 An algorithm with running time $n^{\mathcal{O}(k)}$

In this section we design an algorithm for ECG running in time $n^{\mathcal{O}(k)}$ – an XP algorithm. Clearly, this is not the algorithm we promised. In fact a simple brute force algorithm for ECG will run in time $n^{\mathcal{O}(k)}$. But this algorithm will provide a skeleton for a parameterized algorithm given in the next subsection. Both the algorithms, XP as well as FPT for ECG, are based on dynamic programming (DP). The algorithm given in this section allows us to introduce various aspects of the dynamic programming in a gentler manner. Even though the number of states in the dynamic programming algorithm given in this subsection is bounded by $c^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, the number of partial solutions stored in a DP table entry could potentially be $n^{\mathcal{O}(k)}$. The FPT algorithm given in the next subsection is based on this algorithm and uses tools from representative family to reduce the cardinality of partial solutions stored in any DP table entry to $c^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

Given an instance (G, f, k) of ECG. The main idea of the algorithm is to find a nice deletion set $D \subseteq E(G)$ and an accompanying proper deficiency map ψ on $S(G - D, f)$, if it exists, using dynamic programming. **Throughout this section we will work with a hypothetical deletion set D .** We start with coloring the vertices of G , *green* and *red* in the following way. We color $v \in V(G)$ green if $d_G(v) > f(v)$, otherwise we color v red. That is, $\mathbf{Green} = \{v \mid v \in V(G), d_G(v) > f(v)\}$ and $\mathbf{Red} = V(G) \setminus \mathbf{Green}$. Let $E_r = E(G[\mathbf{Red}])$ and $E_g = E(G) \setminus E_r$. One first does a quick sanity check. That is, if $\sum_{\{v: d_G(v) \neq f(v)\}} |d_G(v) - f(v)| > 2k$ then we output NO, because in this case any solution to ECG requires more than k edge edits (addition/deletion operations). Now we guess the size $k' \leq k$ of D such that $2k' \geq \sum_{v: d_G(v) > f(v)} d_G(v) - f(v)$. Since D is our hypothetical deletion set, we have that for any $v \in \mathbf{Green}$, the number of edges in D which are incident to v is at least $d_G(v) - f(v)$. Now we guess the number k_1 of edges in D which are incident to only green vertices and the number k_2 of edges in D which are incident to at least one vertex in \mathbf{Red} . Note that $k_1 + k_2 = k'$. Also note that the number of ways we can guess (k', k_1, k_2) is at most k^2 . Now for every $v \in \mathbf{Green}$, we guess the number of edges in D which are incident to v . In particular, we guess a function $\Phi : \mathbf{Green} \rightarrow \mathbb{N}$ such that for all $v \in \mathbf{Green}$ we have that $\Phi(v) \geq d_G(v) - f(v)$. We claim that the number of possible functions from \mathbf{Green} to \mathbb{N} , that represent the number of edges incident to a vertex $v \in \mathbf{Green}$ in the hypothetical solution D , is bounded by $\mathcal{O}(4^k k)$. Let k_3 be the number of edges in D which are incident to only red vertices. Observe that we are looking for functions Φ , such that for all $v \in \mathbf{Green}$ we have that $1 \leq d_G(v) - f(v) \leq \Phi(v) \leq k_1 + k_2 - k_3$, and $\sum_{v \in \mathbf{Green}} \Phi(v) \leq 2k_1 + k_2 - k_3$. The number of such functions is upper bounded by

$$\mathcal{O}(2^{|\mathbf{Green}| + 2k_1 + k_2 - k_3 - \sum_{v \in \mathbf{Green}} d_G(v) - f(v)}).$$

The above quantity is upper bounded by $\mathcal{O}(4^k)$ because $|\mathbf{Green}| \leq \sum_{v \in \mathbf{Green}} d_G(v) - f(v)$ and $2k_1 + k_2 - k_3 \leq 2k$. Since there are at most k possible values for k_3 , the number of possible functions $\Phi(v)$ is upper bounded by $\mathcal{O}(4^k k)$. From now onwards we will assume that we are given a Φ . In other words we have guessed the function Φ corresponding to the hypothetical solution D . We say that a solution $D \cup A$ to ECG satisfies the function Φ if for every vertex $v \in \mathbf{Green}$ the number of edges incident to v in D is exactly equal to $\Phi(v)$.

Intuitive Structure of the Algorithm. We start with an intuitive explanation of the structure of the solution that helps us in designing partial solution for the DP algorithm we are trying to give for our problem. Any solution to our problem is of the form $D \cup A \in \binom{V(G)}{2}$ where $D \subseteq E(G)$ and $A \subseteq \overline{E(G)}$. Given $D \cup A$, we first define a notion of an alternating walk. An *alternating walk* is a sequence of vertices u_1, u_2, \dots, u_ℓ such that consecutive pairs $((u_i, u_{i+1}), (u_{i+1}, u_{i+2}))$ either belong to $D \times A$ or $A \times D$. That is, an edge from D is followed by an edge from A or vice-versa. In an *alternating even length closed walk*, $u_1 = u_\ell$ and ℓ is even. One might wonder about the definition of *alternating odd length closed walk*. For our purposes we will think of them as alternating walks that start and end at the

same vertex. Essentially, these will be alternating walks that start and end with the same vertex and the first and the last edge either both belong to D or both belong to A . From now onwards whenever we say an alternating closed walk, we mean an alternating even length closed walk. For any intermediate vertex in the alternating walk or in the alternating closed walk, one of the edges incident to it belongs to D and the other edge belongs to A . Thus the degree of any intermediate vertex is not disturbed either by an alternating walk or alternating closed walk. Our reasons to define these notions are as follows. Let $D \cup A$ be a solution of ECG that satisfies Φ . We can think of edges in $D \cup A$ forming a *family*, \mathcal{P} , of *alternating walks and alternating closed walks* with the following properties.

- For every vertex $v \in V(G)$ and a set $Z \in \{D, A\}$, we define $\text{apdeg}(\mathcal{P}, Z, v)$ as the number of edges from Z that are incident to v and appear as (i) the first edge in alternating walks from \mathcal{P} that start with v ; and (ii) the last edge in alternating walks from \mathcal{P} that end in v . Note that, if there is an alternating walk that both starts and ends in v and the start edge as well as the last edge belong to Z then this path contributes two to $\text{apdeg}(\mathcal{P}, Z, v)$. For every vertex $v \in \text{Green}$, $\text{apdeg}(\mathcal{P}, D, v) = d_G(v) - f(v)$ and $\text{apdeg}(\mathcal{P}, A, v) = 0$. Furthermore, for every vertex $v \in \text{Red}$, $\text{apdeg}(\mathcal{P}, D, v) = 0$ and $\text{apdeg}(\mathcal{P}, A, v) = f(v) - d_G(v)$.
- Every vertex $v \in \text{Green}$, the number of times it appear as an intermediate vertex in an alternating walk or in an alternating closed walk of \mathcal{P} is exactly equal to $\Phi(v) - (d_G(v) - f(v))$.

For any solution $\mathcal{P} = \{P_1, P_2, \dots, P_\alpha\}$, without loss of generality we assume that there is η such that P_1, \dots, P_η are alternating walks and $P_{\eta+1}, \dots, P_\alpha$ are alternating closed walks. *In our solution we will first construct all alternating walks and then construct all alternating closed walks. Also for any alternating closed walk, we always start with a deletion edge.*

Towards an implementation of the intuitive description. Our objective is to design a DP algorithm. Thus, we first need to define a notion of partial solution which will constitute basic building block of our algorithm. We first explain about partial solutions and its structure which will be utilized to design the algorithm. Any solution to our problem is of the form $D \cup A \in \binom{V(G)}{2}$ where $D \subseteq E(G)$ and $A \subseteq \overline{E(G)}$, thus the partial solution for the problem is also a subset $B \cup A' \in \binom{V(G)}{2}$ where $B \subseteq E(G)$ and $A' \subseteq \overline{E(G)}$. Let $D \cup A$ be a solution of ECG that satisfies Φ . As described before we think of edges in $D \cup A$ forming a family, $\mathcal{P} = \{P_1, \dots, P_s\}$, of alternating walks and alternating closed walks. A partial solution can be thought of as $\{P_1, \dots, P_j^*\}$, where we have already created P_1, \dots, P_{j-1} and P_j^* is some subwalk of P_j that we are creating now. This view could be useful in understanding the algorithm we are going to describe later. *At this point we add a caveat that our algorithm slightly differs from this perspective to make the proof more accessible.*

Let $\mathcal{P}' = \{P_1, \dots, P_j^*\}$ be a partial solution. We first assume that P_j^* that we are constructing is going to be an alternating walk. For every vertex $v \in V(G)$ and a set $Z \in \{B, A'\}$, we define $\text{apdeg}^*(\mathcal{P}', Z, v)$ as the number of edges from Z that are incident to v and appear as (i) the first edge in alternating walks from \mathcal{P}' that start with v ; and (ii) the last edge in alternating walks from $\{P_1, \dots, P_{j-1}\}$ that end in v . As before if there is an alternating walk that both starts and ends in v and the start edge as well as the last edge belong to Z then this path contributes two to $\text{apdeg}^*(\mathcal{P}', Z, v)$. If P_j^* that we are constructing is going to be an alternating closed walk then, $\text{apdeg}^*(\mathcal{P}', Z, v) = \text{apdeg}(\{P_1, \dots, P_{j-1}\}, Z, v)$. Before we go further we would like to add that while making an algorithm we will know whether we are currently constructing an alternating walk or an alternating closed walk.

In our algorithm we form partial solutions of size i from partial solutions of size $i - 1$. It is important, for designing the FPT algorithm given in the next subsection, to partition the set of partial solutions based on some of its structures. Note that we have already guessed some structure of the solution: like the number of deletion edges, the number of deletion edges with both end points in **Green**, the number of addition edges and the number of edges deleted from each vertex $v \in \mathbf{Green}$ (via guessing the function Φ). We use these structures to characterize the equivalence classes of partial solutions. Since we are going to compute a solution which respects Φ , in the description of a equivalence class over partial solution we would like to include for every vertex $v \in \mathbf{Green}$ the number of edges that are incident to v and contribute to $\text{apdeg}^*(\mathcal{P}, D, v)$. This is achieved by creating a multiset set T_m containing $d_G(v) - f(v)$ many copies of v for all $v \in \mathbf{Green}$ and keeping a subset of T_m' for each equivalence class. That is, T_m' tells us how many edges incident to a vertex $v \in \mathbf{Green}$ must be present in the partial solutions that respect $\text{apdeg}^*(\mathcal{P}', B, v)$. In other words, $\text{apdeg}^*(\mathcal{P}', B, v) = T_m'(v)$.

Now we define another set T_g for **Green** vertices that stores the information about how many times a vertex v appears as an intermediate vertex in the current partial solution $\mathcal{P}' = \{P_1, \dots, P_j^*\}$. In particular we define

$$T_g = \{ v(i) \mid v \in \mathbf{Green}, \Phi(v) > d_G(v) - f(v), i \in [\Phi(v) - (d_G(v) - f(v))] \}.$$

We use $T_g' \subseteq T_g$ to represent an equivalence class with the property that a partial solution $\mathcal{P}' = \{P_1, \dots, P_j^*\}$ satisfies that for every vertex $v \in \mathbf{Green}$ the number of times v appear as an intermediate vertex in \mathcal{P}' is same as the number of elements corresponding to it in T_g' . Observe that we are differentiating between elements corresponding to a vertex and a copy of a vertex. For example consider a vertex v . When we say that T_m contains 3 copies of a vertex v , we mean $\{v, v, v\} \subseteq T_m$ and when we say T_g contains three elements/vertices corresponding to v , then we mean that $\{v(1), v(2), v(3)\} \subseteq T_g$.

We have taken care of all alternating walks that start or end with deletion edges as well as alternating closed walk. We also have some alternating walks that start or end with an addition edge. Now we define some sets that will help us to have some control over these alternating walks. Towards this we define a set T_r as follows: T_r

is a set such that for any $v \in \text{Red}$ there are exactly $f(v) - d_G(v)$ elements in T_r corresponding to v .

$$T_r = \{ v(i) \mid v \in \text{Red}, f(v) > d_G(v), i \in [f(v) - d_G(v)] \}$$

Ideally, we would like to keep a set $T'_r \subseteq T_r$ to represent an equivalence class with the property that a partial solution $\mathcal{P}' = \{P_1, \dots, P_j^*\}$ satisfies that for every vertex $v \in \text{Red}$, $\text{apdeg}^*(\mathcal{P}', A', v) = T'_r(v)$. However, for technical reasons we represent it as follows. For every vertex $v \in \text{Red}$, $\text{apdeg}^*(\mathcal{P}', A', v) = (T_r \setminus T'_r)(v)$.

Let v be the last vertex of P_j^* . When we are constructing P_j^* it can happen that the edge incident to v could either a deletion edge or an addition edge. In either case we do not know whether v is the last vertex of P_j^* and thus we can not store the information on v either using T'_g or T'_r . To overcome this difficulty, we keep a multiset X of size at most 2 or a set Y of size at most one. If the last edge is an addition edge then we store v in Y and if the last edge is a deletion edge then we store v in X . Now we explain why X is a multiset and $|X| \leq 2$. If P_j^* is going to be an alternating closed walk then in fact we need to store information about both of its end vertices. In other words if P_j^* is going to be an alternating closed walk then P_j^* starts with a deletion edge and the information about its starting vertex is stored in X . If P_j^* ends with deletion edge then the information about it is stored in X and the end vertex very well could be same as the starting vertex of P_j^* . Thus, X could be a multiset of size 2.

Informal Description of the algorithm. Our algorithm works as follows. First we construct all alternating walks P_1, \dots, P_η and then construct alternating closed walks $P_{\eta+1}, \dots, P_\alpha$. Now we explain how each alternating (closed) walk can be constructed. Suppose we have completed constructing P_{j-1} . We start constructing P_j in the following preference order.

1. If $T_m \neq T'_m$, then we start with a deletion edge incident on $v \in T_m \setminus T'_m$ and add a copy of v to T'_m . Let P_j^* denote the current partial alternating walk that we have constructed so far. If P_j^* ends in a vertex $u \in T_m \setminus T'_m$ with a deletion edge, then we say $P_j = P_j^*$ and add a copy of u to T'_m . If P_j^* ends in a vertex u with an addition edge and there is an element corresponding to u in T'_r , then we say $P_j = P_j^*$ and we delete an element corresponding to u from T'_r . Else, we continue constructing this alternating walk by either adding or deleting an appropriate edge incident to u . Notice that if u is a vertex in **Green** then the last edge of P_j^* will be accounted in the next step using T'_g . However, if $u \in \text{Red}$ then we do not account for these edges using any set.
2. If $T'_r \neq \emptyset$, then we start with an addition edge incident on a vertex v , where $T'_r(v) \neq 0$ and we delete an element corresponding to v from T'_r . Let P_j^* denote the current partial alternating walk that we have constructed so far. Now if P_j^* ends in a vertex $u \in T_m \setminus T'_m$ with a deletion edge, then we say $P_j = P_j^*$ and add a copy of u to T'_m . If P_j^* ends in a vertex u with an addition edge

and there is an element corresponding to u in T'_r , then we say $P_j = P_j^*$ and we delete an element corresponding to u from T'_r . Else, we continue constructing this alternating walk by either adding or deleting an appropriate edge incident to u .

3. If both the above cases are not applicable, then in fact $j - 1 \geq \eta$. That is, we have constructed all the alternating walks in the solution. Let $B \subseteq E(G)$ and $A' \subseteq \overline{E(G)}$ be the set of deletion edges and addition edges present in P_1, \dots, P_{j-1} . In fact at this point $d_{G-B+A'}(v) = f(v)$ for all $v \in V(G)$, $T_m = T'_m$, $T'_r = \emptyset$, $X = \emptyset$ and $Y = \emptyset$. So all the degree constraints are satisfied. But to make the resulting graph *connected* we might need to do more editing. Note that any alternating closed walk will not disturb the degree of any vertex. It is only used for connectivity purpose. We start construction of P_j^* by guessing a deletion edge (u, v) in the closed walk P_j and adding both its end vertices, x, y , to X . After this we continue following this trail until we hit x again.

In essence, we are constructing our alternating walks greedily (that is, we stop whenever we have chance to do so!).

A formal definition of partial solution. For our partial solution we would like to use Lemma 15.1. That is, we would like to obtain a nice deletion set. However, to completely characterize a nice deletion set, we also need a proper deficiency map. Note that for any $Z \subseteq E(G)$, $T_r \subseteq S(G - Z, f)$. For each partial solution $B \cup A'$, the subset $T'_r \subseteq T_r$ represents the following: the current partial proper deficiency map does not have T'_r as its domain. This is the main reason we defined T'_r differently than T'_m and T'_g . In other words T'_r denotes that we still “need to add certain number of edges” on vertices belonging to Red. However, note that we have a vertex $v \in X$ and the only reason this vertex is in X is that $f(v) > d_{G-B}(v)$. Thus, when we consider $S(G - B, f) \setminus T'_r$ then there is an element corresponding to v present in it. And we can not take care of this deficiencies using the current partial map for which the corresponding edge set is A' . To circumvent this we remove the newly added deficiencies from our domain. Towards this we define $X_{B,f}$ as follows. Let X be a multiset of size 2 such that for all $u \in X$, $f(u) > d_{G-B}(u) + X(u) - 1$. Then,

$$X_{B,f} = \{u(i) \mid u \in X, f(u) - d_{G-B}(u) - X(u) + 1 \leq i \leq f(u) - d_{G-B}(u), i \in \mathbb{N}^+\}$$

Note that $X_{B,f} \subseteq S(G - B, f)$. Similarly, when $Y \neq \emptyset$ then we know that the last operation was an edge addition incident on a vertex $w \in Y$. Thus to have a proper deficiency map ψ' such that $E_{\psi'} = A'$ we need to add Y to the domain of ψ' . For some partial solution $B \cup A'$, it can happen that there exists $v \in \text{Green}$ such that $d_{G-B}(v) \geq f(v)$ and $\overline{E}_G(v) \cap A' \neq \emptyset$. Thus to have a proper deficiency map ψ' such that $E_{\psi'} = A'$ we add the set T'_g corresponding to the partial solution $B \cup A'$ to the domain of ψ' . Thus for any partial solution $B \cup A'$ which are in an equivalence class characterized by $T'_g \subseteq T_g$, $T'_r \subseteq T_r$, X and Y , we will have a proper deficiency map ψ' over $(S(G - B, f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{B,f})$ such that $E_{\psi'} = A'$.

Now we formally define the notion of partial solutions. Given an instance (G, f, k) we define T_m, T_g and T_r as described earlier. Also, recall that we have k_1, k_2 and Φ . For any $T'_m \subseteq T_m, T'_g \subseteq T_g, T'_r \subseteq T_r, k'_1 \leq k_1, k'_2 \leq k_2, i \leq k$, a multiset X containing elements from $V(G)$ and $Y \subseteq V(G)$ such that $|X| \leq 2, |Y| \leq 1, |X \cup Y| \leq 2$ and $X \cap Y = \emptyset$, we define a family $\mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ of subsets of $\binom{V(G)}{2}$ as follows. For any $B \subseteq E(G)$ and $A \subseteq E(G)$, $B \cup A \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ if the following conditions are met:

- (a) $|E(G[\text{Green}]) \cap B| = k'_1, |B \setminus E(G[\text{Green}])| = k'_2$ and $|B \cup A| = i$.
- (b) For any $v \in \text{Green}$, the number of edges in B which are incident to v is exactly equal to $T'_m(v) + T'_g(v) + X(v)$. That is, for all $v \in \text{Green}$, $|B \cap E_G(v)| = T'_m(v) + T'_g(v) + X(v)$.
- (c) $|X_{B,f}| = |X|$ and $X_{B,f} \subseteq S(G - B, f) \setminus T_r$.
- (d) $G - B$ has at most $k - k' + 1$ connected components.
- (e) There is a proper deficiency map $\psi' : (S(G - B, f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{B,f}) \rightarrow (S(G - B, f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{B,f})$ such that $A = E_{\psi'}$. Furthermore, for $w \in Y$, if $\psi'(w) = u(j)$ for some j then $T_m(u) = T'_m(u)$.

In condition (e) we demanded the following: for $w \in Y$, if $\psi'(w) = u(j)$ for some j then $T_m(u) = T'_m(u)$. We explain the reason for doing this. In our algorithm if $Y \neq \emptyset$, then the last operation is an addition operation with an edge incident to w . If (u, w) is the edge added in the last operation then in the proper deficiency map ψ , w maps to $u(j)$ for some j . The only reason we did not stop at u is because either $u \in \text{Red}$ or $u \in \text{Green}$ and $T'_m(u) = T_m(u)$. Thus, in some sense this condition helps us in knowing when the current alternating walk we are constructing will stop.

For $T'_m \subseteq T_m, T'_g \subseteq T_g, T'_r \subseteq T_r, k'_1 \leq k_1, k'_2 \leq k_2, i \leq k$, a multiset X containing elements from $V(G)$ and $Y \subseteq V(G)$, we say that the tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ is a *valid* tuple if the following happens.

- (1) $|X| \leq 2, |Y| \leq 1, |X \cup Y| \leq 2$ and $X \cap Y = \emptyset$
- (2) For $w \in Y$, $w(j) \notin T'_r$ for all j .
- (3) If $u(j) \in T'_g$ then $u(j') \in T'_g$ for all $0 < j' < j$.
- (4) For any $v \in X$, $T_m(v) = T'_m(v)$.

For the correctness of the algorithm, it is enough to focus on partial solutions defined over valid tuples. We have already explained that the cardinality of $|X| \leq 2$ and $|Y| \leq 1$. Also note that if we have a partially constructed alternating (closed) walk then its current end vertex will be either in X or Y . If we are constructing an alternating closed walk then its starting vertex will also be in X . Because of this

$|X \cup Y| \leq 2$. When both X and Y are non empty then we are constructing an alternating closed walk starting at a vertex $x \in X$ and at present it ends at a vertex $w \in Y$. If $x = w$, then we could have greedily completed this closed walk. Hence, $X \cap Y \neq \emptyset$. For any set of partial solution $B \cup A \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$, we have a proper deficiency map ψ over $(S(G - B, f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{B,f})$. If $u(j) \in S(G - B, f)$ then $u(j') \in S(G - B, f)$ for all $j' \leq j$. Since T'_g also accounts for the number of edges deleted from each green (along with T'_m and X), the condition (3) of the valid tuple is a sanity check. Conditions (2) and (4) are another set of sanity checks which we have already explained.

Now we prove that in fact $\mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ is “a correct notion of partial solution”.

Lemma 15.2. *Let (G, f, k) be an YES instance of ECG with a solution $D \cup A$ such that $D \subseteq E(G)$, $A \subseteq \overline{E(G)}$, $|D \cap E(G[\text{Green}])| = k_1$, $|D \setminus E(G[\text{Green}])| = k_2$, $k_1 + k_2 = k'$ and $|D \cap E_G(v)| = \Phi(v)$ for all $v \in \text{Green}$. Let ψ be a proper deficiency map over $S(G - D, f)$ such that $E_\psi = A$. Then for each $i \leq k$, there exists $D' \cup A' \subseteq D \cup A$ and a valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ such that $D' \cup A' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ and there is a proper deficiency map ψ' over $R = (S(G - D', f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{D',f})$ with the property that $E_{\psi'} = A'$*

Proof. We prove the lemma by induction on i . For $i = 0$ we set $D', A' = \emptyset$ and so $D' \cup A' \in \mathcal{Q}(\emptyset, \emptyset, T_r, 0, 0, 0, \emptyset, \emptyset)$. It is obvious to see that $D' \cup A'$ satisfy the conditions (a), (b), (c) and (e) of being in the family $\mathcal{Q}(\emptyset, \emptyset, T_r, 0, 0, 0, \emptyset, \emptyset)$. Since $D \cup A$ is a solution of ECG, G has at most $k - k' + 1$ connected components, and hence $D' \cup A'$ satisfy the condition (d) of being in the family $\mathcal{Q}(\emptyset, \emptyset, T_r, 0, 0, 0, \emptyset, \emptyset)$. Assume that the statement is true for $i - 1$. That is, there exists $D' \cup A' \subseteq D \cup A$ and a valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, X, Y)$ such that $D' \cup A' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, X, Y)$ and there is a proper deficiency map ψ' over $R = (S(G - D', f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{D',f})$ with the property that $E_{\psi'} = A'$. We need to show that the statement hold for i .

Case 1 : $X, Y = \emptyset$.

Since $i - 1 < k$, we have that $D' \neq D$ or $A' \neq A$.

Subcase (i): $D' \neq D$. Let $e = (x, y) \in D \setminus D'$. Let $D'' = D' \cup \{e\}$ and $Y' = \emptyset$. Now we explain how to construct X' . The set X' is a subset of $\{x, y\}$. If $x \in \text{Green}$ and $x \notin T_m \setminus T'_m$, then we add x to X' . If $x \in \text{Red}$, then we add x to X' . Similar case holds for y as well. We set $X' = \{z \in \{x, y\} \mid z \notin T_m \setminus T'_m\}$. Note that if $z \in \{x, y\}$ is a Red vertex, then $z \notin T_m \setminus T'_m$. The following claim follows from the definition of X' .

Claim 15.2. *Let $z \in \{x, y\} \cap \text{Green}$. Then $z \notin X'$ if and only if $z \in T_m \setminus T'_m$.*

Let $T''_m = T'_m \cup (\{x, y\} \cap (T_m \setminus T'_m))$. That is, we add those elements from $\{x, y\}$ to T'_m that appear in $T_m \setminus T'_m$. Now we define $k''_1 = k'_1 + 1$ and $k''_2 = k'_2$ if $e \in E(G[\text{Green}])$, otherwise $k''_1 = k'_1$ and $k''_2 = k'_2 + 1$. Now we claim that $D'' \cup A' \in \mathcal{Q}(T''_m, T'_g, T'_r, k''_1, k''_2, i, X', Y')$ and $(T''_m, T'_g, T'_r, k''_1, k''_2, i, X', Y')$ is a valid tuple. Since $|X'| \leq 2$, $Y' = \emptyset$, and $(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, X, Y)$ is a valid tuple, $(T''_m, T'_g, T'_r, k''_1, k''_2, i, X', Y')$ satisfies properties (1), (2) and (3) of a valid tuple. Because of Claim 15.2, the tuple $(T''_m, T'_g, T'_r, k''_1, k''_2, i, X', Y')$ satisfies property (4) of a valid tuple. Since $D' \cup A' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, \emptyset, \emptyset)$ and $D'' = D' \cup \{(x, y)\}$, the subset $D'' \cup A'$, satisfies the following conditions.

- (a) $|D'' \cap E(G[\text{Green}])| = k''_1$, $|D'' \setminus E(G[\text{Green}])| = k''_2$ and $|D'' \cup A'| = i$.
- (b) Due to Claim 15.2 and the definition of T''_m , we have that for any $v \in \text{Green}$,
 $|D'' \cap E_G(v)| = T''_m(v) + X'(v) + T'_g(v)$.
- (c) Since for any $v \in X'$, $d_{G-D''}(v) < f(v)$, we have that $|X'| = |X'_{D'',f}|$ and $X'_{D'',f} \subseteq S(G - D'', f) \setminus T_r$.
- (d) Since $D \cup A$ is a solution of ECG, by Lemma 15.1, we have that $G - D$ has at most $k - k' + 1$ connected components. This implies that $G - D''$ has at most $k - k' + 1$ connected components.
- (e) Since $S(G - D'', f) \cup T'_g = S(G - D', f) \cup T'_g \cup X'_{D'',f}$ and $(S(G - D', f) \cup T'_g) \cap X'_{D'',f} = \emptyset$, we have that $(S(G - D', f) \cup T'_g) \setminus T_r = (S(G - D'', f) \cup T'_g) \setminus (T_r \cup$

$X'_{D'',f}$). This implies that ψ' is proper deficiency map over $(S(G - D'', f) \cup T'_g) \setminus (T'_r \cup X'_{D'',f})$

Thus we conclude that $D'' \cup A' \in \mathcal{Q}(T''_m, T'_g, T'_r, k''_1, k''_2, i, X', Y')$.

Subcase (ii): $D' = D$. In this subcase we have that $A' \neq A$. Let $(x, y) \in A \setminus A'$. Let $A'' = A' \cup \{(x, y)\}$. Note that $E_\psi = A$ and $E_{\psi'} = A'$. Since $D' = D$, for all $v \in \text{Green}$ the number of edges in D' which are incident to v is equal to $\Phi(v)$. Also, note that $T_m(v) + T_g(v) = \Phi(v)$. This implies that $T'_m = T_m$ and $T'_g = T_g$. Since ψ is a proper deficiency map over $S(G - D, f) = S(G - D, f) \cup T_g$, ψ' is a proper deficiency map over $(S(G - D', f) \cup T'_g) \setminus T'_r$ and $(x, y) \in E_\psi \setminus E_{\psi'}$, there exists j, j' such that $x(j), y(j') \in T'_r$. Now we claim that $D' \cup A'' \in \mathcal{Q}(T'_m, T'_g, T'_r \setminus \{x(j), y(j')\}, k'_1, k'_2, i, \emptyset, \emptyset)$. Since $D' \cup A' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, \emptyset, \emptyset)$, $D' \cup A''$ satisfies conditions (a), (b), (c) and (d) of being in the family $\mathcal{Q}(T'_m, T'_g, T'_r \setminus \{x(j), y(j')\}, k'_1, k'_2, i, \emptyset, \emptyset)$. Now we need to show that $D' \cup A''$ satisfies condition (e). Consider the bijection ψ'' defined over $(S(G - D', f) \cup T'_g) \setminus (T'_r \setminus \{x(j), y(j')\})$ as follows.

$$\psi''(q) = \begin{cases} y(j') & \text{if } q = x(j) \\ x(j) & \text{if } q = y(j') \\ \psi'(q) & \text{otherwise} \end{cases}$$

Note that $E_{\psi''} = A'' \subseteq A$. Since ψ' is a proper deficiency map, $\psi''(x(j)) = y(j')$, $\psi''(y(j')) = x(j)$, $E_{\psi''}$ is not a multiset and $E_{\psi''} \cap E(G) = \emptyset$, we have that ψ'' is a proper deficiency map. It is easy to see that $(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, \emptyset, \emptyset)$ is a valid tuple.

Case 2 : $X \neq \emptyset, Y = \emptyset$.

Let $v \in X$ and let j be the smallest integer such that $v(j) \in X_{D',f}$. Since $X_{D',f} \subseteq S(G - D', f) \setminus T_r$, we have that $v(j) \in S(G - D', f) \subseteq S(G - D, f)$. Also since ψ' is a proper deficiency map over $(S(G - D', f) \cup T'_g) \setminus (T'_r \cup X_{D',f})$ and $E_{\psi'} \subseteq E_\psi$, there

exists $b \in V(G)$ such that $(v, b) \notin E_{\psi'}$ and $(v, b) \in E_{\psi}$. Let $A'' = A' \cup \{(v, b)\}$.

Subcase (i): $b \in X$. In this subcase the set $X' = \emptyset$. Let $j' = f(b) - d_{G-D'}(b)$. Note that $\{v(j), b(j')\} = X_{D',f} \subseteq S(G-D', f) \setminus T_r$. Let $T_g'' = T_g' \cup (\{v(j), b(j')\} \cap T_g)$. If $v \in \text{Green}$, then we know that $|E_G(v) \cap D'| = T_m'(v) + T_g'(v) + X(v)$ and $X(v) = 1$. Since the new set $X' = \emptyset$, to keep track of the cardinality $|E_G(v) \cap D'|$ we include $v(j)$ to T_g'' . The Claim 15.3 ensures that $v(j)$ does not belongs to T_g' . The similar arguments holds for b as well.

Claim 15.3. *If $v \in \text{Green}$, then $v(j) \notin T_g'$ and $v(j-1) \in T_g'$*

Proof. Since $\{v, b\} = X$, $j = f(v) - d_{G-D'}(v)$. This implies that,

$$\begin{aligned} |E_G(v) \cap D'| &= j + d_G(v) - f(v) \\ &= j + T_m(v) \end{aligned} \tag{15.1}$$

We also know that, by the property (b) of partial solutions,

$$\begin{aligned} |E_G(v) \cap D'| &= T_m'(v) + T_g'(v) + X(v) \\ &= T_m(v) + T_g'(v) + X(v) \quad (\because v \in X) \end{aligned} \tag{15.2}$$

Equations (15.1) and (15.2) implies that $j = T_g'(v) + X(v)$. This implies $T_g'(v) = j - 1$ because $|X(v)| = 1$. Thus we can conclude that $v(j) \notin T_g'$ and $v(j-1) \in T_g'$ \square

Similarly way we can prove the following claim.

Claim 15.4. *If $b \in \text{Green}$, then $b(j') \notin T_g'$ and $b(j'-1) \in T_g'$*

We claim that $D' \cup A'' \in \mathcal{Q}(T_m', T_g'', T_r', k_1', k_2', i, \emptyset, \emptyset)$. We know that $D' \cup A' \in \mathcal{Q}(T_m', T_g', T_r', k_1', k_2', i-1, X, \emptyset)$. Thus, by Claims 15.3 and 15.4, we can conclude that $D' \cup A''$ satisfies conditions (a), (b), (c) and (d) of being in $\mathcal{Q}(T_m', T_g'', T_r', k_1', k_2', i, \emptyset, \emptyset)$.

Now we need to show that $D' \cup A''$ satisfies condition (e). Consider the bijection ψ'' over $(S(G - D'), f) \cup T_g'' \setminus T_r'$ as follows.

$$\psi''(q) = \begin{cases} b(j') & \text{if } q = v(j) \\ v(j) & \text{if } q = b(j') \\ \psi'(q) & \text{otherwise} \end{cases}$$

Note that that $E_{\psi''} = A''$. Since ψ' is a proper deficiency map, $\psi''(u(j)) = v(j')$, $\psi''(v(j')) = u(j)$, $E_{\psi''}$ is not a multiset and $E_{\psi''} \cap E(G) = \emptyset$, we have that ψ'' is a proper deficiency map. It is easy to see that $(T_m', T_g'', T_r', k_1', k_2', i, \emptyset, \emptyset)$ satisfy properties (1), (2) and (4) of a valid tuple. Claim 15.3 and Claim 15.4 implies that $(T_m', T_g'', T_r', k_1', k_2', i, \emptyset, \emptyset)$ satisfy property (3) of a valid tuple.

Subcase (ii): $b \notin X$ and $b(j') \in T_r'$ for some j' . Let $T_g'' = T_g' \cup (\{v(j)\} \cap T_g)$.

Claim 15.5. *If $v \in \text{Green}$, then $v(j) \notin T_g'$ and $v(j-1) \in T_g'$.*

Proof. If $X(v) = 1$, then the proof is same as the proof of Claim 15.3. Suppose $X(v) = 2$. Since $\{v, v\} = X$, $j = f(v) - d_{G-D'}(v) - 1$. This implies that,

$$\begin{aligned} |E_G(v) \cap D'| &= j + 1 + d_G(v) - f(v) \\ &= j + 1 + T_m(v) \end{aligned} \tag{15.3}$$

We also know that, by the property (b) of partial solutions,

$$\begin{aligned} |E_G(v) \cap D'| &= T_m'(v) + T_g'(v) + X(v) \\ &= T_m(v) + T_g'(v) + 2 \quad (\because v \in X) \end{aligned} \tag{15.4}$$

Equations (15.3) and (15.4) implies that $j - 1 = T_g'(v)$. Thus we can conclude that $v(j) \notin T_g'$ and $v(j-1) \in T_g'$ □

Now we claim that $D' \cup A'' \in \mathcal{Q}(T_m', T_g'', T_r' \setminus \{b(j')\}, k_1', k_2', i, X', \emptyset)$ where $X' =$

$X \setminus \{v\}$. Since $D' \cup A' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, X, \emptyset)$ and by Claim 15.5, we can conclude that $D' \cup A''$ satisfies conditions (a), (b), (c) and (d) of being in the family $\mathcal{Q}(T'_m, T'_g, T'_r \setminus \{b(j')\}, k'_1, k'_2, i, X', \emptyset)$. Now we need to show that $D' \cup A''$ satisfies condition (e). Consider the bijection ψ'' over $(S(G - D', f) \cup T''_g) \setminus ((T'_r \setminus \{b(j')\}) \cup X'_{D', f})$ as follows.

$$\psi''(q) = \begin{cases} b(j') & \text{if } q = v(j) \\ v(j) & \text{if } q = b(j') \\ \psi'(q) & \text{otherwise} \end{cases}$$

Note that $E_{\psi''} = A'' \subseteq A$. Since ψ' is a proper deficiency map, $\psi''(v(j)) = b(j')$, $\psi''(b(j')) = v(j)$, $E_{\psi''}$ is not a multiset and $E_{\psi''} \cap E(G) = \emptyset$, we have that ψ'' is a proper deficiency map. Since $X' \subseteq X$ and $(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, X, \emptyset)$ is a valid tuple, we have that $(T'_m, T'_g, T'_r \setminus \{b(j')\}, k'_1, k'_2, i, X', \emptyset)$ satisfies properties (1), (2) and (4) of a valid tuple. Claim 15.5 implies that $(T'_m, T'_g, T'_r \setminus \{b(j')\}, k'_1, k'_2, i, X', \emptyset)$ satisfies property (3).

Subcase (iii): $b \notin X$ and $b(j') \notin T'_r$ for all j' . Let $T''_g = T'_g \cup (\{v(j)\} \cap T_g)$.

Claim 15.6. *If $v \in \text{Green}$, then $v(j) \notin T'_g$ and $v(j - 1) \in T'_g$*

Proof of Claim 15.6 is same as the proof of Claim 15.5. Now we claim that $D' \cup A'' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X', Y')$, where $X' = X \setminus \{v\}$ and $Y' = \{b\}$. Since $D' \cup A' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, X, \emptyset)$, $X' \subseteq X$, and by Claim 15.6, we can conclude that $D' \cup A''$ satisfies conditions (a), (b), (c) and (d) of being in the family $\mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X', Y')$. Now we need to show that $D' \cup A''$ satisfies condition (e). Consider the bijection ψ'' over $(S(G - D', f) \cup T''_g \cup \{b\}) \setminus (T'_r \cup X'_{D', f})$ defined as follows.

$$\psi''(q) = \begin{cases} b & \text{if } q = v(j) \\ v(j) & \text{if } q = b \\ \psi'(q) & \text{otherwise} \end{cases}$$

Note that $E_{\psi''} = E_{\psi'} \cup \{e\} = A''$. Since ψ' is a proper deficiency map, $\psi''(v(j)) =$

b , $\psi''(b) = v(j)$, $E_{\psi''}$ is not a multiset and $E_{\psi''} \cap E(G) = \emptyset$, we have that ψ'' is a proper deficiency map. Also note that $\psi''(b) = v(j)$ and $T_m(v) = T'_m(v)$, because $v \in X$ and $(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, X, \emptyset)$ is a valid tuple. Since $X' \subset X$, $Y' = \{b\}$, $b(j') \notin T'_r$ for all j' and $(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, X, \emptyset)$ is a valid tuple, we have that $(T'_m, T''_g, T'_r, k'_1, k'_2, i, X', Y')$ satisfies properties (1), (2) and (4) of a valid tuple. Claim 15.5 implies that $(T'_m, T''_g, T'_r \setminus \{b(j')\}, k'_1, k'_2, i, X', \emptyset)$ satisfies property (3).

Case 3 : $Y \neq \emptyset$.

Let $Y = \{w\}$. Since $(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, X, Y)$ is a valid tuple and $|Y| = 1$, we have that $|X| \leq 1$. We claim that there exists $x \in V(G)$ such that $(w, x) \in D \setminus D'$. Suppose not, then $d_{G-D'}(w) = d_{G-D}(w)$. This implies that $d_{G-D'+A'}(w) = d_{G-D+A'}(w)$. We know that ψ' is a proper deficiency map over $(S(G - D', f) \cup T'_g \cup \{w\}) \setminus (T'_r \cup X_{D',f})$, $E_{\psi'} = A'$ and $w(i') \notin T'_r \cup X_{D',f}$ for all i' . This implies that $d_{G-D'+A'}(w) \geq f(w) + 1$ and hence $d_{G-D+A}(w) > f(w)$, contradicting the fact that $D \cup A$ is a solution to ECG. Thus we know that there exists x such that $(w, x) \in D \setminus D'$. Let $D'' = D' \cup \{(w, x)\}$ and $X' = X \cup \{z \mid z = x, x \notin T_m \setminus T'_m\}$. Note that $|X'| \leq 2$, because $|X| \leq 1$. Let $T''_m = T'_m \cup (\{x\} \cap (T_m \setminus T'_m))$. The following claim follows from the definition of X' .

Claim 15.7. *Let $x \in \text{Green}$. Then $x \notin X' \setminus X$ if and only if $x \in T_m \setminus T'_m$.*

Subcase (i): $w \in \text{Green}$. Let $k''_1 = k'_1 + 1, k''_2 = k'_2$ if $(w, x) \in E(G[\text{Green}])$, otherwise $k''_1 = k'_1, k''_2 = k'_2 + 1$. We claim that there exists $j \in \mathbb{N}^+$ such that $w(j) \in T_g \setminus T'_g$. Suppose not. Since ψ' is a proper deficiency map over $(S(G - D') \cup T'_g \cup \{w\}) \setminus (T'_r \cup X_{D',f})$ such that $E'_{\psi} = A' \subseteq A$ and $w(j) \notin T_g \setminus T'_g$ for all j , we can conclude that the number of edges in A' which are incident to w is at least $1 + |\{w(j') \mid j' \in \mathbb{N}^+, w(j') \in T_g\}| = 1 + \Phi(w) - (d_G(w) - f(w))$. This implies that $d_{G-D+A}(w) \geq d_{G-D+A'}(w) \geq d_G(w) - |E_G(v) \cap D| + 1 + \Phi(w) - (d_G(w) - f(w)) = d_G(w) - \Phi(w) + 1 + \Phi(w) - (d_G(w) - f(w)) > f(w)$, which

is a contradiction to the fact that $D \cup A$ is a solution of ECG. Without loss of generality let j be the smallest integer such that $w(j) \in T_g \setminus T'_g$. Now we show that $D'' \cup A' \in \mathcal{Q}(T''_m, T''_g, T'_r, k''_1, k''_2, i, X', \emptyset)$, where $T''_g = T'_g \cup \{w(j)\}$. Since $D' \cup A' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i-1, X, Y)$ and $D'' = D' \cup \{(w, x)\}$, the subset $D'' \cup A'$, satisfies the following conditions.

(a) $|D'' \cap E(G[\text{Green}])| = k''_1$, $|D'' \setminus E(G[\text{Green}])| = k''_2$ and $|D'' \cup A'| = i$.

(b) For any $v \in \text{Green}$,

$$\begin{aligned} |D'' \cap E_G(v)| &= \begin{cases} T'_m(v) + T'_g(v) + X(v) + 1 & \text{if } v \in \{w, x\} \\ T'_m(v) + T'_g(v) + X(v) & \text{if } v \notin \{w, x\} \end{cases} \\ &= \begin{cases} T'_m(v) + T''_g(v) + X(v) & \text{if } v = w \text{ (By definition of } T''_g) \\ T''_m(v) + T'_g(v) + X'(v) & \text{if } v = x \text{ (Due to Claim 15.7)} \\ T'_m(v) + T'_g(v) + X(v) & \text{if } v \notin \{w, x\} \end{cases} \\ &= T''_m(v) + T''_g(v) + X'(v) \end{aligned}$$

(c) $|X'| = 1 + |\{z | z = x, x \notin T_m \setminus T'_m\}| = |X'_{D'',f}|$ and $X'_{D'',f} \subseteq S(G - D'', f) \setminus T_r$.

(d) Since $D \cup A$ is a solution of ECG, by Lemma 15.1, we have that $G - D$ has at most $k - k' + 1$ connected components. This implies that $G - D''$ has at most $k - k' + 1$ connected components.

Now we need to show that $D'' \cup A'$, satisfies the condition (e). Consider the bijection ψ'' over $(S(G - D'') \cup T''_g) \setminus (T'_r \cup X'_{D'',f})$ as follows.

$$\psi''(q) = \begin{cases} \psi'(w) & \text{if } q = w(j) \\ w(j) & \text{if } q = \psi'(w) \\ \psi'(q) & \text{otherwise} \end{cases}$$

The function ψ'' is a proper deficiency map such that $E_{\psi''} = A'$. Since the tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i-1, X, Y)$ is a valid tuple, $|X'| \leq 2$ and j is the smallest integer

such that $w(j) \in T_g \setminus T'_g$, we have that $(T''_m, T''_g, T'_r, k''_1, k''_2, i, X', \emptyset)$ satisfies properties (1), (2) and (3) of a valid tuple. Due to Claim 15.7, $(T''_m, T''_g, T'_r, k'_1, k'_2, i, X', \emptyset)$ satisfies property (4) of a valid tuple.

Subcase (ii): $w \in \text{Red}$. In this subcase we claim that $D'' \cup A' \in \mathcal{Q}(T''_m, T'_g, T'_r, k'_1, k'_2 + 1, i, X', \emptyset)$. Let $j = f(w) - d_{G-D''}(w)$. Since $D' \cup A' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i-1, X, Y)$ and $D'' = D' \cup \{(w, x)\}$, the subset $D'' \cup A'$, satisfies the following conditions.

(a) $|D'' \cap E(G[\text{Green}])| = k'_1$, $|D'' \setminus E(G[\text{Green}])| = k'_2 + 1$ and $|D'' \cup A'| = i$.

(b) For any $v \in \text{Green}$,

$$\begin{aligned} |D'' \cap E_G(v)| &= \begin{cases} T'_m(v) + T'_g(v) + X(v) + 1 & \text{if } v = x \\ T'_m(v) + T'_g(v) + X(v) & \text{if } v \neq x \end{cases} \\ &= T''_m(v) + T'_g(v) + X'(v) \quad (\text{Due to claim 15.7}) \end{aligned}$$

(c) $|X'| = 1 + |\{z | z = x, x \notin T_m \setminus T'_m\}| = |X'_{D'',f}|$ and $X'_{D'',f} \subseteq S(G - D'', f) \setminus T_r$.

(d) $G - D''$ has at most $k - k' + 1$ connected components.

Now we need to show that $D'' \cup A'$, satisfies the condition (e). Consider the bijection ψ'' over $(S(G - D'') \cup T'_g) \setminus (T'_r \cup X'_{D'',f})$ as follows.

$$\psi''(q) = \begin{cases} \psi'(w) & \text{if } q = w(j) \\ w(j) & \text{if } q = \psi'(w) \\ \psi'(q) & \text{otherwise} \end{cases}$$

The function ψ'' is a proper deficiency map such that $E_{\psi''} = A'$. Since the tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i-1, X, Y)$ is a valid tuple, $|X'| \leq 2$, we have that $(T''_m, T'_g, T'_r, k'_1, k'_2 + 1, i, X', \emptyset)$ satisfies properties (1), (2) and (3) of a valid tuple. Due to claim 15.7, $(T''_m, T'_g, T'_r, k'_1, k'_2 + 1, i, X', \emptyset)$ satisfies property (4) of a valid tuple. \square

Our algorithm is based on DP. It keeps a table entry $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ for each valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$. The idea is to store a subset of the family $\mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ in the DP table entry $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ which is sufficient to maintain the correctness of the algorithm. Next we write the recurrence relation for $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ and prove its correctness. Towards that consider the following \blacklozenge and \diamond operations defined as follows. For any family \mathcal{S} of subsets of $\binom{V(G)}{2}$, $e \in E(G)$, and $e' \in \overline{E(G)}$,

$$\begin{aligned} \mathcal{S} \blacklozenge e &= \{B \cup A \cup \{e\} \mid A \cup B \in \mathcal{S}, B \subseteq E(G) \setminus \{e\}, A \subseteq \overline{E(G)}, \\ &\quad G - (B \cup \{e\}) \text{ has at most } k - k' + 1 \text{ connected components}\} \\ \mathcal{S} \diamond e' &= \{B \cup A \cup \{e'\} \mid A \cup B \in \mathcal{S}, B \subseteq E(G), A \subseteq \overline{E(G)} \setminus \{e'\}\} \end{aligned}$$

Now we write the recurrence relation. For $i = 0$, we have the following base cases.

$$\mathcal{D}[T'_m, T'_g, T'_r, 0, 0, 0, X, Y] := \begin{cases} \{\emptyset\} & \text{if } T'_m, T'_g, X, Y = \emptyset, \text{ and } T'_r = T_r \\ \emptyset & \text{otherwise} \end{cases} \quad (15.5)$$

For any invalid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$, we set $D[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y] = \emptyset$. Now we describe how to compute DP table entry for $D[T'_m, T'_g, T'_r, k'_1, k'_2, i + 1, X, Y]$ for a valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i + 1, X, Y)$ using the previously calculated table entries. We write the following recurrence by following Lemma 15.2. That is, we see which all cases in Lemma 15.2 will lead to the current table entry and for the current table entry we just take the union of previously calculated table entries corresponding to these cases.

Case 1: $X = \emptyset$ and $Y = \emptyset$.

$$\begin{aligned}
\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i + 1, X, Y] := & \\
& \left(\bigcup_{\substack{(x,y) \in E(G) \\ x,y \in T'_m}} \mathcal{D}[T'_m \setminus \{x, y\}, T'_g, T'_r, k'_1 - 1, k'_2, i, \emptyset, \emptyset] \blacklozenge \{(x, y)\} \right) \\
& \bigcup \left(\bigcup_{\substack{(x,y) \in \overline{E(G)} \\ \exists j, j' x(j), y(j') \in T_r \setminus T'_r}} \mathcal{D}[T'_m, T'_g, T'_r \cup \{x(j), y(j')\}, k'_1, k'_2, i, \emptyset, \emptyset] \diamond \{(x, y)\} \right) \\
& \bigcup \left(\bigcup_{\substack{(x,y) \in \overline{E(G)} \\ j=T'_g(x), j'=T'_g(y)}} \mathcal{D}[T'_m, T'_g \setminus \{x(j), y(j')\}, T'_r, k'_1, k'_2, i, \{x, y\}, \emptyset] \diamond \{(x, y)\} \right) \\
& \bigcup \left(\bigcup_{\substack{(y,x) \in E(G) \\ j=T'_g(y), x \in T'_m}} \mathcal{D}[T'_m \setminus \{x\}, T'_g \setminus \{y(j)\}, T'_r, k'_1 - 1, k'_2, i, \emptyset, \{y\}] \blacklozenge \{(y, x)\} \right) \\
& \bigcup \left(\bigcup_{\substack{(y,x) \in E(G) \\ x \in T'_m, y \in \text{Red}}} \mathcal{D}[T'_m \setminus \{x\}, T'_g, T'_r, k'_1, k'_2 - 1, i, \emptyset, \{y\}] \blacklozenge \{(y, x)\} \right) \\
& \bigcup \left(\bigcup_{\substack{(x,y) \in \overline{E(G)} \\ \exists j, y(j) \in T_r \setminus T'_r}} \mathcal{D}[T'_m, T'_g, T'_r \cup \{y(j)\}, k'_1, k'_2, i, \{x\}, \emptyset] \diamond \{(x, y)\} \right) \quad (15.6)
\end{aligned}$$

Case 2: $x \in X \neq \emptyset$ and $Y = \emptyset$.

$$\begin{aligned}
\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i+1, X, Y] := & \\
& \left(\bigcup_{\substack{(x,y) \in E(G[\text{Green}]) \\ y \in X, T'_m(x) = T'_m(x) \\ T'_m(y) = T'_m(y)}} \mathcal{D}[T'_m, T'_g, T'_r, k'_1 - 1, k'_2, i, \emptyset, \emptyset] \blacklozenge \{(x, y)\} \right) \\
\cup & \left(\bigcup_{\substack{(x,y) \in E(G) \\ y \in X, x \in \text{Green}, y \in \text{Red} \\ T'_m(x) = T'_m(x)}} \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2 - 1, i, \emptyset, \emptyset] \blacklozenge \{(x, y)\} \right) \\
\cup & \left(\bigcup_{\substack{(x,y) \in E(G[\text{Green}]) \\ y \notin X, y \in T'_m \\ T'_m(x) = T'_m(x)}} \mathcal{D}[T'_m \setminus \{y\}, T'_g, T'_r, k'_1 - 1, k'_2, i, X \setminus \{x\}, \emptyset] \blacklozenge \{(x, y)\} \right) \\
\cup & \left(\bigcup_{\substack{(x,y) \in E(G) \\ x \in \text{Red}, y \notin X, y \in T'_m}} \mathcal{D}[T'_m \setminus \{y\}, T'_g, T'_r, k'_1, k'_2 - 1, i, X \setminus \{x\}, \emptyset] \blacklozenge \{(x, y)\} \right) \\
\cup & \left(\bigcup_{\substack{(y,z) \in \overline{E(G)}, j = T'_g(z) \\ \exists j': y(j') \in T_r \setminus T'_r}} \mathcal{D}[T'_m, T'_g \setminus \{z(j)\}, T'_r \cup \{y(j')\}, k'_1, k'_2, i, X \setminus \{x\}, \emptyset] \diamond \{(y, z)\} \right) \\
\cup & \left(\bigcup_{\substack{(y,z) \in E(G) \\ j = T'_g(y), z \in T'_m}} \mathcal{D}[T'_m \setminus \{z\}, T'_g \setminus \{y(j)\}, T'_r, k'_1 - 1, k'_2, i, X, \{y\}] \blacklozenge \{(y, x)\} \right) \\
\cup & \left(\bigcup_{\substack{(y,x) \in E(G[\text{Green}]) \\ x \notin T'_m, j = T'_g(y)}} \mathcal{D}[T'_m, T'_g \setminus \{y(j)\}, T'_r, k'_1 - 1, k'_2, i, X \setminus \{x\}, \{y\}] \blacklozenge \{(y, x)\} \right) \\
\cup & \left(\bigcup_{\substack{(y,x) \in E(G) \\ x \in \text{Red}, j = T'_g(y)}} \mathcal{D}[T'_m, T'_g \setminus \{y(j)\}, T'_r, k'_1 - 1, k'_2, i, X \setminus \{x\}, \{y\}] \blacklozenge \{(y, x)\} \right) \\
\cup & \left(\bigcup_{\substack{(y,z) \in E(G), z \notin X \\ y \in \text{Red}, z \in T'_m}} \mathcal{D}[T'_m \setminus \{z\}, T'_g, T'_r, k'_1, k'_2 - 1, i, X, \{y\}] \blacklozenge \{(y, x)\} \right) \\
\cup & \left(\bigcup_{\substack{(y,x) \in E(G) \\ y \in \text{Red}, x \notin T'_m}} \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2 - 1, i, X \setminus \{x\}, \{y\}] \blacklozenge \{(y, x)\} \right) \tag{15.7}
\end{aligned}$$

Case 3: $y \in Y \neq \emptyset$.

$$\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i + 1, X, Y] := \left(\bigcup_{\substack{(x,y) \in \overline{E(G)}, j=T'_g(x) \\ X'=X \cup \{x\}, T'_r(y)=0}} \mathcal{D}[T'_m, T'_g \setminus \{x(j)\}, T'_r, k'_1, k'_2, i, X', \emptyset] \diamond \{(x, y)\} \right) \quad (15.8)$$

The algorithm computes the family $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ for all valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ and if there exists $D \cup A \in \mathcal{D}[T_m, T_g, \emptyset, k_1, k_2, k, \emptyset, \emptyset]$ such that $D \cup A$ is a solution to ECG, then outputs YES, otherwise outputs NO. Since the size of $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ can potentially be $n^{\mathcal{O}(i)}$, this algorithm takes time $n^{\mathcal{O}(k)}$. Now we prove the correctness of the algorithm.

Correctness. If the algorithm outputs YES, then there exists $D \cup A$ which is a solution to ECG. Now we need to show that if the input instance is an YES instance, then the algorithm will always output YES. Lemma 15.4 achieves this. The following lemma is useful for Lemma 15.4 and it can be proved by induction on i .

Lemma 15.3. $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y] \subseteq \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$, for any valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$,

The next lemma is very similar to Lemma 15.2 and we use some of the arguments used there (for example in showing a particular tuple to be valid) directly in our proof.

Lemma 15.4. Let (G, f, k) be an YES instance of ECG with a solution $D \cup A$ such that $D \subseteq E(G), A \subseteq \overline{E(G)}, |D \cap E(G[\text{Green}])| = k_1, |D \setminus E(G[\text{Green}])| = k_2, k_1 + k_2 = k'$ and $|D \cap E_G(v)| = \Phi(v)$ for all $v \in \text{Green}$. Let ψ be a proper deficiency map over $S(G - D, f)$ such that $E_\psi = A$. Then for each $i \leq k$, there exists $D' \cup A' \subseteq D \cup A$ and a valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ such that $D' \cup A' \in \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ and there is a proper deficiency map ψ' over $R = (S(G - D', f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{D',f})$ with the property that $E_{\psi'} = A'$. Moreover $D \cup A \in \mathcal{D}[T_m, T_g, \emptyset, k_1, k_2, k, \emptyset, \emptyset]$.

Proof. We prove the lemma by induction on i and its proof is very much similar to the proof of Lemma 15.2. For $i = 0$ we set $D', A' = \emptyset$ and by definition, $\emptyset \in$

$\mathcal{D}[\emptyset, \emptyset, T_r, 0, 0, 0, \emptyset, \emptyset]$. We assume that the statement is true for $i - 1$. That is, there exists $D' \cup A' \subseteq D \cup A$ and a valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, X, Y)$ such that $D' \cup A' \in \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, X, Y]$ and there is a proper deficiency map ψ' over $R = (S(G - D', f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{D', f})$ with the property that $E_{\psi'} = A'$. Due to Lemma 15.3, $D' \cup A' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, X, Y)$. We need to show that the statement holds for i .

Case 1 : $X, Y = \emptyset$.

Since $i - 1 < k$, we have that $D' \neq D$ or $A' \neq A$.

Subcase (i): $D' \neq D$. Let $e = (x, y) \in D \setminus D'$. Let $D'' = D' \cup \{e\}$, $X' = \{z \in \{x, y\} \mid z \notin T_m \setminus T'_m\}$ and $Y' = \emptyset$. Let $T''_m = T'_m \cup (\{x, y\} \cap (T_m \setminus T'_m))$. We have several cases based on vertices belong to X' .

Suppose $X' = \{x, y\}$ and $x, y \in \text{Green}$. By the definition of X' , since $x, y \in X'$, we have that $x, y \notin T_m \setminus T'_m$. This implies that $T''_m = T'_m$, $T_m(x) = T'_m(x)$ and $T_m(y) = T'_m(y)$. Thus by Equation 15.7, $D'' \cup A' \in \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, \emptyset, \emptyset] \blacklozenge \{(x, y)\} \subseteq \mathcal{D}[T'_m, T'_g, T'_r, k'_1 + 1, k'_2, i, X', \emptyset]$.

Suppose $X' = \{x, y\}$, $x \in \text{Green}$ and $y \in \text{Red}$. By the definition of X' , since $x \in X'$, we have that $x \notin T_m \setminus T'_m$. This implies that $T''_m = T'_m$ and $T_m(x) = T'_m(x)$. Thus by Equation 15.7, $D'' \cup A' \in \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i - 1, \emptyset, \emptyset] \blacklozenge \{(x, y)\} \subseteq \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2 + 1, i, X', \emptyset]$.

Suppose $X' = \{x\}$ and $x \in \text{Green}$. Then by the definition of X' , since $x \in X'$ and $y \notin X'$, we have that $x \notin T_m \setminus T'_m$ and $y \in T_m \setminus T'_m$. This implies that $T''_m = T'_m \cup \{y\}$ and $T_m(x) = T'_m(x)$. Thus by Equation 15.7, $D'' \cup A' \in \mathcal{D}[T''_m \setminus \{y\}, T'_g, T'_r, k'_1, k'_2, i - 1, \{x\} \setminus \{x\}, \emptyset] \blacklozenge \{(x, y)\} \subseteq \mathcal{D}[T''_m, T'_g, T'_r, k'_1 + 1, k'_2, i, X', \emptyset]$.

Suppose $X' = \{x\}$ and $x \in \text{Red}$. By the definition of X' , since $y \notin X'$, we have that $y \in T_m \setminus T'_m$. This implies that $T''_m = T'_m \cup \{y\}$. Thus by Equation 15.7, $D'' \cup A' \in$

$$\mathcal{D}[T_m'' \setminus \{y\}, T_g', T_r', k_1', k_2', i-1, \{x\} \setminus \{x\}, \emptyset] \diamond \{(x, y)\} \subseteq \mathcal{D}[T_m'', T_g', T_r', k_1', k_2'+1, i, X', \emptyset].$$

Suppose $X' = \emptyset$. Then by the definition of X' , since $x, y \notin X'$, we have that $x, y \in T_m \setminus T_m'$ and $T_m'' = T_m' \cup \{x, y\}$. Thus by Equation 15.6, $D'' \cup A' \in \mathcal{D}[T_m'' \setminus \{x, y\}, T_g', T_r', k_1', k_2', i-1, \emptyset, \emptyset] \diamond \{(x, y)\} \subseteq \mathcal{D}[T_m'', T_g', T_r', k_1'+1, k_2', i, \emptyset, \emptyset]$.

In all the above cases, we can show that ψ' is a proper deficiency map over $(S(G - D'', f) \cup T_g') \setminus (T_r' \cup X_{D'', f}')$ and its proof is same as the corresponding proof in the Lemma 15.2. By argument similar to the one in the proof of Lemma 15.2, we can show that $(T_m'', T_g', T_r', k_1'+1, k_2', i, \emptyset, \emptyset)$ is a valid tuple.

Subcase (ii): $D' = D$. In this subcase we have that $A' \neq A$. Let $(x, y) \in A \setminus A'$. Let $A'' = A' \cup \{(x, y)\}$. Note that $E_\psi = A$ and $E_{\psi'} = A'$. Since $D' = D$, for all $v \in \mathbf{Green}$ the number of edges in D' which are incident to v is equal to $\Phi(v)$. Also, note that $T_m(v) + T_g(v) = \Phi(v)$. This implies that $T_m' = T_m$ and $T_g' = T_g$. Since ψ is a proper deficiency map over $S(G - D, f) = S(G - D, f) \cup T_g$, ψ' is a proper deficiency map over $(S(G - D', f) \cup T_g') \setminus T_r'$ and $(x, y) \in E_\psi \setminus E_{\psi'}$, there exists j, j' such that $x(j), y(j') \in T_r'$. Thus by Equation 15.6, $D' \cup A'' \in \mathcal{D}[T_m', T_g', T_r', k_1', k_2', i-1, \emptyset, \emptyset] \diamond \{(x, y)\} \subseteq \mathcal{D}(T_m', T_g', T_r' \setminus \{x(j), y(j')\}, k_1', k_2', i, \emptyset, \emptyset)$.

We can show that there is a proper deficiency map ψ'' over $(S(G - D', f) \cup T_g') \setminus (T_r' \setminus \{x(j), y(j')\})$ such that $A'' = E_{\psi''}$ and its proof is same as the corresponding proof in the Lemma 15.2. Since $D' \cup A' \in \mathcal{Q}(T_m', T_g', T_r', k_1', k_2', i-1, X, Y)$, by argument similar to the one in the proof of Lemma 15.2, we can show that $(T_m', T_g', T_r' \setminus \{x(j), y(j')\}, k_1', k_2', i, \emptyset, \emptyset)$ is a valid tuple.

Case 2 : $X \neq \emptyset, Y = \emptyset$.

Let $x \in X$ and let j be the smallest integer such that $x(j) \in X_{D', f}$. Since $X_{D', f} \subseteq S(G - D', f) \setminus T_r$, we have that $x(j) \in S(G - D', f) \subseteq S(G - D, f)$. Also since ψ' is a proper deficiency map over $(S(G - D', f) \cup T_g') \setminus (T_r' \cup X_{D', f})$ and $E_{\psi'} \subseteq E_\psi$, there

exists $y \in V(G)$ such that $(x, y) \notin E_{\psi'}$ and $(x, y) \in E_{\psi}$. Let $A'' = A' \cup \{(x, y)\}$.

Subcase (i): $y \in X$. Let $j' = f(y) - d_{G-D'}(y)$. Note that $\{x(j), y(j')\} = X_{D',f} \subseteq S(G - D', f) \setminus T_r$. Let $T_g'' = T_g' \cup (\{x(j), y(j')\} \cap T_g)$.

Claim 15.8. *If $x \in \text{Green}$, then $x(j) \notin T_g'$ and $x(j-1) \in T_g'$. If $y \in \text{Green}$, then $y(j') \notin T_g'$ and $y(j-1) \in T_g'$*

The Claim 15.8 is identical to the Claim 15.3. Thus by Equation 15.6, $D' \cup A'' \in \mathcal{D}[T_m', T_g', T_r', k_1', k_2', i-1, X, \emptyset] \diamond \{(x, y)\} \subseteq \mathcal{D}[T_m', T_g'', T_r', k_1', k_2', i, \emptyset, \emptyset]$. We can show that there is a proper deficiency map ψ'' over $(S(G - D', f) \cup T_g'') \setminus T_r'$ such that $A'' = E_{\psi''}$ and its proof is same as the corresponding proof in the Lemma 15.2. Since $D' \cup A' \in \mathcal{Q}(T_m', T_g', T_r', k_1', k_2', i-1, X, Y)$, by argument similar to the one in the proof of Lemma 15.2, we can show that $(T_m', T_g'', T_r', k_1', k_2', i, \emptyset, \emptyset)$ is a valid tuple.

Subcase (ii): $y(j') \in T_r'$ for some j' . Let $T_g'' = T_g' \cup (\{x(j)\} \cap T_g)$.

Claim 15.9. *If $x \in \text{Green}$, then $x(j) \notin T_g'$ and $x(j-1) \in T_g'$*

Claim 15.9 is identical to Claim 15.5. Thus, by Equation 15.6 (if $X = \{x\}$) and by Equation 15.7 (if $X \neq \{x\}$),

$$D' \cup A'' \in \mathcal{D}[T_m', T_g', T_r', k_1', k_2', i-1, X, \emptyset] \diamond (x, y) \subseteq \mathcal{D}[T_m', T_g'', T_r' \setminus y(j'), k_1', k_2', i, X \setminus \{x\}, \emptyset].$$

We can show that there is a proper deficiency map ψ'' over $(S(G - D', f) \cup T_g'') \setminus ((T_r' \setminus \{y(j)\}) \cup X_{D',f})$, where $X' = X \setminus \{x\}$, such that $A'' = E_{\psi''}$ and its proof is same as the corresponding proof in the Lemma 15.2. Since $D' \cup A' \in \mathcal{Q}(T_m', T_g', T_r', k_1', k_2', i-1, X, Y)$, by argument similar to the one in the proof of Lemma 15.2, we can show that $(T_m', T_g'', T_r' \setminus y(j'), k_1', k_2', i, X \setminus \{x\}, \emptyset)$ is a valid tuple.

Subcase (iii): $y \notin X$ and $y(j') \notin T_r'$ for all j' . Let $T_g'' = T_g' \cup (\{x(j)\} \cap T_g)$.

Claim 15.10. *If $x \in \text{Green}$, then $x(j) \notin T'_g$ and $x(j-1) \in T'_g$*

Proof of the above claim is same as the proof of Claim 15.5 as both are identical.

Thus, by Equation 15.8,

$$D' \cup A'' \in \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i-1, X, \emptyset] \diamond (x, y) \subseteq \mathcal{D}[T'_m, T''_g, T'_r, k'_1, k'_2, i, X \setminus \{x\}, \{y\}].$$

We can show that there is a proper deficiency map ψ'' over $(S(G - D', f) \cup T''_g \cup Y') \setminus (T'_r \cup X'_{D', f})$, where $X' = X \setminus \{x\}$ and $Y' = \{y\}$, such that $A'' = E_{\psi''}$ and its proof is same as the corresponding proof in the Lemma 15.2. Since $D' \cup A' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i-1, X, Y)$, by argument similar to the one in the proof of Lemma 15.2, we can show that $(T'_m, T''_g, T'_r, k'_1, k'_2, i, X \setminus \{x\}, \{y\})$ is a valid tuple.

Case 3 : $Y \neq \emptyset$.

Let $Y = \{y\}$. Since $(T'_m, T'_g, T'_r, k'_1, k'_2, i-1, X, Y)$ is a valid tuple and $|Y| = 1$, we have that $|X| \leq 1$. There exists x such that $(y, x) \in D \setminus D'$ and the proof of this statement can be found in the proof of Lemma 15.2. Let $D'' = D' \cup \{(y, x)\}$ and $X' = X \cup \{z \mid z = x, z \notin T_m \setminus T'_m\}$. Note that $|X'| \leq 2$, because $|X| \leq 1$. Let $T''_m = T'_m \cup (\{x\} \cap (T_m \setminus T'_m))$.

Subcase (i): $y \in \text{Green}$. Then there exists $j \in \mathbb{N}^+$ such that $y(j) \in T_g \setminus T'_g$ and its proof can be found in Lemma 15.2. Without loss of generality let j be the smallest integer such that $y(j) \in T_g \setminus T'_g$. Let $T''_g = T'_g \cup \{y(j)\}$.

Subsubcase (ia): $(y, x) \in E(G[\text{Green}])$. If $X = X' = \emptyset$, then by the definition of X' , $T''_m = T'_m \cup \{x\}$. Then by Equation 15.6,

$$D'' \cup A' \in \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i-1, \emptyset, Y] \blacklozenge (y, x) \subseteq \mathcal{D}[T''_m, T''_g, T'_r, k'_1 + 1, k'_2, i, X', \emptyset].$$

If $X = X' \neq \emptyset$, then by the definition of X' , $T_m'' = T_m' \cup \{x\}$. Then by Equation 15.7, $D'' \cup A' \in \mathcal{D}[T_m', T_g', T_r', k_1', k_2', i-1, X, Y] \blacklozenge(y, x) \subseteq \mathcal{D}[T_m'', T_g'', T_r'', k_1' + 1, k_2', i, X', \emptyset]$.

If $X \neq X'$, then by the definition of X' , $T_m'' = T_m'$. Then by Equation 15.7, $D'' \cup A' \in \mathcal{D}[T_m', T_g', T_r', k_1', k_2', i-1, X, Y] \blacklozenge(y, x) \subseteq \mathcal{D}[T_m'', T_g'', T_r'', k_1' + 1, k_2', i, X', \emptyset]$.

We can show that there is a proper deficiency map ψ'' over $(S(G - D', f) \cup T_g'') \setminus (T_r' \cup X'_{D', f})$, such that $A'' = E_{\psi''}$ and its proof is same as the corresponding proof in the Lemma 15.2. Since $D' \cup A' \in \mathcal{Q}(T_m', T_g', T_r', k_1', k_2', i-1, X, Y)$, by argument similar to the one in the proof of Lemma 15.2, we can show that $(T_m'', T_g'', T_r'', k_1' + 1, k_2', i, X', \emptyset)$ is a valid tuple.

Subsubcase (ib): $(y, x) \notin E(G[\text{Green}])$. In this subcase $X' = X \cup \{x\}$. Then by Equation 15.7, $D'' \cup A' \in \mathcal{D}[T_m', T_g', T_r', k_1', k_2', i-1, X, Y] \blacklozenge(y, x) \subseteq \mathcal{D}[T_m'', T_g'', T_r'', k_1' + 1, k_2' + 1, i, X', \emptyset]$.

We can show that there is a proper deficiency map ψ'' over $(S(G - D', f) \cup T_g'') \setminus (T_r' \cup X'_{D', f})$, such that $A'' = E_{\psi''}$ and its proof is same as the corresponding proof in the Lemma 15.2. Since $D' \cup A' \in \mathcal{Q}(T_m', T_g', T_r', k_1', k_2', i-1, X, Y)$, by argument similar to the one in the proof of Lemma 15.2, we can show that $(T_m'', T_g'', T_r'', k_1', k_2' + 1, i, X', \emptyset)$ is a valid tuple.

Subcase (ii): $y \in \text{Red}$. If $X = X' = \emptyset$, then by the definition of X' , $T_m'' = T_m' \cup \{x\}$. Then by Equation 15.6, $D'' \cup A' \in \mathcal{D}[T_m', T_g', T_r', k_1', k_2', i-1, \emptyset, Y] \blacklozenge(y, x) \subseteq \mathcal{D}[T_m'', T_g'', T_r'', k_1', k_2' + 1, i, \emptyset, \emptyset]$. If $X = X' \neq \emptyset$, then by the definition of X' , $T_m'' = T_m' \cup \{x\}$. Then by Equation 15.7, $D'' \cup A' \in \mathcal{D}[T_m', T_g', T_r', k_1', k_2', i-1, X, Y] \blacklozenge(y, x) \subseteq \mathcal{D}[T_m'', T_g'', T_r'', k_1', k_2' + 1, i, X', \emptyset]$.

If $X \neq X'$, then by the definition of X' , $T_m'' = T_m'$. Then by Equation 15.7, $D'' \cup A' \in \mathcal{D}[T_m', T_g', T_r', k_1', k_2', i-1, X, Y] \blacklozenge(y, x) \subseteq \mathcal{D}[T_m'', T_g'', T_r'', k_1', k_2' + 1, i, X', \emptyset]$.

We can show that there is a proper deficiency map ψ'' over $(S(G - D', f) \cup T_g'') \setminus (T_r' \cup X'_{D', f})$,

$X'_{D',f}$), such that $A'' = E_{\psi''}$ and its proof is same as the corresponding proof in the Lemma 15.2. Since $D' \cup A' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i-1, X, Y)$, by argument similar to the one in the proof of Lemma 15.2, we can show that $(T''_m, T'_g, T'_r, k'_1, k'_2 + 1, i, X', \emptyset)$ is a valid tuple.

Now we need to show that $D \cup A \in \mathcal{D}[T_m, T_g, \emptyset, k_1, k_2, k, \emptyset, \emptyset]$. We proved that there is a valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, k, X, Y)$ such that $D \cup A \in \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, k, X, Y]$. Due to Lemma 15.3, $D \cup A \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, k, X, Y)$. This implies that there is a proper deficiency map ψ' over $(S(G - D, f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{D,f})$ such that $E_{\psi'} = A$. Since $D \cup A$ is a solution to ECG, there is a proper deficiency map ψ over $S(G - D, f)$ such that $E_\psi = A$. This implies that

$$S(G - D, f) = (S(G - D, f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{D,f}) \quad (15.9)$$

Equation 15.9 implies that $Y = \emptyset$. Since $T'_r \subseteq S(G - D, f)$ and $X_{D,f} \subseteq S(G - D, f) \setminus T'_r$, by Equation 15.9, we get that $T'_r = \emptyset$ and $X = \emptyset$. Since $D \cup A \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, k, X, Y)$, we have that for any $v \in \text{Green}$, $|E_G(v) \cap D| = T'_m(v) + T'_g(v)$. By assumption we know that $|E_G(v) \cap D| = \Phi(v) = T_m(v) + T_g(v)$. This implies that $T'_m = T_m$ and $T'_g = T_g$. We also know, by assumption that, $D \cap E(G[\text{Green}]) = k_1$ and $D \setminus E(G[\text{Green}]) = k_2$. This implies that $k'_1 = k_1$ and $k'_2 = k_2$. Hence $D \cup A \in \mathcal{D}[T_m, T_g, \emptyset, k_1, k_2, k, \emptyset, \emptyset]$. This completes the proof. \square

15.2.3 Pruning the DP table – FPT algorithm

Now we explain how to prune the family of partial solutions stored at each DP table entry such that its size is at most $2^{\mathcal{O}(k)}$ and there by get an FPT algorithm. The objective is to find a nice deletion set $D \subseteq E(G)$. In fact if the input instance is an YES instance we will find a set $D \subseteq E(G)$, $A \subseteq \overline{E(G)}$ such that D is a nice deletion set with the property that $A = E_\psi$, where ψ is a proper deficiency map over $S(G - D, f)$.

Recall that for the algorithm we have guessed k' – the size of proposed deletion set D , k_1 – the number edges in $D \cap E(G[\text{Green}])$, k_2 – the number of edges in $D \setminus E(G[\text{Green}])$ and for all $v \in \text{Green}$, $\Phi(v) (\geq d_G(v) - f(v))$ – the number of

edges in D which are incident to v . Consider the property (i) of Lemma 15.1, i.e $d_{G-D}(v) \leq f(v)$ for all v . By guessing $\Phi(v) \geq d_G(v) - f(v)$ for all $v \in \text{Green}$, we know that any solution we compute will satisfy property (i).

Consider property (ii) mentioned in Lemma 15.1, i.e $|S(G-D, f)| = 2(k-k')$. Since the total number of edges in D which has one end point in Green and other in Red is $(\sum_{v \in \text{Green}} \Phi(v)) - 2k_1$, we have that $\sum_{v \in \text{Red}} |D \cap E_G(v)| = 2k_2 - ((\sum_{v \in \text{Green}} \Phi(v)) - 2k_1)$.

$$\begin{aligned}
|S(G-D, f)| &= \left(\sum_{v \in \text{Green}} \Phi(v) - (d_G(v) - f(v)) \right) + \sum_{v \in \text{Red}} (f(v) - d_{G-D}(v)) \\
&= \left(\sum_{v \in \text{Green}} \Phi(v) - (d_G(v) - f(v)) \right) + \\
&\quad \sum_{v \in \text{Red}} (f(v) - d_G(v)) + \sum_{v \in \text{Red}} |D \cap E_G(v)| \\
&= \left(\sum_{v \in \text{Green}} \Phi(v) - (d_G(v) - f(v)) \right) + \sum_{v \in \text{Red}} (f(v) - d_G(v)) \\
&\quad + \left(2k_2 + 2k_1 - \sum_{v \in \text{Green}} \Phi(v) \right) \\
&= 2k_1 + 2k_2 + \sum_{v \in V} (f(v) - d_G(v))
\end{aligned}$$

So after guessing k', k_1, k_2 and $\Phi(v)$ for all $v \in \text{Green}$, we check whether $2k_1 + 2k_2 + \sum_{v \in V} (f(v) - d_G(v)) = 2(k-k')$ and if they are not equal we consider it as a invalid guess. Thus our guesses takes care of property (ii).

The property (iii) of nice deletion set and Lemma 15.5 below imply that D is an independent set in the matroid $M_G^*(\ell)$. That is, ℓ -elongation of the co-graphic matroid, M_G^* , associated with G , where $\ell = |E(G)| - |V(G)| + k - |D| + 1$.

Lemma 15.5. *Let G be a graph and $D \subseteq E(G)$. Then D is an independent set in the ℓ -elongation of M_G^* where $\ell = |E(G)| - |V(G)| + k - |D| + 1$ if and only if $G-D$ has at most $k - |D| + 1$ connected components.*

Proof. Let r be the number of connected components in G . Suppose D is an independent set in $M_G^*(\ell)$. Then there exists $S \subseteq E(G) \setminus D$ such that $S \cup D$ is a basis of $M_G^*(\ell)$. This implies that there exists $S' \subseteq S \cup D$ such that S' is a basis of M_G^* , and hence $G - S'$ is a forest with r connected components and $|S'| = |E(G)| - |V(G)| + r$. Since $|S \cup D| - |S'| = (k - |D| + 1) - r$ and $G - S'$ is a forest with exactly r connected

components, we have that $G - (S \cup D)$ has exactly $k - |D| + 1$ connected components. This implies that $G - D$ has at most $k - |D| + 1$ connected components.

Suppose $G - D$ has at most $k - |D| + 1$ connected components. Let $S \subseteq E(G)$ be a maximal subset such that $G - (S \cup D)$ is a forest with exactly $k - |D| + 1$ connected components. This implies that $|S \cup D| = E(G) - V(G) + k - |D| + 1$. Since G has r connected components, there exists $S' \subseteq (S \cup D)$ of size $(k - |D| + 1) - r$ such that $G - ((S \cup D) \setminus S')$ is a forest with exactly r connected components. Since $|(S \cup D) \setminus S'| = E(G) - V(G) + r$ and $G - ((S \cup D) \setminus S')$ is a forest with exactly r connected components, $(S \cup D) \setminus S'$ is a basis in M_G^* . This implies that $S \cup D$ is a basis in $M_G^*(\ell)$ and hence D is an independent set in $M_G^*(\ell)$. \square

Thus by only considering those D which are independent sets in $M_G^*(\ell)$ we ensure that property (iii) of the nice deletion set is satisfied.

Now consider the property (v) of the nice deletion set, i.e, there exists a proper deficiency map $\psi : S(G - D, f) \rightarrow S(G - D, f)$. Our objective is get a set $D \cup A$ such that there is a proper deficiency map ψ over $S(G - D, f)$ such that $E_{\psi} = A$, along with other properties as well. We have already defined equivalence classes for the partial solutions in the previous section, which is the framework in which we will design our FPT algorithm as well. Let $D_1 \cup A_1, D_2 \cup A_2 \in Q(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ be two partial solutions where $D_1, D_2 \subseteq E(G)$ and $A_1, A_2 \subseteq \overline{E(G)}$. Suppose $D' \subseteq E(G)$, $A' \subseteq \overline{E(G)}$, $(D_1 \cup D') \cup (A_1 \cup A')$ is a solution and $A_2 \cap A' = \emptyset$. Since $D_1 \cup A_1, D_2 \cup A_2$ belongs to same equivalence class and $A_2 \cap A'$ is disjoint, there is a proper deficiency map ψ' over $S(G - (D_2 \cup D'), f)$ such that $E_{\psi'} = A_2 \cup A'$. To take care of the disjointness property between the current addition set and the future addition set while doing the DP, we view the the addition set A of the solution as an independent set in a uniform matroid over the universe $\overline{E(G)}$. Let $U_{m', k-k'}$ be a uniform matroid with ground set $\overline{E(G)}$, where $m' = |\overline{E(G)}|$. From the definition of $U_{m', k-k'}$, any set A of size at most $k - k'$ is independent in $U_{m', k-k'}$. We have already explained that we view the deletion set D as an independent set in $M_G^*(\ell)$ where $\ell = |E(G)| - |V(G)| + k - k' + 1$. Thus, to see the solution set $D \cup A$ as an independent set in a single matroid, we consider direct sum of $M_G^*(\ell)$ and $U_{m', k-k'}$. That is, let $M = M_G^*(\ell) \oplus U_{m', k-k'}$. In M , a set I is an independent set if and only if $I \cap E(G)$ is an independent set in $M_G^*(\ell)$ and $I \cap \overline{E(G)}$ is an independent set in $U_{m', k-k'}$. This ensures that any solution $D \cup A$ is an independent set in M . By viewing any solution of the problem as an independent set in a matroid M (which is linear), we can use the representative families to prune the table. However, we still need to ensure that property (iv) of nice deletion set is satisfied. In what follows we explain how we achieve this.

Consider the property (iv) mentioned in the definition of the nice deletion set. That

is, for any connected component F in $G - D$, $V(F) \cap \text{def}(G - D, f) \neq \emptyset$. The following lemma helps us to satisfy property (iv) partially.

Lemma 15.6. *Let F be a connected component in the graph $G[\text{Red}]$ and $D' \subseteq E(G)$. If at least one edge in D' is incident to a vertex in $V(F)$, then for any connected component C in $G - D'$ such that $V(C) \cap V(F) \neq \emptyset$ there is a vertex $v \in V(C) \cap \text{def}(G - D', f)$.*

Proof. Let $u \in V(F)$ be a vertex such that an edge in D' is incident to u . Consider a connected component C in $G - D'$ such that $V(C) \cap V(F) \neq \emptyset$. We need to show that $V(C) \cap \text{def}(G - D', f) \neq \emptyset$. Suppose $u \in V(C)$. Since $u \in \text{Red}$, $d_G(u) \leq f(v)$. However, u is incident to an edge in D' , and thus we have that $d_{G-D'}(u) < f(u)$. This implies that $u \in V(C) \cap \text{def}(G - D', f)$. Now we are in a case where $u \notin V(C)$. Pick an arbitrary vertex $w \in V(C) \cap V(F)$. Since $w, u \in V(F)$, there exists a path P from w to u using only vertices from Red . Since w and u are in different connected components in $G - D'$, $D' \cap E(P) \neq \emptyset$. Pick the first edge (v, v') in the path P which are also in D' . Note that there exists a path from w to v in $G - D'$ and $v \in V(C)$. Since $v \in \text{Red}$ and $(v, v') \in D'$, we have that $v \in V(C) \cap \text{def}(G - D', f)$. This completes the proof. \square

Now we explain how Lemma 15.6 is useful in satisfying property (iv) partially. Let \mathcal{C} be the set of connected components in G such that for each vertex v in the component, $d_G(v) = f(v)$.

$$\mathcal{C} = \{C \mid C \text{ is a connected component in } G \wedge \forall v \in V(C), d_G(v) = f(v)\}.$$

Let D_1 and D_2 be deletion sets corresponding to two partial solutions such that for all $C \in \mathcal{C}$, $D_1 \cap E(C) \neq \emptyset$ if and only if $D_2 \cap E(C) \neq \emptyset$. Suppose there is a $D' \subseteq E(G)$ such that $D_1 \cup D'$ is a nice deletion set. Now we claim that any connected component F in $G - (D_2 \cup D')$ containing only red vertices will have a deficient vertex. Let $v \in V(F)$ and $v \notin V(\mathcal{C})$. We also know that $v \in \text{Red}$. Since $v \notin V(\mathcal{C}) (= \bigcup_{C \in \mathcal{C}} V(C))$ one of the following conditions hold.

1. There is a path from v to a vertex in Green in the graph G .

Since $v \in V(F)$ and F is a fully red connected component in $G - (D_2 \cup D')$, there is a vertex w in $V(F)$ such that $D_2 \cup D'$ contains an edge incident on w . Since $w \in \text{Red}$ as well, $w \in \text{def}(G - (D_2 \cup D'), f)$.

2. Else, v is in a connected component C_1 of G such that $V(C_1) \subseteq \text{Red}$ and there is a vertex $u \in V(C_1)$ such that $f(u) > d_G(v)$.

If $F = C_1$, then u is the required deficient vertex. If $F \neq C_1$, then by Lemma 15.6, $V(F) \cap \text{def}(G - (D_2 \cup D'), f) \neq \emptyset$.

Let $v \in V(F)$ and $v \in V(C)$ where $C \in \mathcal{C}$. Since $D_1 \cup D'$ is a solution either $D_1 \cap E(C) \neq \emptyset$ or $D' \cap E(C) \neq \emptyset$. If $D_1 \cap E(C) \neq \emptyset$, then by our assumption, $D_2 \cap E(C) \neq \emptyset$. Thus by Lemma 15.6, $V(F) \cap \text{def}(G - (D_2 \cup D'), f) \neq \emptyset$.

Essentially due to Lemma 15.6, if we partition our partial solutions based on how these partial solutions hit the edges from \mathcal{C} and keep at least one from each equivalence class property (iv) of nice deletion set will be satisfied partially. But this only allows us to take care of connected components containing only red vertices. Now we explain how we can ensure property (iv) for the connected components containing vertices from Green as well.

To achieve this we will prove that for corresponding to every deletion set D of a solution, there is a “witness” of $\mathcal{O}(k)$ sized subset of edges whose disjointness from D will ensure property (iv) of nice deletion sets. That is, these witnesses are depended on solutions; the witness for solution D will be different from the witness for solution D^* . Even then, these witnesses allow us to satisfy property (iv). In order to avoid this witness being picked in a deletion set D , that is to keep this witness non deletable, we use **color coding** in our algorithm on top of representative family based pruning of table entries. Towards that we define a weight function w on $E(G)$ as follows.

$$w((u, v)) = \begin{cases} 0 & \text{if } u, v \in \text{Red} \\ 1 & \text{otherwise} \end{cases}$$

For any subset $S \subseteq E(G)$, $w(S) = \sum_{e \in S} w(e)$. The next lemma is crucial for our approach as this not only defines the witness but also gives an upper bound on its size.

Lemma 15.7. *Let $\text{Green} = \{v_1, v_2, \dots, v_\eta\}$, $\eta \leq 2k$. Let $D \subseteq E(G)$ such that for any connected component F in $G - D$, $V(F) \cap \text{def}(G - D, f) \neq \emptyset$. Then there exist paths P_1, \dots, P_η such that for all i , P_i is a path in $G - D$ from v_i to a vertex in $\text{def}(G - D, f)$, and $w(\bigcup_i E(P_i)) \leq 6k$ where $\bigcup_i E(P_i)$ is the set of edges in the paths P_1, \dots, P_η .*

Proof. We construct P_1, \dots, P_η with the required property. Pick an arbitrary vertex $u_1 \in \text{def}(G - D, f)$ such that v_1 and u_1 are in the same connected component in $G - D$. Let P_1 be a smallest weight path according to weight function w , from v_1 to u_1 in $G - D$. Now we explain how to construct P_i , given that we have already

constructed paths P_1, \dots, P_{i-1} . Pick an arbitrary vertex $u_i \in \text{def}(G-D, f)$ such that v_i and u_i are in the same connected component in $G-D$. Let P be a smallest weight path from v_i to u_i in $G-D$. If P is vertex disjoint from P_1, \dots, P_{i-1} , then we set $P_i = P$. Otherwise, let x be the first vertex in P such that $x \in V(P_1) \cup \dots \cup V(P_{i-1})$. Let $x \in P_j$ where $j < i$. Let $P = P'P''$ such that P' ends in x and P'' starts at x . Let $P_j = P'_jP''_j$ such that P'_j ends in x and P''_j starts at x . Now we set $P_i = P'P''_j$. Note that P_i is a path in $G-D$ from v_i to a vertex in $\text{def}(G-D, f)$.

Now we claim that $w(\bigcup_{i=1}^n E(P_i)) \leq 6k$. Towards the proof we need to count that $|(\bigcup_{i=1}^n E(P_i)) \cap w^{-1}(1)| \leq 6k$. We assign each vertex v in $\bigcup_{i=1}^n V(P_i)$ to the smallest indexed path P_j such that $v \in V(P_j)$. That is, v is assigned to P_j , if $v \in V(P_j)$ and $v \notin (\bigcup_{i=1}^{j-1} V(P_i))$. Note that each vertex in $\bigcup_{i=1}^n V(P_i)$ is assigned to a unique path. Consider the edge set $A^* \subseteq (\bigcup_{i=1}^n E(P_i)) \cap w^{-1}(1)$ as follows. An edge $e = (u, v)$ belongs to A^* if $w(e) = 1, e \in E(P_j)$, and vertices u and v are assigned to path P_j for some j . Observe that each edge $e \in A^*$ has at least one end point in **Green**. Since each vertex is assigned to exactly one path, each vertex in a path has degree at most 2 and $|\text{Green}| \leq 2k$, we have that $|A^*| \leq 4k$.

Now we show that there exists sets $\emptyset = B_1 \subseteq B_2 \subseteq \dots \subseteq B_n$ such that $(\bigcup_{i=1}^j E(P_i)) \cap w^{-1}(1) \subseteq A^* \cup B_j$ and $|B_j| \leq j$. We prove the statement using induction on j . For $j = 1$, we know that $(\bigcup_{i=1}^1 E(P_i)) \cap w^{-1}(1) \subseteq A^*$. Thus the statement is true. Now suppose the statement is true for $j-1$. Consider any path P_j . If the vertices in P_j are disjoint from $\bigcup_{i=1}^{j-1} V(P_i)$, then all the weight one edges in $E(P_j)$ are counted in A^* . So we can set $B_j = B_{j-1}$ and the statement is true. Otherwise by the construction of P_j , we have that $P_j = P'_jP''_j$ and there exists $r < j$ such that $P_r = P'_rP''_r$. Let (u_1, u_2) be the last edge in P'_j . Note that all the weight one edges in $E(P''_j)$ are counted in $A^* \cup B_{j-1}$ and all the weight one edges in $E(P'_j) \setminus \{(u_1, u_2)\}$ are counted in A^* . In this case we set $B_j = B_{j-1}$ if $w((u_1, u_2)) = 0$ and $B_j = B_{j-1} \cup \{(u_1, u_2)\}$ otherwise. This implies that $|(\bigcup_{i=1}^n E(P_i)) \cap w^{-1}(1)| \leq 6k$. This concludes the proof. \square

Recall that $E_r = E(G[\text{Red}])$ and $E_g = E(G) \setminus E_r$. Note that in Lemma 15.7, the

weight of each edge in E_g is 1 and the weight of each edge in E_r is 0. By Lemma 15.7, we have that if D is a nice deletion set, then there exists $E' \subseteq E_g$ of cardinality at most $6k$ such that E' witnesses that each connected component of $G - D$ containing at least one vertex from **Green**, will also contain a vertex from $\text{def}(G - D, f)$. We call such an edge set E' as **certificate** of D . Now we explain how Lemma 15.7 helps us to satisfy property (iv) of nice deletion sets for components containing at least one vertex from **Green**. Let $\text{Green} = \{v_1, \dots, v_\eta\}$ and $D_1 \cup D'$ be a deletion set corresponding to a solution. By Lemma 15.7 we know that there are paths P_1, \dots, P_η such that the total number edges from E_g among these paths is bounded by $6k$, and each path P_i is from v_i to a vertex in $\text{def}(G - (D_1 \cup D'), f)$. Suppose we color the edges in E_g with **black** and **orange** such that the coloring guarantees that all the edges in $E_g \cap (\bigcup_{i=1}^\eta E(P_i))$ are colored black and all the edges in $E_g \cap (D_1 \cup D')$ are colored orange. Assume that we are going to find a nice deletion set which does not contains black color edges. Let D_2 be a deletion set corresponding to a partial solution. Also for a vertex $v_i \in \text{Green}$, there is path from v_i to a vertex in $\text{def}(G - D_1, f)$ in the graph $G - D_1$ which does not contain any orange colored edge if and only if there is path from v_i to a vertex in $\text{def}(G - D_2, f)$ in the graph $G - D_2$ which does not contain any orange colored edge. Like in the case of red components, we can show that any connected component in $G - (D_2 \cup D')$ containing a vertex from **Green** will contain a vertex from $\text{def}(G - (D_2 \cup D'), f)$. The formal proof of this statement will be given in Lemma 15.8. Essentially by Lemma 15.7 we get the following. Suppose we take all partial solutions corresponding to a DP table entry (or a subset of it) and now we partition these partial solutions based on which all green vertices have found their deficient vertex currently (there are $2^{|\text{Green}|}$ such partitions), then it is enough to keep a partial solution from each class. Furthermore, suppose \mathcal{A} corresponds to partial solutions with respect to one particular subset of **Green** and we have kept a set D_1 in \mathcal{A} and deleted rest of the partial solutions from \mathcal{A} , say one of the partial solution we threw out was D_2). Then, if there is D' such that $D_2 \cup D'$ is a solution, then all the connected components in $G - (D_1 \cup D')$ containing at least one green vertex will have a deficient vertex. Just a word of caution that in our actual algorithm in fact we keep a subset of \mathcal{A} of size $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ so that we can also take care of all other properties of a nice deletion set. Even though we explained that the property (iv) can be achieved by imposing more structure to the equivalence class we defined in the last section, we will not include these structures to the index of the DP table entries. Rather for each table entry, indexed by an equivalence class, we keep at least one partial solutions for each refinement of this equivalence class based on which green vertices have found their partner deficient vertex. This will ensure that property (iv) is satisfied.

We have explained how we will ensure each of the individual properties of a nice deletion set. Now we design a randomized FPT algorithm for the problem. Later we derandomize the algorithm. The algorithm is a DP algorithm in which we have DP table entries indexed exactly in the same way as in the case of the XP algorithm. But instead of keeping $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$, we store a small representative family of $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ which is enough to maintain the correctness of the algorithm. The algorithm uses both color coding and representative family techniques. We have explained that we use color coding to separate the proposed

deletion set from its certificate mentioned in Lemma 15.7. We color each edge $e \in E_g$ black with probability $6/7$ and orange with probability $1/7$. Let E_b be the set of edges colored black and E_o be the set of edges colored orange. Let D be a deletion set of size k' for the problem and paths P_1, \dots, P_η be its witness mentioned in Lemma 15.7. Then the number of edges in paths P_1, \dots, P_η , which are from E_g is bounded by $6k$. We say that a random coloring is *good* if each edge in $D \cap E_g$ is colored orange and each edge in $E_g \cap (\bigcup_{i=1}^\eta P_i)$ is colored black. The random coloring of edges in E_g is *good*, with probability $\left(\frac{6^6}{7^7}\right)^k$. Now our algorithm works with the edge colored graph and output a nice deletion set D , with the property that $D \cap E_g \subseteq E_o$, if there exists such a deletion set. We know that if the input instance is an YES instance then with probability at least $\left(\frac{6^6}{7^7}\right)^k$ our algorithm will output a solution. Thus we can increase the success probability to at least $(1 - 1/e)$ by running the entire algorithm $\left(\frac{7^7}{6^6}\right)^k$ times. So now onwards we assume that the edges in E_g of the input graph is colored with black or orange, and our objective is to find out a nice deletion set D such that all edges in $D \cap E_g$ is colored orange. Note that the edges in E_r is uncolored.

Recall that \mathcal{C} is the set of connected components in G such that for each vertex v in the component, $d_G(v) = f(v)$. Now we define a family \mathcal{J} of functions as,

$$\mathcal{J} = \{g : \text{Green} \cup \mathcal{C} \rightarrow \{0, 1\}\}.$$

Now we explain how to reduce the size of $\mathcal{D}[T'_g, T'_r, k'_1, k'_2, i, X, Y]$ which is computed using the recurrence relations (equations 15.6, 15.7 and 15.8). We say a partial solution $B \in \mathcal{D}[T'_g, T'_r, k'_1, k'_2, i, X, Y]$ is *properly colored*, if $B \cap E_b = \emptyset$. Since our objective is to find out a nice deletion set disjoint from E_b , we delete all partial solutions which contains an edge from E_b . That is if $B \in \mathcal{D}[T'_g, T'_r, k'_1, k'_2, i, X, Y]$ and $B \cap E_b \neq \emptyset$, then we delete B from $\mathcal{D}[T'_g, T'_r, k'_1, k'_2, i, X, Y]$. So now onwards we assume that for each $B \in \mathcal{D}[T'_g, T'_r, k'_1, k'_2, i, X, Y]$, $B \cap E_b = \emptyset$. Further pruning of the DP table entry $\mathcal{D}[T'_g, T'_r, k'_1, k'_2, i, X, Y]$ is discussed below.

Definition 15.4. *The subset $\mathcal{R}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y] \subseteq \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ is called a representative partial solutions for $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$, denoted by*

$$\mathcal{R}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y] \sqsubseteq_{rep}^{k-i} \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y],$$

if the following happens. If there exists $B, Z \in \binom{V(G)}{2}$ such that $B \cap Z = \emptyset$, $B \in \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$, $B \cap E_b = \emptyset$ and $(B \cup Z) \cap E(G)$ satisfies five properties of a nice deletion set with the property that there exists a proper deficiency map ψ

with $E_\psi = (B \cup Z) \cap \overline{E(G)}$, then there exists $\widehat{B} \subseteq \binom{V(G)}{2}$ such that $\widehat{B} \cap Z = \emptyset$, $\widehat{B} \cap E_b = \emptyset$, $\widehat{B} \in \mathcal{R}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ and $(\widehat{B} \cup Z) \cap E(G)$ satisfies five properties mentioned of a nice deletion set with the property that there exists a proper deficiency map ψ' with $E_{\psi'} = (\widehat{B} \cup Z) \cap \overline{E(G)}$

For each valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ we compute a representative partial solutions for $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ in the increasing order of i and store it instead of $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$. Now we explain how to compute it and prove its correctness. First we compute a subfamily $\mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ of $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ using the recurrence relation (equations 15.6, 15.7 and 15.8) on the DP table entries computed for value $i-1$ and then delete all partial solutions which contain edges from E_b . Now we partition $\mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ according to the refinement of each function in \mathcal{J} . That is $\mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y] = \bigcup_{g \in \mathcal{J}} \mathcal{A}_g$ where \mathcal{A}_g is defined as follows. For each $g \in \mathcal{J}$ $S \cup R \in \mathcal{A}_g$, where $S \in E(G)$ and $R \in \overline{E(G)}$, if the following happens.

- (i) For any $v \in \text{Green}$, $g(v) = 1$ if and only if there exists a path from v to a vertex in $\text{def}(G - S, f)$ in $G[E_b \cup (E_r \setminus S)]$ (checking whether there is a witness path that do not use edges in E_o).
- (ii) For any $C \in \mathcal{C}$, $g(C) = 1$ if and only if $S \cap E(C) \neq \emptyset$.

Recall that any set $S \cup R \in \mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ is an independent set of size i in M . Now we compute $\widehat{\mathcal{A}}_g \subseteq_{rep}^{k-i} \mathcal{A}_g$ using Theorem 13.3. Then we set

$$\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y] = \bigcup_{g \in \mathcal{J}} \widehat{\mathcal{A}}_g$$

and store it instead of $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$. The next lemma prove the correctness of this step.

Lemma 15.8. $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y] \subseteq_{rep}^{k-i} \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$.

Proof. We prove the lemma by induction on i . Suppose there exists $B, Z \subseteq \binom{V(G)}{2}$ such that $B \in \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$, $B \cap Z = \emptyset$, $B \cap E_b = \emptyset$, $(B \cup Z) \cap E(G)$ satisfies the five properties of a nice deletion set and there exists a proper deficiency map ψ over $S(G - ((B \cup Z) \cap E(G)), f)$ with the property that $E_\psi = (B \cup Z) \cap \overline{E(G)}$. By the recurrence relations given by Equations 15.6, 15.7 and 15.8, there exists $e \in B$

and a valid tuple $(T''_m, T''_g, T''_r, k''_1, k''_2, i-1, X', Y')$ such that

$$B \setminus \{e\} \in \mathcal{D}[T''_m, T''_g, T''_r, \widehat{k''_1, k''_2}, i-1, X', Y'].$$

Let $B' = B \setminus \{e\}$ and $Z' = Z \cup \{e\}$. We know that $(B' \cup Z') \cap E(G) = (B \cup Z) \cap E(G)$ satisfies five properties of a nice deletion set and ψ is a proper deficiency map over $S(G - ((B' \cup Z') \cap E(G)), f)$ with the property that $E_\psi = (B' \cup Z') \cap \overline{E(G)}$. Thus, by induction hypothesis we have that there exists $\widehat{B}' \subseteq \binom{V(G)}{2}$ such that

- $\widehat{B}' \in \mathcal{D}[T''_m, T''_g, T''_r, \widehat{k''_1, k''_2}, i-1, X', Y']$,
- $\widehat{B}' \cap Z' = \emptyset, \widehat{B}' \cap E_b = \emptyset$
- $(\widehat{B}' \cup Z') \cap E(G)$ satisfies five properties of a nice deletion set and
- there exists a proper deficiency map ψ' over $S(G - ((\widehat{B}' \cup Z') \cap E(G)), f)$ with the property that $E_{\psi'} = (\widehat{B}' \cup Z') \cap \overline{E(G)}$.

Since $(\widehat{B}' \cup Z') \cap E(G)$ is a nice deletion set, $G - ((\widehat{B}' \cup \{e\}) \cap E(G))$ has at most $k - k' + 1$ connected components. The definitions of \blacklozenge, \diamond and recurrence relation of $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$, imply that

- if $e \in E(G)$ then $\widehat{B}' \cup \{e\} \in \mathcal{D}[T''_m, T''_g, T''_r, \widehat{k''_1, k''_2}, i-1, X', Y'] \bullet e$; and
- if $e \in \overline{E(G)}$ then $\widehat{B}' \cup \{e\} \in \mathcal{D}[T''_m, T''_g, T''_r, \widehat{k''_1, k''_2}, i-1, X', Y'] \circ e$.

Also by our assumption, if $e \in E(G)$, then $e \in E(G) \setminus E_b$. For an ease of presentation, let us call $B = \widehat{B}' \cup \{e\}$. Furthermore, let $D = (B \cup Z) \cap E(G)$ and $A = (B \cup Z) \cap \overline{E(G)}$. Now we have that,

- $B \in \mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$
- ψ' is a proper deficiency map over $S(G - ((B \cup Z) \cap E(G)), f)$ such that $E_{\psi'} = (B \cup Z) \cap \overline{E(G)}$.

Let $g : \text{Green} \cup \mathcal{C} \rightarrow \{2, 3\}$ be defined as follows:

1. For any $v \in \mathbf{Green}$, $g(v) = 1$ if and only if there exists a path from v to a vertex in $\mathbf{def}(G - (B \cap E(G)), f)$ in $G[E_b \cup (E_r \setminus B)]$.
2. For any $C \in \mathcal{C}$, $g(C) = 1$ if and only if $B \cap E(C) \neq \emptyset$.

From the definition of \mathcal{A}_g , we have that $B \in \mathcal{A}_g$. Since $B \cup Z$ is an independent set in the matroid M , by the definition of representative families, there exists $\widehat{B} \in \widehat{\mathcal{A}}_g$ such that $\widehat{B} \cap Z = \emptyset$ and $\widehat{B} \cup Z$ is an independent set in M . Note that $\widehat{B} \in \mathcal{D}[T'_m, T'_g, \widehat{T'_r}, k'_1, k'_2, i, X, Y]$ and $\widehat{B} \cap E_b = \emptyset$. To conclude the proof of lemma the only thing that remains to show is that $(\widehat{B} \cup Z) \cap E(G)$ satisfies all the five properties of a nice deletion set. The next claim does this job.

Claim 15.11. *$(\widehat{B} \cup Z) \cap E(G)$ satisfies five properties of a nice deletion set and there exists a proper deficiency map $\widehat{\psi}$ over $S(G - ((\widehat{B} \cup Z) \cap E(G)), f)$ such that $E_{\widehat{\psi}} = (\widehat{B} \cup Z) \cap \overline{E(G)}$.*

Proof. Let $\widehat{D} = (\widehat{B} \cup Z) \cap E(G)$, and $\widehat{A} = (\widehat{B} \cup Z) \cap \overline{E(G)}$. We know that D satisfies five properties of a nice deletion set and there exists a proper deficiency map ψ' over $S(G - D, f)$ with the property that $E_{\psi'} = A$. We need to show that \widehat{D} satisfies five properties of a nice deletion set and there exists a proper deficiency map $\widehat{\psi}$ over $S(G - \widehat{D}, f)$ with the property that $E_{\widehat{\psi}} = \widehat{A}$.

Property (i). We know that $B, \widehat{B} \in \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ because

$$\widehat{\mathcal{A}}_g \subseteq \mathcal{A}_g \subseteq \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y].$$

Hence, for all $v \in \mathbf{Green}$, $d_{G - (\widehat{B} \cap E(G))}(v) = d_{G - (B \cap E(G))}(v)$. This implies that for all $v \in \mathbf{Green}$, $d_{G - \widehat{D}}(v) = d_{G - D}(v)$. Since for all $v \in \mathbf{Green}$, $d_{G - D}(v) \leq f(v)$, we have that for all $v \in \mathbf{Green}$, $d_{G - \widehat{D}}(v) \leq f(v)$. For any $v \in \mathbf{Red}$, $d_{G - \widehat{D}} \leq f(v)$. Hence \widehat{D} satisfies property (i) of a nice deletion set.

Property (ii). Since $B, \widehat{B} \in \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$, we have that $|S(G - D, f)| = |S(G - \widehat{D}, f)|$ and $|D| = |\widehat{D}| = k'$. Thus \widehat{D} satisfies property (ii) of a nice

deletion set.

Property (iii). We know that $\widehat{B} \cup Z$ is an independent set in the matroid M , and hence \widehat{D} is an independent set in $M_G^*(\ell)$, where $\ell = E(G) - V(G) + k - k' + 1$. Thus, by Lemma 15.5, $G - \widehat{D}$ has at most $k - k' + 1 = k - |\widehat{D}| + 1$ connected components, and so \widehat{D} satisfies property (iii) of a nice deletion set.

Property (iv). Now we consider property (iv) of a nice deletion set. We need to show that for any connected component C in $G - \widehat{D}$, $V(C) \cap \text{def}(G - \widehat{D}, f) \neq \emptyset$.

Case 1. Suppose $V(C) \cap \text{Green} \neq \emptyset$. Let $v \in V(C) \cap \text{Green}$. Suppose $g(v) = 1$. Since $\widehat{B} \in \mathcal{A}_g$, there exists a path P in $G[(E_b \cup (E_r \setminus \widehat{B}))]$ from v to a vertex u in $\text{def}(G - (\widehat{B} \cap E(G)), f)$. This implies that $u \in \text{def}(G - \widehat{D}, f)$. If $E(P) \subseteq E(G - \widehat{D})$, then $u \in V(C)$. This implies that $V(C) \cap \text{def}(G - \widehat{D}) \neq \emptyset$. Suppose $E(P) \not\subseteq E(G - \widehat{D})$. This implies that some of the edges in $E(P)$ are present in \widehat{D} . Since for all $e \in \widehat{D}$, $e \in E_o$ and for all $e' \in E_g \cap E(P)$, $e' \in E_b$, any edge $e \in E(P) \cap \widehat{D}$ also belongs to E_r . Let $e_1 = (u_1, v_1)$ be the first edge in the path P , such that $e_1 \in \widehat{D}$. Note that $u_1 \in V(C)$ and $d_{G-\widehat{D}}(u_1) < f(u_1)$, because $u_1 \in \text{Red}$. Hence $V(C) \cap \text{def}(G - \widehat{D}) \neq \emptyset$.

Now we consider the case $g(v) = 0$. We know that there is a path P in $G - D$ from the vertex v to a vertex u such that $u \in \text{def}(G - D, f)$ and $E(P) \cap E_g \subseteq E_b$. This implies that P is a path in $G[E_b \cup (E_r \setminus D)]$. Since $g(v) = 0$ and P is a path in $G[E_b \cup (E_r \setminus D)]$, we have that $d_{G-(B \cap E(G))}(u) \geq f(u)$. If $u \in \text{Green}$, then $d_{G-\widehat{D}}(u) = d_{G-D}(u) < f(u)$. If $u \in \text{Red}$, then there is $e = (u, w) \in Z \cap E(G)$. This implies that $d_{G-\widehat{D}}(u) < f(u)$ because $d_G(u) \leq f(u)$ and $(u, w) \in \widehat{D}$. In either case $u \in \text{def}(G - \widehat{D}, f)$. If $E(P) \subseteq E(G - \widehat{D})$, then $u \in V(C)$. This implies that $V(C) \cap \text{def}(G - \widehat{D}) \neq \emptyset$. Suppose $E(P) \not\subseteq E(G - \widehat{D})$. This implies that some of the edges in $E(P)$ are present in \widehat{D} . Since for all $e \in \widehat{D}$, $e \in E_o$ and for all $e' \in E_g \cap E(P)$, $e' \in E_b$, any edge $e \in E(P) \cap \widehat{D}$ also belongs to E_r . Let $e_1 = (x, y)$ be the first edge in the path P , such that $e_1 \in \widehat{D}$. Note that $x \in V(C)$ and

$d_{G-\widehat{D}}(x) < f(x)$, because $x \in \text{Red}$. Hence $V(C) \cap \text{def}(G - \widehat{D}) \neq \emptyset$.

Case 2. Suppose $V(C) \subseteq \text{Red}$ and there exists a connected component F in \mathcal{C} such that $V(F) \cap V(C) \neq \emptyset$. If $g(F) = 1$, then we know that $\widehat{B} \cap E(G)$ contains an edge which is also present in $E(F)$, because $\widehat{B} \in \mathcal{A}_g$. Then by Lemma 15.6 we have that $V(C) \cap \text{def}(G - \widehat{D}) \neq \emptyset$. If $g(F) = 0$, then there exists an edge e in $Z \cap E(G)$ such that $e \in E(F)$, because D satisfies properties of Lemma 15.1 and $B \in \mathcal{A}_g$. This implies that, by Lemma 15.6, we have that $V(C) \cap \text{def}(G - \widehat{D}) \neq \emptyset$.

Case 3. Suppose $V(C) \subseteq \text{Red}$ and for all connected component F in \mathcal{C} , $V(F) \cap V(C) = \emptyset$. Then there exists a path P from a vertex v in $V(C)$ to a vertex u in **Green** in graph G . Let $e = (x, y)$ be the first edge in the path P such that $e \in \widehat{D}$. Note that such an edge e exists because $V(C) \subseteq \text{Red}$ and $x \in \text{Red}$. Since $x \in \text{Red}$ and $(x, y) \in \widehat{D}$, we have that $d_{G-\widehat{D}}(x) < f(x)$. This implies that $V(C) \cap \text{def}(G - \widehat{D}) \neq \emptyset$.

Property (v). Now consider property (v) of a nice deletion set. We need to show that there exists a proper deficiency map $\widehat{\psi}$ over $S(G - \widehat{D})$. We know that ψ' is a proper deficiency map over $S(G - D)$. Let

$$P = B \cap E(G), \widehat{P} = \widehat{B} \cap E(G), Q = B \cap \overline{E(G)} \text{ and } \widehat{Q} = \widehat{B} \cap \overline{E(G)}.$$

We claim that for all $v \in V(G)$, $d_{G-\widehat{P}+\widehat{Q}}(v) = d_{G-P+Q}(v)$. Since $\widehat{P} \cup \widehat{Q}, P \cup Q \in \mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y] \subseteq \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$, there exist proper deficiency maps ψ_1 over $(S(G - \widehat{P}, f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{\widehat{P}, f})$ and ψ_2 over $(S(G - P, f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{P, f})$ such that $E_{\psi_1} = \widehat{Q}$ and $E_{\psi_2} = Q$. This implies that for any $v \in \text{Red}$,

$$d_{G-\widehat{P}+\widehat{Q}}(v) = f(v) + Y(v) - T'_r(v) - X(v) = d_{G-P+Q}(v).$$

For any $v \in \text{Green}$, since $d_{G-P}(v) = d_{G-\widehat{P}}(v)$ and ψ_1, ψ_2 are proper deficiency maps over $(S(G - \widehat{P}, f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{\widehat{P}, f})$, $(S(G - P, f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{P, f})$ respectively, we have that $d_{G-P+Q}(v) = d_{G-\widehat{P}+\widehat{Q}}(v)$. Hence, for all $v \in V(G)$, $d_{G-\widehat{P}+\widehat{Q}}(v) =$

$d_{G-P+Q}(v)$.

Now we claim that for all $v \in V(G)$, $d_{G-\widehat{D}+\widehat{A}}(v) = d_{G-D+A}(v)$.

$$\begin{aligned} d_{G-\widehat{D}+\widehat{A}}(v) &= d_{G-\widehat{P}+\widehat{Q}}(v) - |E_G(v) \cap Z| + |\overline{E}_G(v) \cap Z| \\ &= d_{G-P+Q}(v) - |E_G(v) \cap Z| + |\overline{E}_G(v) \cap Z| \\ &= d_{G-D+A}(v) \end{aligned}$$

We have that ψ' is a proper deficiency map over $S(G-D), f$ such that $E_{\psi'} = A$. This implies that $d_{G-D+A}(v) = f(v)$ for all $v \in V(G)$. Since $d_{G-\widehat{D}+\widehat{A}}(v) = d_{G-D+A}(v)$, we have that $d_{G-\widehat{D}+\widehat{A}}(v) = f(v)$ for all $v \in V(G)$. Let $\widehat{A} = \{e_1, e_2, \dots, e_r\}$ where $r = k - |D|$. Since for all $v \in V(G)$, $d_{G-\widehat{D}+\widehat{A}}(v) = f(v)$, we have that there are exactly $f(v) - d_{G-D}(v)$ edges in \widehat{A} which are adjacent to v . Now we define a function $\widehat{\psi} : S(G - \widehat{D}, f) \rightarrow S(G - \widehat{D}, f)$ as follows. $\widehat{\psi}(u(i)) = v(j)$ if $(u, v) = e_\ell$ such that there are exactly $i - 1$ edges from $\{e_1, \dots, e_{\ell-1}\}$ are incident on u and there are exactly $j - 1$ edges from $\{e_1, \dots, e_{\ell-1}\}$ are incident on v . By Claim 15.1 of Lemma 15.1 we have that $\widehat{\psi}$ is a proper deficiency map. Since we constructed $\widehat{\psi}$ from \widehat{A} , $E_{\widehat{\psi}} = \widehat{A}$. \square

The proof of above claim completes the proof of the lemma. \square

So our algorithm computes $\mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ using Equations 15.6, 15.7 and 15.8 from DP table entries computed for $i - 1$ and then computes a family $\mathcal{D}[T'_m, T'_g, \widehat{T'_r, k'_1, k'_2}, i, X, Y]$ as explained above. If there exists a set B in the family $\mathcal{D}[T_m, T_g, \widehat{\emptyset, k_1, k_2, k}, \emptyset, \emptyset]$ such that $B \cap E(G)$ is a nice deletion set then the algorithm outputs YES. Otherwise the algorithm outputs NO. The correctness of the algorithm follows from Lemmata 15.1 and 15.8

Running Time. Let $|V(G)| = n$ and $E(G) = m$. Then the rank of the matroid M is bounded by $m + k$. Consider the construction of $\mathcal{D}[T'_m, T'_g, \widehat{T'_r, k'_1, k'_2}, i, X, Y]$. First we constructed $\mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ using equations 15.6, 15.7 or 15.8 from $\mathcal{D}[T''_m, T''_g, \widehat{T''_r, k''_1, k''_2}, i - 1, X', Y']$. Thus the size of $\mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ is,

$$\mathcal{O} \left(\max_{T''_m, T''_g, T''_r, k''_1, k''_2, X', Y'} |\mathcal{D}[T''_m, T''_g, T''_r, \widehat{k''_1, k''_2}, i - 1, X', Y']| \cdot \binom{n}{2} \right)$$

We know that $\mathcal{D}[T'_m, T'_g, T'_r, \widehat{k'_1, k'_2}, i-1, X', Y'] = \bigcup_{g \in \mathcal{J}} \widehat{\mathcal{A}}_g$ where $\widehat{\mathcal{A}}_g$ is a $(k - (i - 1))$ -representative family computed using Theorem 13.3. Thus by Theorem 13.3, $|\widehat{\mathcal{A}}_g|$ is bounded by $(m + k)k \binom{k}{i-1}$. The cardinality of $\text{Green} \cup \mathcal{C}$ is bounded by $2k$, because any solution should contain at least one edge incident to each green vertex and one edge from each component in \mathcal{C} . Hence $|\mathcal{J}| = 4^k$. Thus the size of $\mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ is bounded by $4^k \binom{k}{i-1} (m + k)k \log n = 4^k \binom{k}{i-1} n^{\mathcal{O}(1)}$.

Then we have partitioned $\mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ based on $g \in \mathcal{J}$. That is,

$$\mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y] = \bigcup_{g \in \mathcal{J}} \mathcal{A}_g.$$

Then we computed a $(k - i)$ -representative family $\widehat{\mathcal{A}}_g$ of \mathcal{A}_g for each $g \in \mathcal{J}$. By Theorem 13.3, the running time of this computation is upper bounded by,

$$4^k \binom{k}{i-1} \binom{k}{i}^{\omega-1} n^{\mathcal{O}(1)}$$

Thus the running time to compute $\mathcal{D}[T'_m, T'_g, \widehat{T'_r, k'_1, k'_2}, i, X, Y]$ is bounded by,

$$4^k \binom{k}{i-1} \binom{k}{i}^{\omega-1} n^{\mathcal{O}(1)}.$$

The cardinality of $T'_m \cup T'_g \cup T'_r$ is at most $2k$, otherwise we need more than k edges in the solution. Thus the running time of the algorithm, once we guessed k', k_1, k_2 and $\Phi(v)$ for all v , is upper bounded by,

$$\sum_{i=1}^k 2^{2k} 4^k \binom{k}{i-1} \binom{k}{i}^{\omega-1} n^{\mathcal{O}(1)} = 2^{(4+\omega)k+o(k)} n^{\mathcal{O}(1)}.$$

Since the number of possible guesses for k', k_1, k_2 and Φ is at most $4^k k^{\mathcal{O}(1)}$, the total running time of the algorithm is $2^{(6+\omega)k} n^{\mathcal{O}(1)}$. Also note that we run the entire algorithm $\left(\frac{7^7}{6^6}\right)^k$ time to improve the success probability to at least $(1 - 1/e)$.

Theorem 15.3. *There is a randomized algorithm running in time $\left(\frac{7^7}{6^6}\right)^k 2^{(6+\omega)k} n^{\mathcal{O}(1)}$ for ECG.*

15.2.4 Derandomization

In this subsection we explain how to derandomize the above algorithm. Our algorithm can be derandomized using n - p - q -lopsided-universal family. Recall 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 $B \cap F = \emptyset$. By Lemma 5.5 we know that there is an algorithm that constructs an n - p - q -lopsided-universal family \mathcal{F} of size $\binom{p+q}{p} \cdot 2^{o(p+q)} \cdot \log n$ in time $\mathcal{O}(\binom{p+q}{p} \cdot 2^{o(p+q)} \cdot n \log n)$. To derandomize our algorithm, instead of randomly coloring the edges in E_g , we use $(|E_g|, 7k, k)$ -lopsided-universal family \mathcal{F} . We run our algorithm $|\mathcal{F}|$ many times as follows. For each $F \in \mathcal{F}$, we color F with orange and $E_g \setminus F$ with black and run our algorithm. The correctness of derandomization follows from the definition of n - $7k$ - k -lopsided-universal family.

Fact 15.1. *By Stirling's approximation, $\binom{k}{\alpha k} \leq (\alpha^{-\alpha}(1-\alpha)^{(\alpha-1)})^k$ [112].*

Thus by using Lemma 5.5, we can derandomize our algorithm and we get the following theorem where its running time follows from Fact 21.1.

Theorem 15.4. *There is a deterministic algorithm running in time $\left(\frac{77}{68}\right)^k 2^{(6+\omega)k} n^{\mathcal{O}(1)}$ for ECG.*

15.3 EDITING TO CONNECTED f -DEGREE GRAPH WITH COSTS

In this section we prove hardness of the following weighted variant of the editing problem.

EDITING TO CONNECTED f -DEGREE GRAPH WITH COSTS **Parameter:** $k+d$
Input: A graph G , integers $d, k, C \in \mathbb{N}^+ \cup \{0\}$ and functions $f : V(G) \rightarrow \{1, 2, \dots, d\}$ and $c : \binom{V(G)}{2} \rightarrow \mathbb{N}^+ \cup \{0\}$
Question: Does there exist a connected graph F such that for every vertex v , $d_F(v) = f(v)$, $|E(G) \Delta E(F)| \leq k$, and $c(E(G) \Delta E(F)) \leq C$?

Theorem 15.5. EDITING TO CONNECTED f -DEGREE GRAPH WITH COSTS is $W[1]$ -hard for trees when parameterized by $k+d$ even if costs are restricted to be 0 or 1.

Proof. We reduce the CLIQUE problem that is well known to be $W[1]$ -complete [41].

In this problem we are given an undirected graph G and a positive integer k as an

input and the objective is to check whether G has a clique of size at least k . It is straightforward to observe that CLIQUE is $W[1]$ -complete for the instances where k is restricted to be odd. To see this, it is sufficient to notice that if G' is the graph obtained from a graph G by adding a vertex that is adjacent to every other vertex of G , then G has a clique of size k if and only if G' has a clique of size $k + 1$.

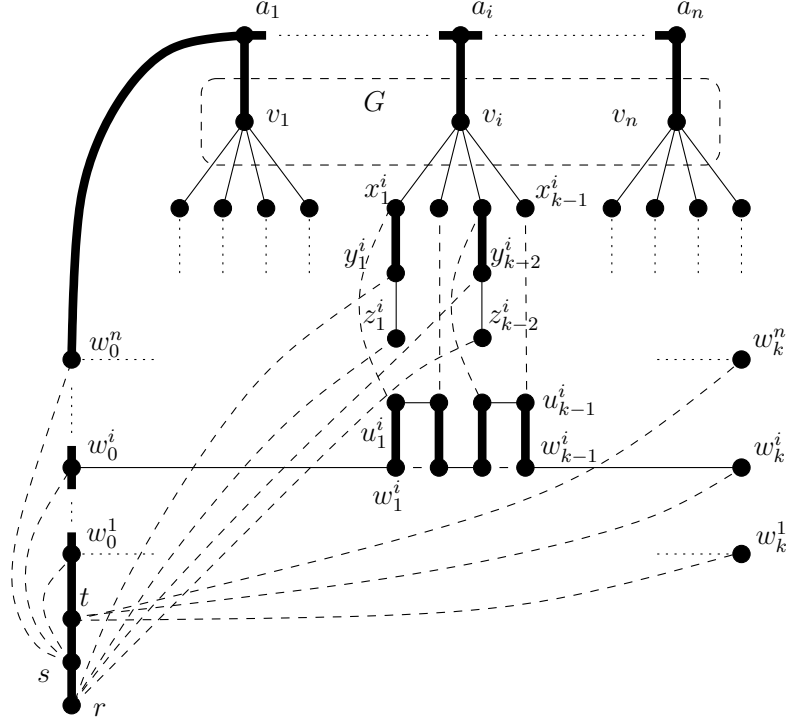


Figure 15.1: Construction of T . The edges of cost 0 are shown by thin lines, the edges of cost 1 are shown by thick lines and the non-edges of cost 0 are shown by dashed lines. Notice that the graph G is encoded by assigning the cost 0 to every non-edge of T corresponding to an edge of G .

Let (G, k) be an instance of CLIQUE and $k \geq 3$ is odd. Let $V(G) = \{v_1, \dots, v_n\}$.

We construct the tree T and define the function c as follows.

- i) Construct vertices v_1, \dots, v_n and set $c(v_i v_j) = 0$ if $v_i v_j \in E(G)$ for $i, j \in \{1, \dots, n\}$.
- ii) For each $i \in \{1, \dots, n\}$, construct vertices $a_i, x_1^i, \dots, x_{k-1}^i, y_1^i, y_3^i, y_5^i, \dots, y_{k-2}^i$ and $z_1^i, z_3^i, z_5^i, \dots, z_{k-2}^i$ and edges $a_i v_i, v_i x_1^i, \dots, v_i x_{k-1}^i, x_1^i y_1^i, x_3^i y_3^i, \dots, x_{k-2}^i y_{k-2}^i$ and $y_1^i z_1^i, y_3^i z_3^i, y_5^i z_5^i, \dots, y_{k-2}^i z_{k-2}^i$. Set $c(a_i v_i) = 1, c(v_i x_1^i) = \dots = c(v_i x_{k-1}^i) =$

0, $c(x_1^i y_1^i) = c(x_3^i y_3^i) = \dots = c(x_{k-2}^i y_{k-2}^i) = 1$ and $c(y_1^i z_1^i) = c(y_3^i z_3^i) = \dots = c(y_{k-2}^i z_{k-2}^i) = 0$.

iii) For each $i \in \{1, \dots, n\}$, construct vertices u_1^i, \dots, u_{k-1}^i and w_0^i, \dots, w_k^i and edges $u_1^i w_1^i, \dots, u_{k-1}^i w_{k-1}^i, w_0^i w_1^i, w_2^i w_3^i, \dots, w_{k-1}^i w_k^i$ and $u_1^i u_2^i, u_3^i u_4^i, \dots, u_{k-2}^i u_{k-1}^i$. Set $c(u_1^i w_1^i) = \dots = c(u_{k-1}^i w_{k-1}^i) = 1$, $c(w_0^i w_1^i) = c(w_2^i w_3^i) = \dots = c(w_{k-1}^i w_k^i) = 0$ and $c(u_1^i u_2^i) = c(u_3^i u_4^i) = \dots = c(u_{k-2}^i u_{k-1}^i) = 0$. Set $c(w_1^i w_2^i) = c(w_3^i w_4^i) = \dots = c(w_{k-2}^i w_{k-1}^i) = 0$.

iv) For each $i \in \{1, \dots, n\}$, set $c(u_1^i x_1^i) = \dots = c(u_{k-1}^i x_{k-1}^i) = 0$.

v) Construct vertices s, t, r and edges $rs, st, tw_0^1, w_0^1 w_0^2, w_0^2 w_0^3, \dots, w_0^{n-1} w_0^n, w_0^n a_1$ and $a_1 a_2, a_2 a_3, \dots, a_{n-1} a_n$. Set $c(rs) = c(st) = c(tw_0^1) = 1$, $c(w_0^1 w_0^2) = c(w_0^2 w_0^3) = \dots = c(w_0^{n-1} w_0^n) = 1$, $c(w_0^n a_1) = 1$ and $c(a_1 a_2) = c(a_2 a_3) = \dots = c(a_{n-1} a_n) = 1$.

vi) Set $c(sw_0^1) = \dots = c(sw_0^n) = 0$, $c(tw_k^1) = \dots = c(tw_k^n) = 0$.

vii) For each $i \in \{1, \dots, n\}$, set $c(ry_1^i) = c(ry_3^i) = \dots = c(ry_{k-2}^i) = 0$ and $c(rz_1^i) = c(rz_3^i) \dots = c(rz_{k-2}^i) = 0$.

viii) For any $pq \in \binom{V(T)}{2} \setminus E(T)$, set $c(pq) = 1$ if $c(pq)$ was not set to be 0 in i)–vii).

We define $f(s) = k+2$, $f(t) = k+2$, $f(r) = 1+k(k-1)$ and set $f(p) = d_T(p)$ for $p \in V(G) \setminus \{s, t, r\}$. Finally, we set $C = 0$, $d = 2+k(k-1)$ and $k' = 5k^2 - 2k + k(k-1)/2$ and obtain an instance (T, d, k', C, f, c) of EDITING TO CONNECTED f -DEGREE GRAPH WITH COSTS. Clearly, T is a tree. We show that (T, d, k', C, f, c) is a YES-instance of EDITING TO CONNECTED f -DEGREE GRAPH WITH COSTS if and only if G has a clique of size k .

For $i \in \{1, \dots, n\}$, let

$$D_i = \{w_0^i w_1^i, w_2^i w_3^i, \dots, w_{k-1}^i w_k^i\} \cup \{u_1^i u_2^i, u_3^i u_4^i, \dots, u_{k-2}^i u_{k-1}^i\} \cup \\ \{v_i x_1^i, \dots, v_i x_{k-1}^i\} \cup \{y_1^i z_1^i, y_3^i z_3^i, \dots, y_{k-2}^i z_{k-2}^i\},$$

and

$$A_i = \{sw_0^i, tw_k^i\} \cup \{w_1^i w_2^i, w_3^i w_4^i, \dots, w_{k-2}^i w_{k-1}^i\} \cup \{x_1^i u_1^i, \dots, x_{k-1}^i u_{k-1}^i\} \cup \\ \{ry_1^i, ry_3^i, \dots, ry_{k-2}^i\} \cup \{rz_1^i, rz_3^i, \dots, rz_{k-2}^i\}.$$

Notice that $D_i \subseteq E(T)$, $c(D_i) = 0$, $|D_i| = 2k - 1 + (k - 1)/2$ and $A_i \subseteq \binom{V(T)}{2} \setminus E(T)$, $c(A_i) = 0$, $|A_i| = 2k + (k - 1)/2$ for $i \in \{1, \dots, n\}$.

Suppose that G has a clique $K = \{v_{i_1}, \dots, v_{i_k}\}$. Let $A' = \{v_{i_j} v_{i_h} \mid 1 \leq j < h \leq k\}$. Because K is a clique in G , $c(A') = 0$. Clearly, $A' \subseteq \binom{V(T)}{2} \setminus E(T)$ and $|A'| = k(k - 1)/2$. We let $D = \bigcup_{j=1}^k D_{i_j}$ and $A = A' \cup (\bigcup_{j=1}^k A_{i_j})$. It is straightforward to verify that $c(D \cup A) = 0$, $|D| + |A| = k'$, $G' = T - D + A$ is a connected graph and for every $p \in V(G')$, $d_{G'}(p) = f(p)$.

Assume now that (T, d, k', C, f, c) is a YES-instance of EDITING TO CONNECTED f -DEGREE GRAPH WITH COSTS. Then there are sets $D \subseteq E(T)$ and $A \subseteq \binom{V(T)}{2}$ such that $|D| + |A| \leq k'$, $c(D \cup A) = 0$, $G' = T - D + A$ is a connected graph and for every $p \in V(G')$, $d_{G'}(p) = f(p)$. Because $f(s) = k + 2$ and $d_T(s) = 2$, A contains at least k edges incident to s . Since $c(A) = 0$, we have that there are $sw_0^{i_1}, \dots, sw_0^{i_k} \in A$ for some distinct $i_1, \dots, i_k \in \{1, \dots, n\}$.

Consider $sw_0^{i_j}$ for some $j \in \{1, \dots, k\}$. Because $f(w_0^{i_j}) = d_T(w_0^{i_j})$, D has an edge of cost 0 incident to $w_0^{i_j}$. Hence, $w_0^{i_j} w_1^{i_j} \in D$. Now we consider $w_1^{i_j}$ and observe that there is an edge of cost 0 in A that is incident to $w_1^{i_j}$ and, therefore, $w_1^{i_j} w_2^{i_j} \in A$. Repeating these arguments, we conclude that

$$R = \{w_0^{i_j} w_1^{i_j}, w_1^{i_j} w_2^{i_j}, \dots, w_{k-1}^{i_j} w_k^{i_j}\} \subseteq D \quad \text{and}$$

$$S = \{sw_0^{i_j}, w_1^{i_j} w_2^{i_j}, \dots, w_{k-2}^{i_j} w_{k-1}^{i_j}, w_k^{i_j} t\} \subseteq A.$$

Consider $F = T - R + S$. Observe that for any $h \in \{1, \dots, (k - 1)/2\}$, we have that $F[\{w_{2h-1}^{i_j}, w_{2h}^{i_j}, u_{2h-1}^{i_j}, u_{2h}^{i_j}\}]$ is a component of F . Since G' is connected, A has an edge

incident to a vertex of each component of this type. We have that for $h \in \{1, \dots, (k-1)/2\}$, $x_{2h-1}^{i_j} u_{2h-1}^{i_j} \in A$ or $x_{2h}^{i_j} u_{2h}^{i_j} \in A$. As $f(u_{2h-1}^{i_j}) = d_T(u_{2h-1}^{i_j})$ and $f(u_{2h}^{i_j}) = d_T(u_{2h}^{i_j})$, D has an edge incident to one of these vertices and, therefore, $u_{2h-1}^{i_j} u_{2h}^{i_j} \in D$ and $x_{2h-1}^{i_j} u_{2h-1}^{i_j}, x_{2h}^{i_j} u_{2h}^{i_j} \in A$. Because $f(x_{2h-1}^{i_j}) = d_T(x_{2h-1}^{i_j})$, $x_{2h-1}^{i_j} v_{i_j}, x_{2h}^{i_j} v_{i_j} \in D$. We obtain that

$$R' = R \cup \{u_1^{i_j} u_2^{i_j}, u_3^{i_j} u_4^{i_j}, \dots, u_{k-2}^{i_j} u_{k-1}^{i_j}\} \cup \{v_{i_j} x_1^{i_j}, \dots, v_{i_j} x_{k-1}^{i_j}\} \subseteq D$$

and

$$S' = S \cup \{x_1^{i_j} u_1^{i_j}, \dots, x_{k-1}^{i_j} u_{k-1}^{i_j}\} \subseteq A.$$

Let $F' = T - R' + S'$. Now we have that for $h \in \{1, \dots, (k-1)/2\}$,

$F'[\{w_{2h-1}^{i_j}, w_{2h}^{i_j}, u_{2h-1}^{i_j}, u_{2h}^{i_j}, x_{2h-1}^{i_j}, x_{2h}^{i_j}, y_{2h-1}^{i_j}, z_{2h-1}^{i_j}\}]$ is a component of F' . Because G' is connected, $ry_{2h-1}^{i_j} \in A$ or $rz_{2h-1}^{i_j} \in A$. As $f(y_{2h-1}^{i_j}) = d_T(y_{2h-1}^{i_j})$ and $f(z_{2h-1}^{i_j}) = d_T(z_{2h-1}^{i_j})$, $y_{2h-1}^{i_j} z_{2h-1}^{i_j} \in D$ and $ry_{2h-1}^{i_j}, rz_{2h-1}^{i_j} \in A$. We conclude that $D_{i_j} \subseteq D$ and $A_{i_j} \subseteq A$. Let

$$R'' = \bigcup_{j=1}^k D_{i_j}, \quad S'' = \bigcup_{j=1}^k A_{i_j}.$$

We have that $R'' \subseteq D$ and $S'' \subseteq A$. Notice that $|R''| + |S''| = 5k^2 - 2k = k' - k(k-1)/2$. Consider $F'' = T - R'' + S''$. For $j \in \{1, \dots, k\}$, $d_{F''}(v_{i_j}) = d_T(v_{i_j}) - (k-1)$. It implies that $A' = \{v_{i_j} v_{i_h} | 1 \leq j < h \leq k\} \subseteq A$ and $c(A') = 0$. Hence, $K = \{v_{i_1}, \dots, v_{i_k}\}$ is a clique in G . \square

Chapter 16

Representative Family

Computation for a Product Family in a Linear Matroid

Let $M = (E, \mathcal{I})$ be a matroid and \mathcal{A} and \mathcal{B} be two subsets of \mathcal{I} . Then we define the product of \mathcal{A} and \mathcal{B} , denoted by $\mathcal{A} \odot \mathcal{B}$ as,

$$\mathcal{A} \odot \mathcal{B} = \{A \cup B : A \cup B \in \mathcal{I}, A \in \mathcal{A}, B \in \mathcal{B} \text{ and } A \cap B = \emptyset\}.$$

That is $\mathcal{A} \odot \mathcal{B} = (\mathcal{A} \bullet \mathcal{B}) \cap \mathcal{I}$. In this chapter we give an algorithm for computing representative families of product families in a linear matroid. A naive approach for computing a representative family of $\mathcal{F} = \mathcal{A} \odot \mathcal{B}$ would be to compute the product $\mathcal{A} \odot \mathcal{B}$ first and then compute a representative family of the product. By applying Theorem 13.1, we can compute a q -representative family of a p -family \mathcal{F} of independent sets in time $\binom{p+q}{p}^{\omega-1} |\mathcal{F}|$. In this chapter we give an algorithm that significantly outperforms the naive approach. An appealing feature of our algorithm is that it works by reducing the computation of a representative family for \mathcal{F} to the computation of representative families for many smaller families. Thus an improved algorithm for the computation of representative sets for general families will automatically accelerate our algorithm for product families as well. In this chapter we prove the following theorem.

Theorem 16.1. *Let $M = (E, \mathcal{I})$ be a linear matroid of rank k , \mathcal{L}_1 be a p_1 -family of independent sets of M and \mathcal{L}_2 be a p_2 -family of independent sets of M . Given a representation A_M of M over a field \mathbb{F} , we can find $\widehat{\mathcal{L}_1 \odot \mathcal{L}_2} \subseteq_{\minrep}^{k-p_1-p_2} \mathcal{L}_1 \odot \mathcal{L}_2$*

of size at most $\binom{k}{p_1+p_2}$ in $\mathcal{O}\left(|\mathcal{L}_2||\mathcal{L}_1|\binom{k-p_2}{p_1}^{\omega-1}p_1^\omega + |\mathcal{L}_2|\binom{k-p_2}{p_1}\binom{k}{p_1+p_2}^{\omega-1}(p_1+p_2)^\omega\right)$ operations over \mathbb{F} .

Algorithm. We give an algorithm to compute q -representative family for product families of a linear matroid. That is, given a matroid $M = (E, \mathcal{I})$, families of independent sets \mathcal{A} and \mathcal{B} of sizes p_1 and p_2 respectively, and a positive integer q , we compute $\widehat{\mathcal{F}} \subseteq_{rep}^q \mathcal{F}$, where, $\mathcal{F} = \mathcal{A} \odot \mathcal{B}$, of size $\binom{p_1+p_2+q}{p_1+p_2}$ efficiently. We compute a q -representative family for \mathcal{F} in two steps. In the first step we compute an intermediate q -representative family and then apply Theorem 13.2 to compute q -representative family of the desired size. The intermediate q -representative family is obtained by computing q -representative families of slices, $\mathcal{A} \odot \{B\}$ for all $B \in \mathcal{B}$, and then take its union. We start with the following lemma that will be central to our faster algorithm for computing the desired q -representative family for a product family of a linear matroid.

Lemma 16.1 (Slice Computation Lemma). *Let $M = (E, \mathcal{I})$ be a linear matroid of rank k , \mathcal{L} be a p_1 -family of independent sets of M and $S \in \mathcal{I}$ of size p_2 . Furthermore, let $w : \mathcal{L} \odot \{S\} \rightarrow \mathbb{N}$ be a non-negative weight function. Then given a representation A_M of M over a field \mathbb{F} , we can find $\widehat{\mathcal{L} \odot \{S\}} \subseteq_{minrep}^{k-p_1-p_2} \mathcal{L} \odot \{S\}$ of size at most $\binom{k-p_2}{p_1}$ in $\mathcal{O}\left(\binom{k-p_2}{p_1}|\mathcal{L}|p_1^\omega + |\mathcal{L}|\binom{k-p_2}{p_1}^{\omega-1}\right)$ operations over \mathbb{F} .*

Proof. Observe that $\mathcal{L} \odot \{S\}$ is a $p_1 + p_2$ -family of independent sets of M and all sets in $\mathcal{L} \odot \{S\}$ contain S as a subset. Let A_M the matrix representing the matroid M over a field \mathbb{F} . Without loss of generality we can assume that the first p_2 columns of A_M correspond to the elements in S . Furthermore, we can also assume that the first p_2 columns and p_2 rows form an identity matrix $I_{p_2 \times p_2}$. That is, if S denotes the first p_2 columns and Z denotes the first p_2 rows then the submatrix $A_M[Z, S]$ is $I_{p_2 \times p_2}$. The reason for the last assertion is that if the matrix is not in the required form then we can apply elementary row operations and obtain the matrix in the desired form. This also allows us to assume that the number of rows in A_M is k . So A_M have the following form.

$$\left(\begin{array}{c|c} I_{p_2 \times p_2} & A \\ \hline 0 & B \end{array} \right)$$

Let $A_{M/S}$ be the matrix obtained after deleting first p_2 rows and first p_2 columns from A_M . That is, $A_{M/S} = B$. Let $M/S = (E_s, \mathcal{I}_s)$ be the matroid represented by the matrix $A_{M/S}$ on the underlying ground set $E_s = E \setminus S$. Observe that the $\text{rank}(M/S) = \text{rank}(B) = k - p_2$, else the $\text{rank}(A_M)$ would become strictly smaller than k . Let e_1, e_2, \dots, e_{p_2} be the first p_2 column vectors of A_M , i.e., they are columns corresponding to the elements of S . For a column vector v in A_M , \bar{v} is used to denote the column vector restricted to the matrix $A_{M/S}$ (i.e., \bar{v} contains the last $k - p_2$ entries of v).

Now consider the set $\mathcal{L}(S) = \{X \mid X \cup S \in \mathcal{L} \odot \{S\}\}$. We also define a new non-negative weight function $w' : \mathcal{L}(S) \rightarrow \mathbb{N}$ as follows: $w'(X) = w(X \cup S)$. We would like to compute $k - p_2$ representative for $\mathcal{L}(S)$. Towards that goal we first show that $\mathcal{L}(S)$ is a p_1 -family of independent sets of M/S . Let $X \in \mathcal{L}(S)$. We know that $X \cup S \in \mathcal{I}$. Let v_1, v_2, \dots, v_{p_1} be the column vectors in A_M corresponding to the elements in X . Suppose $X \notin \mathcal{I}_s$. Then there exist coefficients $\lambda_1, \dots, \lambda_{p_1}$ such that $\lambda_1 \bar{v}_1 + \lambda_2 \bar{v}_2 + \dots + \lambda_{p_1} \bar{v}_{p_1} = \vec{0}$ and at least one of them is non-zero. Then

$$\lambda_1 v_1 + \lambda_2 v_2 + \dots + \lambda_{p_1} v_{p_1} = \begin{pmatrix} a_1 \\ \vdots \\ a_{p_2} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

This implies that $-a_1 e_1 - a_2 e_2 - \dots - a_{p_2} e_{p_2} + \lambda_1 v_1 + \lambda_2 v_2 + \dots + \lambda_{p_1} v_{p_1} = \vec{0}$, which contradicts the fact that $S \cup X \in \mathcal{I}$. Hence $X \in \mathcal{I}_s$ and $\mathcal{L}(S)$ is a p_1 -family of independent sets of M/S .

Now we apply Theorem 13.2 and find $\widehat{\mathcal{L}(S)} \subseteq_{\text{minrep}}^{k-p_1-p_2} \mathcal{L}(S)$ of size $\binom{k-p_2}{p_1}$, by considering $\mathcal{L}(S)$ as a p_1 -family of independent sets of the matroid M/S . We claim that $\widehat{\mathcal{L}(S)} \odot \{S\} \subseteq_{\text{minrep}}^{k-p_1-p_2} \mathcal{L} \odot \{S\}$. Let $X \cup S \in \mathcal{L} \odot \{S\}$ and $Y \subseteq E \setminus (X \cup S)$ such that

$|Y| = k - p_1 - p_2$ and $X \cup S \cup Y \in \mathcal{I}$. We need to show that there exists a $\widehat{X} \in \widehat{\mathcal{L}(S)}$ such that $\widehat{X} \cup S \cup Y \in \mathcal{I}$ and $w(\widehat{X} \cup S) \leq w(X \cup S)$. We start by showing that $X \cup Y \in \mathcal{I}_s$. Let $v_1, v_2, \dots, v_{k-p_2}$ be the column vectors in A_M corresponding to the elements of $X \cup Y$. Suppose $X \cup Y \notin \mathcal{I}_s$. Then there exist coefficients $\lambda_1, \dots, \lambda_{k-p_2}$ such that $\lambda_1 \bar{v}_1 + \lambda_2 \bar{v}_2 + \dots + \lambda_{k-p_2} \bar{v}_{k-p_2} = \vec{0}$ and at least one of them is non-zero. Then we have the following.

$$\lambda_1 v_1 + \lambda_2 v_2 + \dots + \lambda_{k-p_2} v_{k-p_2} = \begin{pmatrix} b_1 \\ \vdots \\ b_{p_2} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

However this implies that $-b_1 e_1 - b_2 e_2 - \dots - b_{p_2} e_{p_2} + \lambda_1 v_1 + \lambda_2 v_2 + \dots + \lambda_{k-p_2} v_{k-p_2} = \vec{0}$, which contradicts the fact that $S \cup X \cup Y \in \mathcal{I}$. Hence $X \cup Y \in \mathcal{I}_s$. Since $\widehat{\mathcal{L}(S)} \subseteq_{\minrep}^{k-p_1-p_2} \mathcal{L}(S)$, there exists a set $\widehat{X} \in \mathcal{L}(S)$, with $w'(\widehat{X}) \leq w'(X)$ (i.e $w(\widehat{X} \cup S) \leq w(X \cup S)$) and $\widehat{X} \cup Y \in \mathcal{I}_s$. We claim that $\widehat{X} \cup S \cup Y \in \mathcal{I}$. Let $u_1, u_2, \dots, u_{k-p_2}$ be the column vectors in A_M corresponding to the elements of $\widehat{X} \cup Y$. Suppose $\widehat{X} \cup S \cup Y \notin \mathcal{I}$. Then there exist coefficients $\alpha_1, \dots, \alpha_k$ such that $\alpha_1 e_1 + \alpha_2 e_2 + \dots + \alpha_{p_2} e_{p_2} + \alpha_{p_2+1} u_1 + \dots + \alpha_k u_{k-p_2} = \vec{0}$ and at least one of the coefficients is non-zero. We claim that at least one of the coefficients among $\{\alpha_{p_2+1}, \dots, \alpha_k\}$ is non-zero. Suppose not, then $\alpha_1 e_1 + \dots + \alpha_{p_2} e_{p_2} = 0$ and at least one of the coefficients among $\{\alpha_1, \dots, \alpha_{p_2}\}$ is non-zero. This contradicts the fact that $S \in \mathcal{I}$. Since $\alpha_1 e_1 + \dots + \alpha_{p_2} e_{p_2} + \alpha_{p_2+1} u_1 + \dots + \alpha_k u_{k-p_2} = \vec{0}$, we have that $\alpha_{p_2+1} \bar{u}_1 + \dots + \alpha_k \bar{u}_{k-p_2} = \vec{0}$, where \bar{u}_j are restrictions of u_j to the last $k - p_2$ entries. Also note that at least one of the coefficients among $\{\alpha_{p_2+1}, \dots, \alpha_k\}$ is non-zero. This contradicts our assumption that $\widehat{X} \cup Y \in \mathcal{I}_s$. Thus we have shown that $\widehat{X} \cup Y \cup S \in \mathcal{I}$. The size of $\widehat{\mathcal{L}(S)} \odot \{S\}$ is $\binom{k-p_2}{p_1}$ and it can be found in $\mathcal{O}\left(\binom{k-p_2}{p_1} |\mathcal{L}| p_1^\omega + |\mathcal{L}| \binom{k-p_2}{p_1} \omega^{-1}\right)$ operations over \mathbb{F} . \square

Now we are ready to prove the main Theorem 16.1 by using Lemma 16.1.

Theorem 16.1 *Let $M = (E, \mathcal{I})$ be a linear matroid of rank k , \mathcal{L}_1 be a p_1 -family of independent sets of M and \mathcal{L}_2 be a p_2 -family of independent sets of M . Given a representation A_M of M over a field \mathbb{F} , we can find $\widehat{\mathcal{L}}_1 \odot \widehat{\mathcal{L}}_2 \subseteq_{\minrep}^{k-p_1-p_2} \mathcal{L}_1 \odot \mathcal{L}_2$ of size at most $\binom{k}{p_1+p_2}$ in $\mathcal{O}\left(|\mathcal{L}_2| |\mathcal{L}_1| \binom{k-p_2}{p_1}^{\omega-1} p_1^\omega + |\mathcal{L}_2| \binom{k-p_2}{p_1} \binom{k}{p_1+p_2}^{\omega-1} (p_1+p_2)^\omega\right)$ operations over \mathbb{F} .*

Proof. Let $\mathcal{L}_2 = \{S_1, S_2, \dots, S_\ell\}$. Then we have

$$\mathcal{L}_1 \odot \mathcal{L}_2 = \bigcup_{i=1}^{\ell} \mathcal{L}_1 \odot \{S_i\}.$$

By Lemma 7.2,

$$\mathcal{L} = \bigcup_{i=1}^{\ell} \widehat{\mathcal{L}}_1 \odot \{S_i\} \subseteq_{\minrep}^{k-p_1-p_2} \mathcal{L}_1 \odot \mathcal{L}_2.$$

Using Lemma 16.1, for all $1 \leq i \leq \ell$, we find $\widehat{\mathcal{L}}_1 \odot \{S_i\} \subseteq_{\minrep}^{k-p_1-p_2} \mathcal{L}_1 \odot \{S_i\}$ of size $\binom{k-p_2}{p_1}$ in $\mathcal{O}\left(\binom{k-p_2}{p_1} |\mathcal{L}_1| p_1^\omega + |\mathcal{L}_1| \binom{k-p_2}{p_1}^{\omega-1}\right) = \mathcal{O}\left(|\mathcal{L}_1| \binom{k-p_2}{p_1}^{\omega-1} p_1^\omega\right)$ operations over \mathbb{F} . Now $|\mathcal{L}| = |\bigcup_{i=1}^{\ell} \widehat{\mathcal{L}}_1 \odot \{S_i\}| \leq |\mathcal{L}_2| \binom{k-p_2}{p_1}$. Now we apply Theorem 13.2 and find $\widehat{\mathcal{L}} \subseteq_{\minrep}^{k-p_1-p_2} \mathcal{L}$ of size $\binom{k}{p_1+p_2}$. The number of operations, denoted by T_1 , over \mathbb{F} to find $\widehat{\mathcal{L}}$ from \mathcal{L} is

$$\begin{aligned} T_1 &= \mathcal{O}\left(\binom{k}{p_1+p_1} |\mathcal{L}_2| \binom{k-p_2}{p_1} (p_1+p_2)^\omega + |\mathcal{L}_2| \binom{k-p_2}{p_1} \binom{k}{p_1+p_2}^{\omega-1}\right) \\ &= \mathcal{O}\left(|\mathcal{L}_2| \binom{k-p_2}{p_1} \binom{k}{p_1+p_2}^{\omega-1} (p_1+p_2)^\omega\right). \end{aligned}$$

By Lemma 7.1, $\widehat{\mathcal{L}} \subseteq_{\minrep}^{k-p_1-p_2} \mathcal{L}_1 \odot \mathcal{L}_2$. The number of operations, denoted by T , over \mathbb{F} to find $\widehat{\mathcal{L}}$ from \mathcal{L}_1 and \mathcal{L}_2 is

$$\begin{aligned} T &= |\mathcal{L}_2| \cdot \mathcal{O}\left(|\mathcal{L}_1| \binom{k-p_2}{p_1}^{\omega-1} p_1^\omega\right) + T_1 \\ &= \mathcal{O}\left(|\mathcal{L}_2| |\mathcal{L}_1| \binom{k-p_2}{p_1}^{\omega-1} p_1^\omega + |\mathcal{L}_2| \binom{k-p_2}{p_1} \binom{k}{p_1+p_2}^{\omega-1} (p_1+p_2)^\omega\right). \end{aligned}$$

This completes the proof of the theorem. \square

The following form of Theorem 16.1 will be directly useful in some applications.

Corollary 16.1. *Let $M = (E, \mathcal{I})$ be a linear matroid of rank k , \mathcal{L}_1 and \mathcal{L}_2 be two families of independent sets of M and the number of sets of size p in \mathcal{L}_1 and \mathcal{L}_2 be at most $\binom{k+c}{p}$. Here, c is a fixed constant. Let $\mathcal{L}_{r,i}$ be the set of independent sets of size exactly i in \mathcal{L}_r for $r \in \{1, 2\}$. Then for all the pairs $i, j \in [k]$, we can find $\widehat{\mathcal{L}_{1,i} \odot \mathcal{L}_{2,j}} \subseteq_{\min rep}^{k-i-j} \mathcal{L}_{1,i} \odot \mathcal{L}_{2,j}$ of size $\binom{k}{i+j}$, in total of $\mathcal{O}\left(k^\omega (2^\omega + 2)^k + k^\omega 2^{k(\omega-1)} 3^k\right)$ operations over \mathbb{F} .*

Proof. By using Theorem 16.1 we can find $\widehat{\mathcal{L}_{1,i} \odot \mathcal{L}_{2,j}} \subseteq_{\min rep}^{k-i-j} \mathcal{L}_{1,i} \odot \mathcal{L}_{2,j}$ of size $\binom{k}{i+j}$ for any $i, j \in [k]$ in $\mathcal{O}\left(\binom{k+c}{j} \binom{k+c}{i} \binom{k-j}{i}^{\omega-1} i^\omega + \binom{k+c}{j} \binom{k-j}{i} \binom{k}{i+j}^{\omega-1} (i+j)^\omega\right)$ operations over \mathbb{F} . Let $k' = k + c$. So the total number of operations, denoted by T , over \mathbb{F} to find $\widehat{\mathcal{L}_{1,i} \odot \mathcal{L}_{2,j}}$ for all $i, j \in [k]$ is,

$$\begin{aligned}
T &= \mathcal{O}\left(\left(\sum_{i=0}^k \sum_{j=0}^k \binom{k'}{j} \binom{k'}{i} \binom{k-j}{i}^{\omega-1} i^\omega\right) + \right. \\
&\quad \left. \left(\sum_{i=0}^k \sum_{j=0}^k \binom{k'}{j} \binom{k-j}{i} \binom{k}{i+j}^{\omega-1} (i+j)^\omega\right)\right) \\
&= \mathcal{O}\left(\left(k^\omega \sum_{i=0}^k \binom{k'}{i} \sum_{j=0}^k \binom{k'}{j} 2^{(k-j)(\omega-1)}\right) + \right. \\
&\quad \left. \left(k^\omega \sum_{j=0}^k \binom{k'}{j} \sum_{i=0}^{k-j} \binom{k-j}{i} \binom{k}{i+j}^{\omega-1}\right)\right) \\
&= \mathcal{O}\left(\left(k^\omega 2^{k(\omega-1)} \sum_{i=0}^k \binom{k'}{i} \left(1 + \frac{1}{2^{(\omega-1)}}\right)^{k'}\right) + \right. \\
&\quad \left. \left(k^\omega 2^{k(\omega-1)} \sum_{j=0}^k \binom{k'}{j} \sum_{i=0}^{k-j} \binom{k-j}{i}\right)\right) \\
&= \mathcal{O}\left(\left(k^\omega 2^{k'} (2^{(\omega-1)} + 1)^k\right) + \left(k^\omega 2^{k(\omega-1)} \sum_{j=0}^k \binom{k'}{j} 2^{k-j}\right)\right) \\
&= \mathcal{O}\left(k^\omega 2^k (2^{(\omega-1)} + 1)^k + k^\omega 2^{k(\omega-1)} 3^k\right) \\
&= \mathcal{O}\left(k^\omega (2^\omega + 2)^k + k^\omega 2^{k(\omega-1)} 3^k\right).
\end{aligned}$$

The above simplification completes the proof. \square

Chapter 17

Dynamic Programming over graphs of bounded treewidth

In this chapter we show how the fast computation of representative family of a product family in linear matroid is helpful to design fast FPT algorithms for problems over graphs of bounded treewidth. It is well known that many intractable problems can be solved efficiently when the input graph has bounded treewidth. Moreover, many fundamental problems like MAXIMUM INDEPENDENT SET or MINIMUM DOMINATING SET can be solved in time $2^{\mathcal{O}(t)}n$. On the other hand, it was believed until very recently that for some “connectivity” problems such as HAMILTONIAN CYCLE or STEINER TREE no such algorithm exists. In their breakthrough paper, Cygan et al. [38] introduced a new algorithmic framework called Cut&Count and used it to obtain $2^{\mathcal{O}(t)}n^{\mathcal{O}(1)}$ time Monte Carlo algorithms for a number of connectivity problems. Recently, Bodlaender et al. [23] obtained the first deterministic single-exponential algorithms for these problems using two novel approaches. One of the approaches of Bodlaender et al. is based on rank estimations in specific matrices and the second based on matrix-tree theorem and computation of determinants.

It is interesting to note that for a number of connectivity problems such as STEINER TREE or FEEDBACK VERTEX SET the “bottleneck” of treewidth based dynamic programming algorithms is the *join* operation. For example, as it was shown by Bodlaender et al. in [23], FEEDBACK VERTEX SET and STEINER TREE can be solved in time $\mathcal{O}((1 + 2^\omega)^{\mathbf{pw}} \mathbf{pw}^{\mathcal{O}(1)} n)$ and $\mathcal{O}((1 + 2^{\omega+1})^{\mathbf{tw}} \mathbf{tw}^{\mathcal{O}(1)} n)$, where \mathbf{pw} and \mathbf{tw} are the pathwidth and the treewidth of the input graph. The reason for the difference in the exponents of these two algorithms is due to the cost of the join operation, which is required for treewidth and does not occur for pathwidth.

Our approach to solve these problems is based on representative families in matroids. The main idea behind our approach is that all the relevant information about “partial solutions” in bags of the tree decomposition, can be encoded as an independent

set of graphic matroid. For many computational problems on graphs of bounded treewidth in the join nodes of the decomposition, the family of partial solutions is the product of the families of its children, and we wish to store a representative family (for a graphic matroid) for this product family. Here our algorithm for computation of representative family of a product family in a linear matroid comes into play. By making use of this algorithm one can obtain faster deterministic algorithms for many connectivity problems. We exemplify this by providing algorithms with running time $\mathcal{O}((1 + 2^{\omega-1} \cdot 3)^{\mathbf{tw}} \mathbf{tw}^{O(1)} n)$ for STEINER TREE (in Section 17.1) and FEEDBACK VERTEX SET (in Section 17.2).

Before explaining our algorithms for STEINER TREE and FEEDBACK VERTEX SET, we first define treewidth formally. Let G be a graph. A *tree-decomposition* of a graph G is a pair $(\mathbb{T}, \mathcal{X} = \{X_t\}_{t \in V(\mathbb{T})})$ such that

- $\bigcup_{t \in V(\mathbb{T})} X_t = V(G)$,
- for every edge $xy \in E(G)$ there is a $t \in V(\mathbb{T})$ such that $\{x, y\} \subseteq X_t$, and
- for every vertex $v \in V(G)$ the subgraph of \mathbb{T} induced by the set $\{t \mid v \in X_t\}$ is connected.

The *width* of a tree decomposition is $\max_{t \in V(\mathbb{T})} |X_t| - 1$ and the *treewidth* of G is the minimum width over all tree decompositions of G and is denoted by $\mathbf{tw}(G)$.

A tree decomposition $(\mathbb{T}, \mathcal{X})$ is called a *nice tree decomposition* if \mathbb{T} is a tree rooted at some node r where $X_r = \emptyset$, each node of \mathbb{T} has at most two children, and each node is of one of the following kinds:

1. **Introduce node:** a node t that has only one child t' where $X_t \supset X_{t'}$ and $|X_t| = |X_{t'}| + 1$.
2. **Forget node:** a node t that has only one child t' where $X_t \subset X_{t'}$ and $|X_t| = |X_{t'}| - 1$.
3. **Join node:** a node t with two children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.
4. **Base node:** a node t that is a leaf of \mathbb{T} , is different than the root, and $X_t = \emptyset$.

Notice that, according to the above definition, the root r of \mathbb{T} is either a forget node or a join node. It is well known that any tree decomposition of G can be transformed into a nice tree decomposition maintaining the same width in linear time [76]. We use G_t to denote the graph induced by the vertex set $\bigcup_{t'} X_{t'}$, where t' ranges over all descendants of t , including t . By $E(X_t)$ we denote the edges present in $G[X_t]$. We use H_t to denote the graph on vertex set $V(G_t)$ and the edge set $E(G_t) \setminus E(X_t)$. For clarity of presentation we use the term nodes to refer to the vertices of the tree \mathbb{T} .

17.1 STEINER TREE

The problem we study in this section is defined below.

<p>STEINER TREE</p> <p>Input: An undirected graph G together with a tree-decomposition $(\mathbb{T}, \mathcal{X})$ of width \mathbf{tw}, $T \subseteq V(G)$ function $w : E(G) \rightarrow \mathbb{N}$.</p> <p>Task: Find a subtree in G of minimum weight spanning all vertices of T.</p>	<p>Parameter: \mathbf{tw}</p>
--	---

Let G be an input graph of the STEINER TREE problem. Throughout this section, we say that $E' \subseteq E(G)$ is a *solution* if the subgraph induced on this edge set is connected and it contains all the terminal vertices. We call $E' \subseteq E(G)$ an *optimal solution* if E' is a solution of the minimum weight. Let \mathcal{S} be a family of edge subsets such that every edge subset corresponds to an optimal solution. That is,

$$\mathcal{S} = \{E' \subseteq E(G) \mid E' \text{ is an optimal solution}\}.$$

Observe that any edge set in \mathcal{S} induces a forest. We start with few definitions that will be useful in explaining the algorithm. Let $(\mathbb{T}, \mathcal{X})$ be a tree decomposition of G of width \mathbf{tw} . Let t be a node of $V(\mathbb{T})$. By \mathcal{S}_t we denote the family of edge subsets of $E(H_t)$, $\{E' \subseteq E(H_t) \mid G[E'] \text{ is a forest}\}$, that satisfies the following properties.

- Either E' is a solution tree (that is, the subgraph induced on this edge set is connected and it contains all the terminal vertices); or
- every vertex of $(T \cap V(G_t)) \setminus X_t$ is incident with some edge from E' , and every connected component of the graph induced by E' contains a vertex from X_t .

We call \mathcal{S}_t a *family of partial solutions* for t . We denote by K^t a complete graph on the vertex set X_t . For an edge subset $E^* \subseteq E(G)$ and bag X_t corresponding to a node t , we define the following.

1. Set $\partial^t(E^*) = X_t \cap V(E^*)$, the set of endpoints of E^* in X_t .
2. Let G^* be the subgraph of G on the vertex set $V(G)$ and the edge set E^* . Let C'_1, \dots, C'_ℓ be the connected components of G^* such that for all $i \in [\ell]$, $C'_i \cap X_t \neq \emptyset$. Let $C_i = C'_i \cap X_t$. Observe that C_1, \dots, C_ℓ is a partition of $\partial^t(E^*)$. By $F_t(E^*)$ we denote a forest $\{Q_1, \dots, Q_\ell\}$ where each Q_i is an arbitrary spanning tree of $K^t[C_i]$. For an example, since $K^t[C_i]$ is a complete graph we could take Q_i as a star. The purpose of $F_t(E^*)$ is to keep track for the vertices in C_i whether they were in the same connected component of G^* .
3. We define $w(F_t(E^*)) = w(E^*)$.

Let \mathcal{A} and \mathcal{B} be two family of edge subsets of $E(G)$, then we define

$$\mathcal{A} \diamond \mathcal{B} = \{E_1 \cup E_2 \mid E_1 \in \mathcal{A} \wedge E_2 \in \mathcal{B} \wedge E_1 \cap E_2 = \emptyset \wedge G[E_1 \cup E_2] \text{ is a forest}\}.$$

With every node t of \mathbb{T} , we associate a subgraph of G . In our case it will be H_t . For every node t , we keep a family of partial solutions for the graph H_t . That is, for every optimal solution $L \in \mathcal{S}$ and its intersection $L_t = E(H_t) \cap L$ with the graph H_t , we have some partial solution in the family that is “as good as L_t ”. More precisely, we have some partial solution, say \hat{L}_t in our family such that $\hat{L}_t \cup L_R$ is also an optimum solution for the whole graph, where $L_R = L \setminus L_t$. As we move from one node t in the decomposition tree to the next node t' the graph H_t changes to $H_{t'}$, and so does the set of partial solutions. The algorithm updates its set of partial solutions accordingly. Here matroids come into play: in order to bound the size of the family of partial solutions that the algorithm stores at each node we employ Theorem 13.2 and Corollary 16.1 for graphic matroids. More details are given in the proof of the following theorem, which is one of the main results in this section.

Theorem 17.1. *Let G be an n -vertex graph given together with its tree decomposition $(\mathbb{T}, \mathcal{X})$ of width \mathbf{tw} . Then STEINER TREE on G can be solved in time $\mathcal{O}\left((1 + 2^{\omega-1} \cdot 3)^{\mathbf{tw}} \mathbf{tw}^{\mathcal{O}(1)} n\right)$.*

Proof. For every node t of \mathbb{T} and subset $Z \subseteq X_t$, we store a family of edge subsets $\hat{\mathcal{S}}_t[Z] \subseteq \mathcal{S}_t$ of H_t satisfying the following correctness invariant.

Correctness Invariant: For every $L \in \mathcal{S}$ we have the following. Let $L_t = E(H_t) \cap L$, $L_R = L \setminus L_t$, and $Z = \partial^t(L)$. Then there exists $\hat{L}_t \in \hat{\mathcal{S}}_t[Z]$ such that $w(\hat{L}_t) \leq w(L_t)$, $\hat{L} = \hat{L}_t \cup L_R$ is a solution, and $\partial^t(\hat{L}) = Z$. Observe that since $w(\hat{L}_t) \leq w(L_t)$ and $L \in \mathcal{S}$, we have that $\hat{L} \in \mathcal{S}$.

We process the nodes of the tree \mathbb{T} from base nodes to the root node while doing the dynamic programming. Throughout the process we maintain the correctness invariant, which will prove the correctness of the algorithm. However, our main idea is to use representative sets to obtain $\hat{\mathcal{S}}_t[Z]$ of small size. That is, given the set $\hat{\mathcal{S}}_t[Z]$ (as a product of two families \mathcal{A} and \mathcal{B} , i.e. $\hat{\mathcal{S}}_t[Z] = \mathcal{A} \diamond \mathcal{B}$) that satisfies the correctness invariant, we use Corollary 16.1 to obtain a subset $\hat{\hat{\mathcal{S}}}_t[Z]$ of $\hat{\mathcal{S}}_t[Z]$ that also satisfies the correctness invariant and has size upper bounded by $2^{|Z|}$ in total. More precisely, the number of partial solutions with i connected components

in $\widehat{\mathcal{S}}'_t[Z]$ is upper bounded by $\binom{|Z|}{|Z|-i} = \binom{|Z|}{i}$. Thus, we maintain the following size invariant.

Size Invariant: After node t of \mathbb{T} is processed by the algorithm, for every $Z \subseteq X_t$ we have that $|\widehat{\mathcal{S}}'_t[Z, i]| \leq \binom{|Z|}{i}$, where $\widehat{\mathcal{S}}'_t[Z, i]$ is the partial solutions with i connected components in $\widehat{\mathcal{S}}'_t[Z]$.

The main ingredient of the dynamic programming algorithm for STEINER TREE is the use of Theorem 13.2 and Corollary 16.1 to compute $\widehat{\mathcal{S}}'_t[Z]$ maintaining the size invariant. The next lemma shows how to implement it.

Lemma 17.1 (Product Shrinking Lemma). *Let t be a node of \mathbb{T} , and let $Z \subseteq X_t$ be a set of size k . Let \mathcal{P} and \mathcal{Q} be two family of edge sets of H_t . Furthermore, let $\widehat{\mathcal{S}}'_t[Z] = \mathcal{P} \diamond \mathcal{Q}$ be the family of edge subsets of H_t satisfying the correctness invariant. If the number of edge sets with i connected components in \mathcal{P} as well as in \mathcal{Q} is bounded by $\binom{k+c}{i}$ where c is some fixed constant, then in time $\mathcal{O}\left(k^\omega (2^\omega + 2)^k n + k^\omega 2^{k(\omega-1)} 3^k n\right)$ we can compute $\widehat{\mathcal{S}}'_t[Z] \subseteq \widehat{\mathcal{S}}_t[Z]$ satisfying correctness and size invariants.*

Proof. We start by associating a matroid with the node t and the set $Z \subseteq X_t$ as follows. We consider a graphic matroid $M = (E, \mathcal{I})$ on $K^t[Z]$. Here, the element set E of the matroid is the edge set $E(K^t[Z])$ and the family of independent sets \mathcal{I} consists of forests of $K^t[Z]$.

Let $\mathcal{P} = \{A_1, \dots, A_\ell\}$ and $\mathcal{Q} = \{B_1, \dots, B_{\ell'}\}$. Let $\mathcal{L}_1 = \{F_t(A_1), \dots, F_t(A_\ell)\}$ and $\mathcal{L}_2 = \{F_t(B_1), \dots, F_t(B_{\ell'})\}$ be the set of forests in $K^t[Z]$ corresponding to the edge subsets in \mathcal{P} and \mathcal{Q} respectively. For $r \in \{1, 2\}$ and $i \in \{1, \dots, k-1\}$, let $\mathcal{L}_{r,i}$ be the family of forests of \mathcal{L}_r with i edges. Now we apply Corollary 16.1 and find $\widehat{\mathcal{L}}_{1,i} \odot \widehat{\mathcal{L}}_{2,j} \subseteq_{\minrep}^{k-1-i-j} \mathcal{L}_{1,i} \odot \mathcal{L}_{2,j}$ of size $\binom{k-1}{i+j}$ for all $i, j \in [k]$ such that $i+j < k$. Let $\widehat{\mathcal{S}}'_t[Z, k-d] \subseteq \widehat{\mathcal{S}}_t[Z, k-d]$ be such that for every $D \in \widehat{\mathcal{S}}'_t[Z, k-d]$ we have that $F_t(D) \in \widehat{\mathcal{L}}_{1,i} \odot \widehat{\mathcal{L}}_{2,j}$. Note that $F_t(D)$ has d edges if and only if $G[D]$ have $k-d$ connected components. Let $\widehat{\mathcal{S}}'_t[Z] = \bigcup_{j=1}^k \widehat{\mathcal{S}}'_t[Z, j]$. By Corollary 16.1, $|\widehat{\mathcal{S}}'_t[Z, k-d]| \leq k \binom{k-1}{d} \leq \binom{k}{k-d}$, and hence $\widehat{\mathcal{S}}'_t[Z]$ maintains the size invariant.

Now we show that the $\widehat{\mathcal{S}}'_t[Z]$ maintains the correctness invariant. Let $L \in \mathcal{S}$. Let $L_t = E(H_t) \cap L$, $L_R = L \setminus L_t$ and $Z = \partial^t(L)$. Since $\widehat{\mathcal{S}}_t[Z]$ satisfies correctness invariant, there exists $L'_t \in \widehat{\mathcal{S}}_t[Z]$ such that $w(L'_t) \leq w(L_t)$, $\widehat{L} = L'_t \cup L_R$ is an optimal solution and $\partial^t(\widehat{L}) = Z$. Since $\widehat{\mathcal{S}}_t[Z] = \mathcal{P} \diamond \mathcal{Q}$, there exists $A \in \mathcal{P}$ and $B \in \mathcal{Q}$ such that $L'_t = A \cup B$. Observe that $G[L'_t], G[A]$ and $G[B]$ form forests. Consider the forests $F_t(A)$ and $F_t(B)$. Suppose $F_t(A)$ has i edges and $F_t(B)$ has j edges, then $F_t(L'_t) \in \mathcal{L}_{1,i} \odot \mathcal{L}_{2,j}$. This is because, if $F_t(L'_t)$ contain a cycle, then corresponding to that cycle we can get a cycle in $G[L'_t]$, which is a contradiction. Now let $F_t(L_R)$ be the forest corresponding to L_R . Since \widehat{L} is a solution, we have that $F_t(L'_t) \cup F_t(L_R)$ is a spanning tree in $K^t[Z]$. Since $\mathcal{L}_{1,j} \widehat{\odot} \mathcal{L}_{2,j} \subseteq_{\text{minrep}}^{k-1-i-j} \mathcal{L}_{1,i} \odot \mathcal{L}_{2,j}$, we have that there exists a forest $F_t(\widehat{L}'_t) \in \mathcal{L}_{1,i} \widehat{\odot} \mathcal{L}_{2,j}$ such that $w(F_t(\widehat{L}'_t)) \leq w(F_t(L'_t))$ and $F_t(\widehat{L}'_t) \cup F_t(L_R)$ is a spanning tree in $K^t[Z]$. Thus, we have that $\widehat{L}'_t \cup L_R$ is an optimum solution and $\widehat{L}'_t \in \widehat{\mathcal{S}}'_t[Z]$. This proves that $\widehat{\mathcal{S}}'_t[Z]$ maintains the correctness invariant.

For a given edge set D , we need to compute the forest $F_t(D)$ and that can take $\mathcal{O}(n)$ time. The running time to compute $\widehat{\mathcal{S}}'_t[Z]$ is,

$$\mathcal{O}\left(k^\omega (2^\omega + 2)^k n + k^\omega 2^{k(\omega-1)} 3^k n\right).$$

□

We now return to the dynamic programming algorithm over the tree-decomposition $(\mathbb{T}, \mathcal{X})$ of G and prove that it maintains the correctness invariant. We assume that $(\mathbb{T}, \mathcal{X})$ is a nice tree-decomposition of G . By $\widehat{\mathcal{S}}_t$ we denote $\bigcup_{Z \subseteq X_t} \widehat{\mathcal{S}}_t[Z]$ (also called a *representative family of partial solutions*). We show how $\widehat{\mathcal{S}}_t$ is obtained by doing dynamic programming from base node to the root node.

Base node t . Here the graph H_t is empty and thus we take $\widehat{\mathcal{S}}_t = \{\emptyset\}$.

Introduce node t with child t' . Here, we know that $X_t \supset X_{t'}$ and $|X_t| = |X_{t'}| + 1$. Let v be the vertex in $X_t \setminus X_{t'}$. Furthermore observe that $E(H_t) = E(H_{t'})$ and v is degree zero vertex in H_t . Thus the graph H_t only differs from $H_{t'}$ at a isolated vertex v . Since we have not added any edge to the new graph, the family of solutions, which contains edge-subsets, does not change. Thus, we take $\widehat{\mathcal{S}}_t = \widehat{\mathcal{S}}_{t'}$. Formally, we take $\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}_{t'}[Z \setminus \{v\}]$. Since, H_t and $H_{t'}$ have same set of edges the invariant is vacuously maintained.

Forget node t with child t' . Here we know $X_t \subset X_{t'}$ and $|X_t| = |X_{t'}| - 1$. Let v be the vertex in $X_{t'} \setminus X_t$. Let $\mathcal{E}_v[Z]$ denote the set of edges between v and the vertices in $Z \subseteq X_t$. Observe that $E(H_t) = E(H_{t'}) \cup \mathcal{E}_v[X_t]$. Before we define things formally, observe that in this step the graphs H_t and $H_{t'}$ differ by at most **tw** edges - the edges with one endpoint in v and the other in X_t . We go through every possible way an optimal solution can intersect with these newly added edges. Let $\mathcal{P}_v[Z] = \{Y \mid \emptyset \neq Y \subseteq \mathcal{E}_v[Z]\}$. Then the new set of partial solutions is defined as follows.

$$\widehat{\mathcal{S}}_t[Z] = \begin{cases} \left(\widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] \diamond \mathcal{P}_v[Z] \right) \cup \left\{ A \in \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] : A \in \mathcal{S}_t \right\} & \text{if } v \in T \\ \left(\widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] \diamond \mathcal{P}_v[Z] \right) \cup \left\{ A \in \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] : A \in \mathcal{S}_t \right\} \cup \widehat{\mathcal{S}}_{t'}[Z] & \text{if } v \notin T \end{cases}$$

Now we claim that $\widehat{\mathcal{S}}_t[Z] \subseteq \mathcal{S}_t$. Towards the proof we first show that $\widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] \diamond \mathcal{P}_v[Z] \subseteq \mathcal{S}_t$. Let $E' \in \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] \diamond \mathcal{P}_v[Z]$. Note that $E' \cap \mathcal{E}_v[Z] \neq \emptyset$. If E' is a solution tree then $E' \in \mathcal{S}_t$ and we are done. Since $E' \setminus \mathcal{E}_v[Z] \in \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] \subseteq \mathcal{S}_{t'}$, every vertex of $(T \cap V(G_t)) \setminus (X_t \cup \{v\})$ is incident with some edge from E' . Since $E' \cap \mathcal{E}_v[Z] \neq \emptyset$, there exists an edge in E' which is incident to v . This implies that every vertex of $(T \cap V(G_t)) \setminus X_t$ is incident with some edge from E' . Now consider any connected component C in $G[E']$. If $v \notin V(C)$, then C contains a vertex from $X_{t'} \setminus \{v\} = X_t$, because $E' \setminus \mathcal{E}_v[Z] \in \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] \subseteq \mathcal{S}_{t'}$. If $v \in V(C)$, then C contains a vertex from X_t because $E' \cap \mathcal{E}_v[Z] \neq \emptyset$. Thus we have shown that $E' \in \mathcal{S}_t$. It is

easy to see that $\{A \in \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] : A \in \mathcal{S}_t\} \subseteq \mathcal{S}_t$. If $v \notin T$ then $\widehat{\mathcal{S}}_{t'}[Z] \subseteq \mathcal{S}_t$, because $\widehat{\mathcal{S}}_{t'}[Z] \subseteq \mathcal{S}_{t'}$ and $X_t = X_{t'} \setminus \{v\}$.

Now we show that $\widehat{\mathcal{S}}_t$ maintains the invariant of the algorithm. Let $L \in \mathcal{S}$.

1. Let $L_t = E(H_t) \cap L$ and $L_R = L \setminus L_t$. Furthermore, edges of L_t can be partitioned into $L_{t'} = E(H_{t'}) \cap L$ and $L_v = L_t \setminus L_{t'}$. That is, $L_t = L_{t'} \uplus L_v$.
2. Let $Z = \partial^t(L)$ and $Z' = \partial^{t'}(L)$.

By the property of $\widehat{\mathcal{S}}_{t'}$, there exists a $\widehat{L}_{t'} \in \widehat{\mathcal{S}}_{t'}[Z']$ such that

$$\begin{aligned} L \in \mathcal{S} &\iff L_{t'} \uplus L_v \uplus L_R \in \mathcal{S} \\ &\iff \widehat{L}_{t'} \uplus L_v \uplus L_R \in \mathcal{S} \end{aligned} \tag{17.1}$$

and $\partial^{t'}(L) = \partial^{t'}(\widehat{L}_{t'} \uplus L_v \uplus L_R) = Z'$.

We put $\widehat{L}_t = \widehat{L}_{t'} \cup L_v$ and $\widehat{L} = \widehat{L}_t \cup L_R$. We now show that $\widehat{L}_t \in \widehat{\mathcal{S}}_t[Z]$. If $v \notin Z'$, then $v \notin T$, $\widehat{L}_t = \widehat{L}_{t'}$ and $Z = Z'$. This implies that $\widehat{L}_t \in \widehat{\mathcal{S}}_t[Z]$. If $v \in Z'$ and $L_v \neq \emptyset$ then $Z' = Z \cup \{v\}$. This implies that $\widehat{L}_t \in \widehat{\mathcal{S}}_{t'}[Z'] \diamond \{L_v\} \subseteq \widehat{\mathcal{S}}_t[Z]$. If $v \in Z'$ and $L_v = \emptyset$ then $Z' = Z \cup \{v\}$ and $\widehat{L}_t = \widehat{L}_{t'}$. This implies that $\widehat{L}_t \in \{A \in \widehat{\mathcal{S}}_{t'}[Z'] : A \in \mathcal{S}_t\} \subseteq \widehat{\mathcal{S}}_t[Z]$. By (17.1), $\widehat{L} \in \mathcal{S}$. Finally, we need to show that $\partial^t(\widehat{L}) = Z$. Towards this just note that $\partial^t(\widehat{L}) = Z' \setminus \{v\} = Z$. This concludes the proof for the fact that $\widehat{\mathcal{S}}_t$ maintains the correctness invariant.

Join node t with two children t_1 and t_2 . Here, we know that $X_t = X_{t_1} = X_{t_2}$. Also we know that the edges of H_t is obtained by the union of edges of H_{t_1} and H_{t_2} which are disjoint. Of course they are separated by the vertices in X_t . A natural way to obtain a family of partial solutions for H_t is that we take the union of edges subsets of the families stored at nodes t_1 and t_2 . This is exactly what we do. Let

$$\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}_{t_1}[Z] \diamond \widehat{\mathcal{S}}_{t_2}[Z].$$

Now we show that $\widehat{\mathcal{S}}_t$ maintains the invariant. Let $L \in \mathcal{S}$.

1. Let $L_t = E(H_t) \cap L$ and $L_R = L \setminus L_t$. Furthermore edges of L_t can be partitioned into those belonging to H_{t_1} and those belonging to H_{t_2} . Let $L_{t_1} = E(H_{t_1}) \cap L$ and $L_{t_2} = E(H_{t_2}) \cap L$. Observe that since $E(H_{t_1}) \cap E(H_{t_2}) = \emptyset$, we have that $L_{t_1} \cap L_{t_2} = \emptyset$. Also observe that $L_t = L_{t_1} \uplus L_{t_2}$ and $G[L_{t_1}], G[L_{t_2}]$ form forests.
2. Let $Z = \partial^t(L)$. Since $X_t = X_{t_1} = X_{t_2}$ this implies that $Z = \partial^t(L) = \partial^{t_1}(L) = \partial^{t_2}(L)$.

Now observe that

$$\begin{aligned}
L \in \mathcal{S} &\iff L_{t_1} \uplus L_{t_2} \uplus L_R \in \mathcal{S} \\
&\iff \widehat{L}_{t_1} \uplus L_{t_2} \uplus L_R \in \mathcal{S} \quad (\text{by the property of } \widehat{\mathcal{S}}_{t_1} \text{ we have } \widehat{L}_{t_1} \in \widehat{\mathcal{S}}_{t_1}[Z]) \\
&\iff \widehat{L}_{t_1} \uplus \widehat{L}_{t_2} \uplus L_R \in \mathcal{S} \quad (\text{by the property of } \widehat{\mathcal{S}}_{t_2} \text{ we have } \widehat{L}_{t_2} \in \widehat{\mathcal{S}}_{t_2}[Z])
\end{aligned}$$

We put $\widehat{L}_t = \widehat{L}_{t_1} \cup \widehat{L}_{t_2}$. By the definition of $\widehat{\mathcal{S}}_t[Z]$, we have that $\widehat{L}_{t_1} \cup \widehat{L}_{t_2} \in \widehat{\mathcal{S}}_t[Z]$. The above inequalities also show that $\widehat{L} = \widehat{L}_t \cup L_R \in \mathcal{S}$. It remains to show that $\partial^t(\widehat{L}) = Z$. Since $\partial^{t_1}(L) = Z$, we have that $\partial^{t_1}(\widehat{L}_{t_1} \uplus L_{t_2} \uplus L_R) = Z$. Now since $X_{t_1} = X_{t_2}$ we have that $\partial^{t_2}(\widehat{L}_{t_1} \uplus L_{t_2} \uplus L_R) = Z$ and thus $\partial^{t_2}(\widehat{L}_{t_1} \uplus \widehat{L}_{t_2} \uplus L_R) = Z$. Finally, because $X_{t_2} = X_t$, we conclude that $\partial^t(\widehat{L}_{t_1} \uplus \widehat{L}_{t_2} \uplus L_R) = \partial^t(\widehat{L}) = Z$. This concludes the proof of correctness invariant.

Root node r . Here, $X_r = \emptyset$. We go through all the solution in $\widehat{\mathcal{S}}_r[\emptyset]$ and output the one with the minimum weight. This concludes the description of the dynamic programming algorithm.

Computation of $\widehat{\mathcal{S}}_t$. Now we show how to implement the algorithm described above in the desired running time by making use of Lemma 17.1. For our discussion

let us fix a node t and $Z \subseteq X_t$ of size k . While doing dynamic programming algorithm from the base nodes to the root node we always maintain the size invariant.

Base node t . Trivially, in this case we have maintained size invariant.

Introduce node t with child t' . Here, we have that $\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}_{t'}[Z \setminus \{v\}]$ and thus the number of partial solutions with i connected components in $\widehat{\mathcal{S}}_t[Z]$ is bounded $\binom{k}{i}$.

Forget node t with child t' . In this case,

$$\widehat{\mathcal{S}}_t[Z] = \begin{cases} \left(\widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] \diamond \mathcal{P}_v[Z] \right) \cup \left\{ A \in \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] : A \in \mathcal{S}_t \right\} & \text{if } v \in T \\ \left(\widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] \diamond \mathcal{P}_v[Z] \right) \cup \left\{ A \in \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] : A \in \mathcal{S}_t \right\} \cup \widehat{\mathcal{S}}_{t'}[Z] & \text{if } v \notin T \end{cases}$$

Since $\widehat{\mathcal{S}}_{t'}[Z \cup \{v\}]$ maintains size invariant, the number of edge subsets with i connected components in $\widehat{\mathcal{S}}_{t'}[Z \cup \{v\}]$ is upper bounded by $\binom{k+1}{i}$. It is easy to see that the number of edge subsets with i connected components in $\mathcal{P}_v[Z]$ is upper bounded by $\binom{k}{i}$. So first we apply Lemma 17.1 and obtain $\mathcal{R} \subseteq \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] \diamond \mathcal{P}_v[Z]$ that maintains the correctness and size invariants. Now let,

$$\widehat{\mathcal{S}}'_t[Z] = \begin{cases} \mathcal{R} \cup \left\{ A \in \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] : A \in \mathcal{S}_t \right\} & \text{if } v \in T \\ \mathcal{R} \cup \left\{ A \in \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] : A \in \mathcal{S}_t \right\} \cup \widehat{\mathcal{S}}_{t'}[Z] & \text{if } v \notin T \end{cases}$$

Note that $\widehat{\mathcal{S}}'_t[Z]$ maintains correctness invariant. Since the number of edge subsets with i connected components in $\{A \in \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}] : A \in \mathcal{S}_t\}$ and $\widehat{\mathcal{S}}_{t'}[Z]$ is bounded by $\binom{k+1}{i}$, the the number of edge subsets with i connected components in $\widehat{\mathcal{S}}'_t[Z]$ is at most $\binom{k+4}{i}$. Also note that $\widehat{\mathcal{S}}'_t[Z] = \widehat{\mathcal{S}}'_t[Z] \diamond \{\emptyset\}$. Thus we can apply Lemma 17.1 and obtain $\widehat{\mathcal{S}}''_t[Z] \subseteq \widehat{\mathcal{S}}'_t[Z]$ that maintains the correctness and size invariants. We update $\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}''_t[Z]$.

The running time to compute $\{A \in \widehat{\mathcal{S}}_t[Z \cup \{v\}] : A \in \mathcal{S}_t\}$ is $\mathcal{O}(2^{|Z|}n)$. Thus the running time T to compute $\widehat{\mathcal{S}}_t$ (that is, across all subsets of X_t) is

$$\begin{aligned} T &= \mathcal{O}\left(\sum_{i=1}^{\mathbf{tw}+1} \binom{\mathbf{tw}+1}{i} \left(i^\omega (2^\omega + 2)^i n + i^\omega 2^{i(\omega-1)} 3^i n\right) + \sum_{i=1}^{\mathbf{tw}+1} \binom{\mathbf{tw}+1}{i} 2^i n\right) \\ &= \mathcal{O}\left(\mathbf{tw}^\omega n (2^\omega + 3)^{\mathbf{tw}} + \mathbf{tw}^\omega n (1 + 2^{\omega-1} \cdot 3)^{\mathbf{tw}}\right) \end{aligned}$$

Join node t with two children t_1 and t_2 . Here we defined

$$\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}_{t_1}[Z] \diamond \widehat{\mathcal{S}}_{t_2}[Z].$$

The number of edge subsets with i connected components in $\widehat{\mathcal{S}}_{t_1}[Z]$ and $\widehat{\mathcal{S}}_{t_2}[Z]$ are bounded by $\binom{k}{i}$. Now, we apply Lemma 17.1 and obtain $\widehat{\mathcal{S}}'_t[Z]$ that maintains the correctness invariant and has size at most 2^k . We put $\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}'_t[Z]$. The running time to compute $\widehat{\mathcal{S}}_t$ is

$$\mathcal{O}\left(\mathbf{tw}^\omega n (2^\omega + 3)^{\mathbf{tw}} + \mathbf{tw}^\omega n (1 + 2^{\omega-1} \cdot 3)^{\mathbf{tw}}\right).$$

Thus the whole algorithm takes $\mathcal{O}\left(\mathbf{tw}^\omega n^2 (2^\omega + 3)^{\mathbf{tw}} + \mathbf{tw}^\omega n^2 (1 + 2^{\omega-1} \cdot 3)^{\mathbf{tw}}\right) = \mathcal{O}(8.7703^{\mathbf{tw}} n^2)$ time as the number of nodes in a nice tree-decomposition is upper bounded by $\mathcal{O}(n)$. However, observe that we do not need to compute the forests and the associated weight at every step of the algorithm. The size of the forest is at most $\mathbf{tw} + 1$ and we can maintain these forests across the bags during dynamic programming in time $\mathbf{tw}^{\mathcal{O}(1)}$. Also, these forests can be used to compute the set $\{A \in \widehat{\mathcal{S}}_t[Z \cup \{v\}] : A \in \mathcal{S}_t\}$ during the computation in the forget node t . This will lead to an algorithm with the claimed running time. This completes the proof. \square

17.2 FEEDBACK VERTEX SET

In this subsection we study the FEEDBACK VERTEX SET problem which is defined as follows.

FEEDBACK VERTEX SET Input: An undirected graph G together with a tree-decomposition $(\mathbb{T}, \mathcal{X})$ of width \mathbf{tw} and a non negative weight function $w : V(G) \rightarrow \mathbb{N}$. Task: Find a minimum weight set $Y \subseteq V(G)$ such that $G[V(G) \setminus Y]$ is a forest.	Parameter: \mathbf{tw}
--	---------------------------------

Let G be an input graph of the FEEDBACK VERTEX SET problem. In this subsection instead of saying feedback vertex set $Y \subseteq V(G)$ is a solution, we say that $V(G) \setminus Y$ is a solution, i.e, our objective is to find a maximum weight set $V' \subseteq V(G)$ such that $G[V']$ is a forest. We call $V' \subseteq V(G)$ is an *optimal solution* if V' is a solution with maximum weight. Let \mathcal{S} be a family of vertex subsets such that every vertex subset corresponds to an optimal solution. That is,

$$\mathcal{S} = \{V' \subseteq V(G) \mid V' \text{ is an optimal solution}\}.$$

Let $(\mathbb{T}, \mathcal{X})$ be a tree decomposition of G of width \mathbf{tw} . For each tree node t and $Z \subseteq X_t$, we define $\mathcal{S}_t[Z]$, *family of partial solutions* as follows.

$$\mathcal{S}_t[Z] = \{U \subseteq V(H_t) \mid U \cap X_t = Z \text{ and } H_t[U] \text{ is a forest}\}$$

We denote by K^t a complete graph on the vertex set X_t . Let G^* be subgraph of G . Let C'_1, \dots, C'_ℓ be the connected components of G^* that have nonempty intersection with X_t . Let $C_i = C'_i \cap X_t$. By $F_t(G^*)$ we denote the a forest $\{Q_1, \dots, Q_\ell\}$ where each Q_i is an arbitrary spanning tree of $K^t[C_i]$.

For two family of vertex subsets \mathcal{P} and \mathcal{Q} of the subgraph H_t , we denote

$$\mathcal{P} \otimes_t \mathcal{Q} = \{U_1 \cup U_2 \mid U_1 \in \mathcal{P}, U_2 \in \mathcal{Q} \text{ and } H_t[U_1 \cup U_2] \text{ is a forest}\}.$$

With every node t of \mathbb{T} , we associate the subgraph H_t of G . For every node t , we keep a family of partial solutions for the graph H_t which sufficient to guarantee the correctness of the algorithm. That is for every optimal solution $L \in \mathcal{S}$ with $L \cap X_t = Z$ and its intersection $L_t = V(H_t) \cap L$ with the graph H_t , we have some partial solution \hat{L}_t in our subset such that $\hat{L}_t \cap X_t = Z$ and $\hat{L}_t \cup L_R$ is an optimal solution, i.e $G[\hat{L}_t \cup L_R]$ is a forest, where $L_R = L \setminus L_t$ and $w(\hat{L}_t \cup L_R) \geq w(L)$. Now we are ready to state the main theorem.

Theorem 17.2. *Let G be an n -vertex graph given together with its tree decomposition of width \mathbf{tw} . Then FEEDBACK VERTEX SET on G can be solved in time*

$$\mathcal{O}\left((1 + 2^{\omega-1} \cdot 3)^{\mathbf{tw}} \mathbf{tw}^{\mathcal{O}(1)n}\right).$$

Proof. For every node t of \mathbb{T} and $Z \subseteq X_t$, we store a family of vertex subsets $\widehat{\mathcal{S}}_t[Z]$ of $V(H_t)$ satisfying the following correctness invariant.

Correctness Invariant: For every $L \in \mathcal{S}$ we have the following. Let $L_t = V(H_t) \cap L$, $L_R = L \setminus L_t$ and $L \cap X_t = Z$. Then there exists $\hat{L}_t \in \widehat{\mathcal{S}}_t[Z]$ such that $\hat{L} = \hat{L}_t \cup L_R$ is an optimal solution, i.e $G[\hat{L}_t \cup L_R]$ is a forest with $w(\hat{L}_t) \geq w(L_t)$. Thus we have that $\hat{L} \in \mathcal{S}$.

We process the nodes of the tree \mathbb{T} from base nodes to the root node while doing the dynamic programming. Throughout the process we maintain the correctness invariant, which will prove the correctness of the algorithm. However, our main idea is to use representative sets to obtain $\widehat{\mathcal{S}}_t[Z]$ of small size. That is, given the set $\widehat{\mathcal{S}}_t[Z]$ that satisfies the correctness invariant, we use *representative set* tool to obtain a subset $\widehat{\mathcal{S}}'_t[Z]$ of $\widehat{\mathcal{S}}_t[Z]$ that also satisfies the correctness invariant and has size upper bounded by $2^{|Z|}$ in total. More precisely, the number of partial solutions in $\widehat{\mathcal{S}}'_t[Z]$ that have i connected components with nonempty intersection with X_t is upper bounded by $\binom{|Z|}{i}$. Thus, we maintain the following size invariant.

Size Invariant: After node t of \mathbb{T} is processed by the algorithm, we have that $|\widehat{\mathcal{S}}'_t[Z, i]| \leq \binom{|Z|}{i}$, where $\widehat{\mathcal{S}}'_t[Z, i]$ is the set of partial solutions that have i connected components with nonempty intersection with X_t .

Lemma 17.2 (Product Shrinking Lemma). *Let t be a node of \mathbb{T} and let $Z \subseteq X_t$ be a set of size k . Let \mathcal{P} and \mathcal{Q} be two family of vertex subsets of $V(H_t)$ (partial solutions) such that for any $A \in \mathcal{P}$ and $B \in \mathcal{Q}$, $E(H_t[A]) \cap E(H_t[B]) = \emptyset$. Furthermore, let $\widehat{\mathcal{S}}_t[Z] = \mathcal{P} \otimes_t \mathcal{Q}$ be the family of vertex subsets of $V(H_t)$ satisfying the correctness invariant. If the number of partial solutions with i connected components having nonempty intersection with Z in \mathcal{P} as well as in \mathcal{Q} is bounded by $\binom{k+c}{i}$ where c is*

some fixed constant, then in time $\mathcal{O}\left(k^\omega (2^\omega + 2)^k n + k^\omega 2^{k(\omega-1)} 3^k n\right)$ we can compute $\widehat{\mathcal{S}}'_t[Z] \subseteq \widehat{\mathcal{S}}_t[Z]$ satisfying correctness and size invariants.

Proof. We start by associating a matroid with node t and the set $Z \subseteq X_t$ as follows. We consider a graphic matroid $M = (E, \mathcal{I})$ on $K^t[Z]$. Here, the element set E of the matroid is the edge set $E(K^t[Z])$ and the family of independent sets \mathcal{I} consists of spanning forests of $K^t[Z]$. Here our objective is to find a small subfamily of $\widehat{\mathcal{S}}_t[Z] = \mathcal{P} \otimes_t \mathcal{Q}$ satisfying correctness and size invariants using efficient computation of representative family in the graphic matroid M . The main idea to prune the size of partial solutions is as follows: for each independent set $U \in \widehat{\mathcal{S}}_t[Z]$ we associate $F_t(H_t[U])$ as the corresponding independent set in the graphic matroid M and compute representative family in the graphic matroid M .

Let $\mathcal{P} = \{A_1, \dots, A_\ell\}$ and $\mathcal{Q} = \{B_1, \dots, B_{\ell'}\}$. Let $\mathcal{L}_1 = \{F_t(H_t[A_1]), \dots, F_t(H_t[A_\ell])\}$ and $\mathcal{L}_2 = \{F_t(H_t[B_1]), \dots, F_t(H_t[B_{\ell'}])\}$ be the set of forests in $K^t[Z]$ corresponding to the vertex subsets in \mathcal{P} and \mathcal{Q} respectively. Now we define a non negative weight function $w' : \mathcal{L}_1 \bullet \mathcal{L}_2 \rightarrow \mathbb{N}$ as follows. For each $F_t(H_t[A_i]) \cup F_t(H_t[B_j]) \in \mathcal{L}_1 \bullet \mathcal{L}_2$ we set $w'(F_t(H_t[A_i]) \cup F_t(H_t[B_j])) = w(A_i \cup B_j)$. For $i \in [k]$ and $r \in \{1, 2\}$, let $\mathcal{L}_{r,i}$ be the family of forests of \mathcal{L}_r with i edges. Now we apply Corollary 16.1 and find $\widehat{\mathcal{L}}_{1,i} \bullet \widehat{\mathcal{L}}_{2,j} \subseteq_{\maxrep}^{k-1-i-j} \mathcal{L}_{1,i} \bullet \mathcal{L}_{2,j}$ of size $\binom{k-1}{i+j}$ for all $i, j \in [k]$. Let $\widehat{\mathcal{S}}'_t[Z, k-d] \subseteq \widehat{\mathcal{S}}_t[Z, k-d]$ be such that for every $U_1 \cup U_2 \in \widehat{\mathcal{S}}'_t[Z, k-d]$ we have that $F_t(H_t[U_1]) \cup F_t(H_t[U_2]) \in \widehat{\mathcal{L}}_{1,i} \bullet \widehat{\mathcal{L}}_{2,j}$. Let $\widehat{\mathcal{S}}'_t[Z] = \bigcup_{j=0}^k \widehat{\mathcal{S}}'_t[Z, j]$. By Corollary 16.1, $|\widehat{\mathcal{S}}'_t[Z, k-d]| \leq k \binom{k-1}{d} \leq \binom{k}{k-d}$, and hence $\widehat{\mathcal{S}}'_t[Z]$ maintains the size invariant.

Now we show that the $\widehat{\mathcal{S}}'_t[Z]$ maintains the correctness invariant. Let $L \in \mathcal{S}$ and let $L_t = V(H_t) \cap L$, $L_R = L \setminus L_t$ and $Z = L \cap X_t$. Since $\widehat{\mathcal{S}}_t[Z]$ satisfy correctness invariant, there exists $\widehat{L}_t \in \widehat{\mathcal{S}}_t[Z]$ such that $w(\widehat{L}_t) \geq w(L_t)$, $\widehat{L} = \widehat{L}_t \cup L_R$ is an optimal solution and $\widehat{L} \cap X_t = Z$. Since $\widehat{\mathcal{S}}_t[Z] = \mathcal{P} \otimes_t \mathcal{Q}$, there exists $U_1 \in \mathcal{P}$ and $U_2 \in \mathcal{Q}$ such that $\widehat{L}_t = U_1 \cup U_2$. Observe that $H_t[U_1 \cup U_2]$ form a forest. Consider the forests $F_t(H_t[U_1])$ and $F_t(H_t[U_2])$. Suppose $|F_t(H_t[U_1])| = i_1$ and $|F_t(H_t[U_2])| = i_2$, then $F_t(H_t[U_1]) \cup$

$F_t(H_t[U_2]) \in \mathcal{L}_{1,i_1} \bullet \mathcal{L}_{1,i_2}$. This is because, if $F_t(H_t[U_1]) \cup F_t(H_t[U_2])$ contains a cycle, then corresponding to that cycle we can get a cycle in $H_t[U_1 \cup U_2]$, which is a contradiction. Now let $E' = F_t(G[L_R \cup Z])$ be the forest corresponding to $L_R \cup Z$ with respect to the bag X_t . Since \hat{L} is a solution, we have that $F_t(H_t[U_1]) \cup F_t(H_t[U_2]) \cup E'$ is a forest in $K^t[Z]$. Since $\widehat{\mathcal{L}_{1,i_1} \bullet \mathcal{L}_{2,i_2}} \subseteq_{maxrep}^{k-1-i_1-i_2} \mathcal{L}_{1,i_1} \bullet \mathcal{L}_{2,i_2}$, there exists a forest $F_t(H_t[U'_1]) \cup F_t(H_t[U'_2]) \in \widehat{\mathcal{L}_{1,i_1} \bullet \mathcal{L}_{2,i_2}}$ such that $w'(F_t(H_t[U'_1]) \cup F_t(H_t[U'_2])) \geq w'(F_t(H_t[U_1]) \cup F_t(H_t[U_2])) = w(U_1 \cup U_2)$ and $F_t(H_t[U'_1]) \cup F_t(H_t[U'_2]) \cup E'$ is a forest in $K^t[Z]$. Hence $U'_1 \cup U'_2 \in \widehat{\mathcal{S}}'_t[Z]$. Since $w(U'_1 \cup U'_2) = w'(F_t(H_t[U'_1]) \cup F_t(H_t[U'_2]))$, $w(U'_1 \cup U'_2) \geq w(U_1 \cup U_2)$. Thus, we can conclude that $U'_1 \cup U'_2 \cup L_R$ is an optimal solution. This proves that $\widehat{\mathcal{S}}'_t[Z]$ maintains the correctness invariant.

By Corollary 16.1, the running time to compute $\widehat{\mathcal{S}}'_t[Z]$ is upper bounded by,

$$\mathcal{O}\left(k^\omega (2^\omega + 2)^k n + k^\omega 2^{k(\omega-1)} 3^k n\right).$$

□

We now explain the dynamic programming algorithm over the tree-decomposition $(\mathbb{T}, \mathcal{X})$ of G and prove that it maintains the correctness invariant. We assume that $(\mathbb{T}, \mathcal{X})$ is a nice tree-decomposition of G . By $\widehat{\mathcal{S}}_t$ we denote $\bigcup_{Z \subseteq X_t} \widehat{\mathcal{S}}'_t[Z]$ (also called a *representative family of partial solutions*). We show how $\widehat{\mathcal{S}}_t$ is obtained by doing dynamic programming from base node to the root node.

Base node t . Here the graph H_t is empty and thus we take $\widehat{\mathcal{S}}_t = \{\emptyset\}$.

Introduce node t with child t' . Here, we know that $X_t \supset X_{t'}$ and $|X_t| = |X_{t'}| + 1$. Let v be the vertex in $X_t \setminus X_{t'}$. Furthermore observe that $E(H_t) = E(H_{t'})$ and v is degree zero vertex in H_t . Thus the graph H_t only differs from $H_{t'}$ at a isolated vertex v . Since we have not added any edge to the new graph, the family of solutions does not change. Thus, we take $\widehat{\mathcal{S}}_t = \widehat{\mathcal{S}}_{t'}$. Formally, we take

$\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}_{t'}[Z \setminus \{v\}]$. Since, H_t and $H_{t'}$ have same set of edges both the correctness and size invariant is maintained.

Forget node t with child t' . Here we know $X_t \subset X_{t'}$, $|X_t| = |X_{t'}| - 1$ Let $v \in X_{t'} \setminus X_t$. Observe that $E(H_t) \supseteq E(H_{t'})$. Thus for any $U \in \widehat{\mathcal{S}}_{t'}$, $H_t[U]$ may or may not be a forest. So in this case we collect all the vertex subsets in $\widehat{\mathcal{S}}_{t'}$ which is a forest as induced subgraph in H_t . Formally,

$$\widehat{\mathcal{S}}_t[Z] = \left\{ A \in \widehat{\mathcal{S}}_{t'}[Z] \cup \widehat{\mathcal{S}}_{t'}[Z \cup v] \mid H_t[A] \text{ is a forest} \right\}.$$

Let $\widehat{\mathcal{S}}_t = \bigcup_{Z \subseteq X_t} \widehat{\mathcal{S}}_t[Z]$. Now we show that $\widehat{\mathcal{S}}_t$ satisfies correctness invariant. Let $L \in \mathcal{S}$. Let $L_{t'} = V(H_{t'}) \cap L$ and $L_R = L \setminus L_{t'}$. Let $Z' = L \cap X_{t'}$ Now observe that

$$\begin{aligned} L \in \mathcal{S} &\iff L_{t'} \cup L_R \in \mathcal{S} \\ &\iff \widehat{L}_{t'} \cup L_R \in \mathcal{S} \quad (\text{by the property of } \widehat{\mathcal{S}}_{t'} \text{ we have that } \widehat{L}_{t'} \in \widehat{\mathcal{S}}_{t'}[Z']) \end{aligned}$$

Since $H_t[\widehat{L}_{t'}]$ is a forest, $\widehat{L}_{t'} \in \widehat{\mathcal{S}}_t[Z' \setminus \{v\}]$. This concludes the proof of correctness invariant.

Since $\widehat{\mathcal{S}}_t[Z] \subseteq \widehat{\mathcal{S}}_{t'}[Z] \cup \widehat{\mathcal{S}}_{t'}[Z \cup v]$, the number of partial solutions with i connected components having nonempty intersection with Z in $\widehat{\mathcal{S}}_t[Z]$ is bounded by $\binom{k}{i} + \binom{k+1}{i} \leq \binom{k+2}{i}$. Since $\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}_t[Z] \otimes_t \{\emptyset\}$, we apply Lemma 17.2 and find $\widehat{\mathcal{S}}'_t[Z] \subseteq \widehat{\mathcal{S}}_t[Z]$ satisfies correctness and size invariant in time $\mathcal{O}\left(k^\omega (2^\omega + 2)^k n + k^\omega 2^{k(\omega-1)} 3^k n\right)$ and we set $\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}'_t[Z]$.

Join node t with two children t_1 and t_2 . Here, we know that $X_t = X_{t_1} = X_{t_2}$. The natural way to get a family of partial solutions for X_t is the union of vertex sets of two families stored at node t_1 and t_2 which form a forest as an induced subgraph

of H_t , i.e,

$$\begin{aligned}\widehat{\mathcal{S}}_t[Z] &= \{U_1 \cup U_2 \mid U_1 \in \widehat{\mathcal{S}}_{t_1}[Z], U_2 \in \widehat{\mathcal{S}}_{t_2}[Z], H_t[U_1 \cup U_2] \text{ is a forest}\} \\ &= \widehat{\mathcal{S}}_{t_1}[Z] \otimes_t \widehat{\mathcal{S}}_{t_2}[Z]\end{aligned}$$

Now we show that $\widehat{\mathcal{S}}_t$ maintains the invariant. Let $L \in \mathcal{S}$. Let $L_t = V(G_t) \cap L$, $L_{t_1} = V(G_{t_1}) \cap L$, $L_{t_2} = V(G_{t_2}) \cap L$ and $L_R = L \setminus L_t$. Let $Z = L \cap X_t$. Now observe that

$$\begin{aligned}L \in \mathcal{S} &\iff L_{t_1} \cup L_{t_2} \cup L_R \in \mathcal{S} \\ &\iff \widehat{L}_{t_1} \cup L_{t_2} \cup L_R \in \mathcal{S} \quad (\text{by the property of } \widehat{\mathcal{S}}_{t_1} \text{ we have } \widehat{L}_{t_1} \in \widehat{\mathcal{S}}_{t_1}[Z]) \\ &\iff \widehat{L}_{t_1} \cup \widehat{L}_{t_2} \cup L_R \in \mathcal{S} \quad (\text{by the property of } \widehat{\mathcal{S}}_{t_2} \text{ we have } \widehat{L}_{t_2} \in \widehat{\mathcal{S}}_{t_2}[Z])\end{aligned}$$

We put $\widehat{L}_t = \widehat{L}_{t_1} \cup \widehat{L}_{t_2}$. By the definition of $\widehat{\mathcal{S}}_t[Z]$, we have that $\widehat{L}_t \cup L_R \in \widehat{\mathcal{S}}_t[Z]$. The above inequalities also show that $\widehat{L}_t \cup L_R \in \mathcal{S}$. Note that $(\widehat{L}_t \cup L_R) \cap X_t = Z$. This concludes the proof of correctness invariant.

We apply Lemma 17.2 and find $\widehat{\mathcal{S}}'_t[Z] \subseteq \widehat{\mathcal{S}}_t[Z]$ satisfies correctness and size invariant in time $\mathcal{O}\left(k^\omega (2^\omega + 2)^k n + k^\omega 2^{k(\omega-1)} 3^k n\right)$ and we set $\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}'_t[Z]$.

Root node r . Here, $X_r = \emptyset$. We go through all the solution in $\widehat{\mathcal{S}}_r[\emptyset]$ and output the one with the maximum weight.

In worst case, in every tree node t , for all subset $Z \subseteq X_t$, we apply Lemma 17.2. So by doing the same run time analysis as in the case of Steiner Tree, the total running time will be upper bounded by $\mathcal{O}\left(\left((2^\omega + 3)^{\text{tw}} + (1 + 2^{\omega-1} \cdot 3)^{\text{tw}}\right) \text{tw}^{O(1)} n\right)$. \square

Chapter 18

Matroidal Multilinear Monomial Detection

In this chapter we extend the MULTILINEAR MONOMIAL DETECTION problem to a matroidal version, where variables of a monomial should form an independent set of a matroid and design an algorithm for this. The problem MATROIDAL MULTILINEAR MONOMIAL DETECTION (k -WMMLD) is defined as follows.

k -WMMLD

Parameter: k

Input: An arithmetic circuit C over variables $X = \{x_1, x_2, \dots, x_n\}$ representing a polynomial $P(X)$ over \mathbb{Z} , a linear matroid $M = (E, \mathcal{I})$ where the ground set $E = X$ with its representation matrix A_M and an additive weight function $w : 2^X \rightarrow \mathbb{N}$.

Question: Does $P(X)$ construed as a sum of monomials contains a multilinear monomial Z of degree k such that $Z \in \mathcal{I}$? If yes find a minimum weighted such Z .

Our main theorem of this section is as follows. The proof of this theorem is along the lines of Theorem 12.1. The only difference is that we compute representative family with respect to the given matroid.

Theorem 18.1. k -WMMLD can be solved in time $\mathcal{O}(7.7703^k k^\omega s(C))$.

Proof. We outline a proof here. Let $\pi = v_1, \dots, v_n$ be a topological ordering of C such that all the nodes corresponding to variables appear before any other gate and for every directed arc uv we have that $u <_\pi v$. As in Theorem 12.1, at every node we keep a family $\mathcal{F}_{v_i}^j$ of j -multilinear term that are also members of \mathcal{I} , where

$j \in \{1, \dots, k\}$. Let $\mathcal{F}_{v_i} = \bigcup_{x=1}^k \mathcal{F}_{v_i}^x$. So $\mathcal{F}_v \subseteq \mathcal{I}$. We process the nodes from left to right and keep $\widehat{\mathcal{F}}_{v_i}^j \subseteq_{\minrep}^{k-j} \mathcal{F}_{v_i}^j$ of size $\binom{k}{p}$.

When v is an input node then the associated family contains only one set. That is, if v is labelled with x_i and $\{x_i\} \in \mathcal{I}$ then $\mathcal{F}_v = \{\{x_i\}\}$, otherwise $\mathcal{F}_v = \{\emptyset\}$. If v is labelled from \mathbb{Z}^+ then $\mathcal{F}_v = \{\emptyset\}$. When v is not an input node, then we have two cases.

Addition Gate. $v = v_1 + v_2$

Due to the left to right computation in the topological order, we have representative families \mathcal{F}_{v_1} and \mathcal{F}_{v_2} for v_1 and v_2 respectively, where the number of subsets with p elements in \mathcal{F}_{v_1} as well as in \mathcal{F}_{v_2} will be at most $\binom{k}{p}$. So the representative family corresponding to v will be the representative family of $\mathcal{F}_{v_1} \cup \mathcal{F}_{v_2}$. We partition $\mathcal{F}_{v_1} \cup \mathcal{F}_{v_2}$ based on the size of subsets in it. Let $\mathcal{F}_{v_1} \cup \mathcal{F}_{v_2} = \bigsqcup_{p \leq k} \mathcal{H}_p$, where \mathcal{H}_p contains all subsets of size p in $\mathcal{F}_{v_1} \cup \mathcal{F}_{v_2}$. Note that $|\mathcal{H}_p| \leq 2 \binom{k}{p}$. Now using Theorem 13.2 we can compute all $\widehat{\mathcal{H}}_p \subseteq_{\minrep}^{k-p} \mathcal{H}_p$ in time

$$\mathcal{O} \left(2 \sum_{p \leq k} \left\{ \binom{k}{p} \binom{k}{p} p^\omega + \binom{k}{p} \binom{k}{p}^{\omega-1} \right\} \right).$$

The above running time is upper bounded by $\mathcal{O}(4^k p^\omega k + 2^{\omega k} k)$. We output $\bigcup_{p \leq k} \widehat{\mathcal{H}}_p$ as the representative family corresponding to the node v . By Theorem 13.2, $|\widehat{\mathcal{H}}_p| \leq \binom{k}{p}$ and thus the number of subsets with p elements in $\bigcup_{p \leq k} \widehat{\mathcal{H}}_p$ is at most $\binom{k}{p}$.

Multiplication Gate. $v = v_1 \times v_2$

Similar to the previous case we have a representative families \mathcal{F}_{v_1} and \mathcal{F}_{v_2} for v_1 and v_2 respectively, where the number of subsets with p elements in \mathcal{F}_{v_1} as well as in \mathcal{F}_{v_2} , is at most $\binom{k}{p}$. Here, the representative family corresponding to v will be the representative family of $\mathcal{F}_{v_1} \bullet \mathcal{F}_{v_2}$. We have that

$$\mathcal{F}_{v_1} \bullet \mathcal{F}_{v_2} = \bigcup_{p_1, p_2} \mathcal{F}_{v_1}^{p_1} \bullet \mathcal{F}_{v_2}^{p_2},$$

where $\mathcal{F}_{v_i}^{p_i}$ contains all the subsets of size p_i in \mathcal{F}_{v_i} . We know that $|\mathcal{F}_{v_i}^{p_i}| \leq \binom{k}{p_i}$. Now by using Corollary 16.1, we can compute $\widehat{\mathcal{F}_{v_1}^{p_1} \bullet \mathcal{F}_{v_2}^{p_2}} \subseteq_{\minrep}^{k-p_1-p_2} \mathcal{F}_{v_1}^{p_1} \bullet \mathcal{F}_{v_2}^{p_2}$ of size $\binom{k}{p_1+p_2}$ for all p_1, p_2 together in time $\mathcal{O}\left(k^\omega (2^\omega + 2)^k + k^\omega 2^{k(\omega-1)} 3^k\right)$.

Now let $\mathcal{F} = \bigcup_{p_1, p_2} \widehat{\mathcal{F}_{v_1}^{p_1} \bullet \mathcal{F}_{v_2}^{p_2}} = \uplus_p \mathcal{H}_p$, where $\uplus_p \mathcal{H}_p$ is the partition of \mathcal{F} based on the size of subsets. It is easy to see that $|\mathcal{H}_p| \leq k \binom{k}{p}$. Now using Theorem 13.2 we can compute $\widehat{\mathcal{H}_p} \subseteq_{\minrep}^{k-p} \mathcal{H}_p$ for all $p \leq k$ together in time

$$\mathcal{O}\left(k \sum_{p \leq k} \left\{ \binom{k}{p} \binom{k}{p} p^\omega + \binom{k}{p} \binom{k}{p}^{\omega-1} \right\}\right)$$

The above running time is upper bounded by $\mathcal{O}(4^k k^{\omega+1} + 2^{\omega k} k^2)$. We output $\bigcup_{p \leq k} \widehat{\mathcal{H}_p}$ as the representative family corresponding to the node v .

Now we output a minimum weight set of size k among the representative family corresponding to the root node, otherwise we output NO. Since there are $s(C)$ nodes in C , the total running time is bounded by $\mathcal{O}\left(k^\omega (2^\omega + 2)^k s(C) + k^\omega 2^{k(\omega-1)} 3^k s(C)\right)$. This completes the proof. \square

Part IV

Matroid Girth and Connectivity

Chapter 19

Matroid Girth

Matroids are mathematical objects which have many applications in algorithms. Certain problems on matroids are known to be equivalent to fundamental combinatorial problems like `MINIMUM WEIGHT SPANNING TREE` or `PERFECT MATCHING`. Matroids are an exact characterization of structures on which a greedy algorithm produces an optimum solution. This motivates our study on several natural matroid problems. In this part of the thesis we study `MATROID GIRTH` and `MATROID CONNECTIVITY` problems.

One of the most fundamental problems in coding theory is the problem of computing the minimum distance of a linear code. The decision version of this problem for binary linear codes was conjectured to be NP-complete by Berlekamp, McEliece, and van Tilborg [12] in 1978 and it was only in 1997 that its NP-completeness was proved by Vardy [116]. In fact, the result of Vardy can be extended to linear codes over all finite fields. The `MINIMUM DISTANCE` problem is a special case of the `MATROID GIRTH` problem, where the objective is to compute the length of a shortest circuit in a given matroid. Also note that the notion of girth for graphic matroids coincides with the notion of girth defined in the context of graphs. It was known in the 80's that `MATROID GIRTH` is NP-complete [95]. Later Vardi showed that the problem is NP-complete even for binary matroids (matroids over \mathbb{F}_2) [116]. Therefore, `MATROID GIRTH` is a natural candidate for a study in the realm of parameterized complexity. We study `MATROID GIRTH` on various parameters.

MATROID GIRTH

Input: A linear matroid $M = (E, \mathcal{I})$ together with its representation matrix A_M of dimension $\text{rank}(M) \times |E|$ over a field \mathbb{F}_q , and a positive integer k .

Parameters: (1) k , (2) $\text{rank}(M)$ and (3) $\text{rank}(M)+q$

Question: Does there exist a circuit of size at most k in M ?

A first natural parameter for our problems is the solution size, k . We argue that this problem is unlikely to have an FPT algorithm in general. For this, we consider

the HALL SET problem. In this problem we are given a bipartite graph G with bipartition into A and B and a positive integer k , and the objective is to find a set $S \subseteq A$ of size at most k such that the number of neighbors of S in B is strictly smaller than $|S|$, that is, $|N(S)| < |S|$. It is known that HALL SET is W[1]-hard [63]. This problem is clearly a special case of MATROID GIRTH. Indeed, if the input to MATROID GIRTH is a transversal matroid then the problem is precisely HALL SET. We would also like to point out that EVEN SET, the parameterized version of the problem of computing the minimum distance of a binary linear code is a long standing open problem in the area and is stated among the most “infamous open problems” in the Research Horizons section of the recent textbook by Downey and Fellows [41, Chapter 33.1]. However, the *exact* version of EVEN SET, where we want to check for a circuit of size exactly k is known to be W[1]-hard [43]. These intractability results force us to look for alternate parameterizations.

The next natural parameter would be the rank of the input matroid. Since it is a larger parameter than the solution size k , one might hope for tractability results in place of previous intractability results. But the NP-hardness reduction by Khachiyan et al [73] for the problem LINEAR DEGENERACY, a special case of MATROID GIRTH, also gives W[1]-hardness for MATROID GIRTH when parameterized by the rank of the input matroid.

LINEAR DEGENERACY

Input: A $k \times m$ matrix M of rank k .

Question: Is there a set of k columns in M which are linearly dependent

Khachiyan et al [73] showed that LINEAR DEGENERACY is NP-hard, by giving a reduction from SMALL SUBSET SUM, which is defined as follows.

SMALL SUBSET SUM

Input: A set of n positive integers $S = \{\alpha_1, \dots, \alpha_n\}$ and $k, \beta \in \mathbb{N}^+$.

Question: Does there exist a subset of k integers in S which sum up to β ?

The reduction to prove LINEAR DEGENERACY is NP-hard [73, Theorem 1] takes as an input (S, k, β) , an instance of SMALL SUBSET SUM and produces an instance $(M, k + 3)$ to LINEAR DEGENERACY, where M is a $(k + 3) \times (|S| + 2)$ matrix. It is known that SMALL SUBSET SUM parameterized by k is W[1]-hard [41]. This implies that LINEAR DEGENERACY parameterized by k is W[1]-hard. Thus we can conclude that MATROID GIRTH is W[1]-hard.

Therefore, we choose as our parameter, $\text{rank}(M) + q$, where q is the size of the field in which the matroid is represented. Indeed, q is constant for a fixed finite field such as \mathbb{F}_2 . Observe that since the number of distinct column vectors in \mathbb{F}_q^r is q^r , MATROID GIRTH can be solved in time $q^{r^2} |E|^{\mathcal{O}(1)}$, where $r = \text{rank}(M)$. Furthermore, an algorithm for MATROID GIRTH with running time $\mathcal{O}(q^{\text{rank}(M)+k} \text{rank}(M) \log k)$ can be found as a byproduct in [14] (see Theorem 14 in [14]). However, it was unknown if the additive dependence on k in the exponent can be avoided. In Section 19.1, we give a faster algorithm for MATROID GIRTH. This algorithm gives an exponential

speedup over the previous algorithm when k is close to $\text{rank}(M)$.

Theorem 19.1. *MATROID GIRTH can be solved in time $\mathcal{O}(q^{\text{rank}(M)} \text{rank}(M) + |E|k^2)$.*

Theorems 19.1 imply FPT algorithms parameterized by $\text{rank}(M)$ for all matroids defined over a constant size field (such as graphic matroids and co-graphic matroids). Our lower bounds rule out having an algorithm without having any dependence on the field size. However, it is possible that for certain matroids, for instance, transversal matroids, gammoids and strict gammoids, which are only representable over fields whose size depends on $|E|$, one can obtain an FPT algorithm parameterized by $\text{rank}(M)$ alone. In fact, we give such an algorithm, running in time $2^{\text{rank}(M)}|E|^{\mathcal{O}(1)}$, for transversal matroids. For strict gammoids however, we give a polynomial time algorithm and leave open the same problem for gammoids. In Section 19.2 we explain algorithms for MATROID GIRTH on transversal matroids and strict gammoids.

19.1 Algorithm for MATROID GIRTH

In this section we design a $q^{\text{rank}(M)}|E|^{\mathcal{O}(1)}$ time algorithm for MATROID GIRTH parameterized by $\text{rank}(M)$ using the MacWilliams identity. In what follows we give basics of coding theory and recall the MacWilliams identity.

Coding Theory. A linear code C over a finite field \mathbb{F}_q , defined by $n \times m$ matrix A , is the set of m -dimensional vectors $C = \{vA \mid v \in \mathbb{F}_q^n\}$. The matrix A is called the generator matrix of C . The code C is the linear subspace of \mathbb{F}_q^m spanned by the row vectors of A and its dimension is equal to $\text{rank}(A)$. Without loss of generality we can assume $n = \text{rank}(A)$. A (m, n) -linear code is one such that the length of codewords is m and its dimension is n .

Let C be a linear code with generator matrix A . Let $\vec{0}$ be the zero vector $(0, \dots, 0)^T$. The length of $\vec{0}$ will be clear from the context. The parity check matrix H of C is a $(m - n) \times m$ matrix satisfying $Hw^T = \vec{0}$ for any codeword $w \in C$. It is well-known that there is a duality between generator matrices and parity check matrices: For the code C^\perp with generator matrix H^T , it is easily verified that $Av = \vec{0}$ holds for any $v \in C^\perp$. That is, A is the parity check matrix of C^\perp . The code C^\perp is called the dual code of C . Given a codeword w , the number of non-zero entries in w is called the weight of w and is denoted by $\text{wt}(w)$. The *weight enumerator* of an (m, n) -linear code C is a polynomial in x, y and is given by,

$$W_C(x, y) = \sum_{c \in C} x^{m - \text{wt}(c)} y^{\text{wt}(c)} = \sum_{i=0}^m \xi_i x^{m-i} y^i,$$

where ξ_i is the number of words of weight i in C . The following theorem shows that the *weight enumerator* of C^\perp can be calculated from that of C .

Proposition 19.1 (MacWilliams identity [89]). $W_{C^\perp}(x, y) = \frac{1}{|C|} W_C(x + (q-1)y, x - y)$.

Algorithm. We next prove Theorem 19.1.

Proof of Theorem 19.1. Let (M, k) be an instance to MATROID GIRTH and let A_M be the representation matrix of M of order $r \times m$ over \mathbb{F}_q , where $m = |E|$ and $r = \text{rank}(M)$. Consider the system of linear equations $A_M v = \vec{0}$, where $v = (v_1, \dots, v_m)^t$ is a vector of variables. Recall that girth of a matroid M is denoted by $g(M)$. We have the following claim.

Claim 19.1. $g(M) \leq k$ if and only if there exists a vector $z \in \mathbb{F}_q^m$ with $\text{wt}(z) \leq k$ and $A_M z = \vec{0}$.

Proof. Let $C' \subseteq E$ be a circuit of length at most $\ell \leq k$ in M . Let $W \subseteq \{1, \dots, m\}$ be the set of indices corresponding to the elements of the circuit C' . Since C' is a circuit in M , C' is linearly dependent. Thus, the columns corresponding to indices in W are also linearly dependent. Hence there exist $\lambda_1, \dots, \lambda_\ell \in \mathbb{F}_q$, not all zeros, such that $\sum_{j \in W} \lambda_j A_j = \vec{0}$. Here, A_j denotes the j -th column of A_M . Now consider the vector $z = (z_1, \dots, z_m)^T$, where $z_j = \lambda_j$ if $j \in W$, else $z_j = 0$. Note that $\text{wt}(z) = \ell \leq k$. Since $\sum_{j \in W} \lambda_j A_j = \vec{0}$, we have that $A_M z = \vec{0}$.

Suppose there exists a vector z with $\text{wt}(z) \leq k$ such that $A_M z = \vec{0}$, then $\sum_{i=1}^m z_i A_i = \vec{0}$ where A_i is i^{th} column in A_M . Let $W \subseteq \{1, \dots, m\}$ such that $i \in W$ if and only if $z_i \neq 0$, then $\sum_{i \in W} z_i A_i = \vec{0}$. Since $|W| \leq k$, there exist (at most) k columns in A which are linearly dependent. Hence, $g(M) \leq k$. \square

By Claim 19.1, to show that $g(M) \leq k$, it is sufficient to show that there exists a vector $z \in \mathbb{F}_q^m$ with $\text{wt}(z) \leq k$ and $A_M z = \vec{0}$. Let C be the code generated by

the matrix A_M , i.e., $C = \{vA_M \mid v \in \mathbb{F}_q^r\}$. Let C^\perp be the dual code of C , i.e. $C^\perp = \{u \mid A_M u = \vec{0}\}$. Using the MacWilliams identity we have,

$$W_{C^\perp}(x, y) = \frac{1}{|C|} W_C(x + (q-1)y, x - y). \quad (19.1)$$

Since $|C| \leq q^r$, the polynomial $W_C(x, y)$ can be computed in $\mathcal{O}(q^r r + |E|)$ time. Now we have,

$$W_C(x + (q-1)y, x - y) = \sum_{i=0}^m \xi_i (x + (q-1)y)^{m-i} (x - y)^i,$$

where ξ_i is the number of codewords in C of weight i . Claim 19.1 implies that there exists a circuit of size at most k in M if and only if there exists a codeword z in C^\perp such that $\text{wt}(z) \leq k$, that is, the coefficient of $x^{m-j}y^j$ in $W_{C^\perp}(x, y)$ is non zero for some $j \leq k$ (by the definition of weight enumerator of C^\perp). Due to the MacWilliams identity (Equation 19.1), there exists a codeword z in C^\perp such that $\text{wt}(z) = j$ if and only if coefficient of $x^{m-j}y^j$ in $W_C(x + (q-1)y, x - y)$ is not equal to zero. Using the binomial theorem, we have that the coefficient of $x^{m-j}y^j$ in $\xi_i (x + (q-1)y)^{m-i} (x - y)^i$ is

$$\xi_i \sum_{j'+j''=j} (-1)^{j''} (q-1)^{j'} \binom{m-i}{j'} \binom{i}{j''}.$$

Hence the coefficient of $x^{m-j}y^j$ in $W_C(x + (q-1)y, x - y)$ is

$$\sum_{i=0}^m \xi_i \sum_{j'+j''=j} (-1)^{j''} (q-1)^{j'} \binom{m-i}{j'} \binom{i}{j''} \quad (19.2)$$

Thus we can check whether the coefficient of $x^{m-j}y^j$ in $W_C(x + (q-1)y, x - y)$ is non zero or not in time $\mathcal{O}(q^r + mj)$. We output YES if Equation 19.2 is non zero for any $j \leq k$. Hence the total running time is $\mathcal{O}(q^r r + |E|k^2)$. \square

19.2 MATROID GIRTH on Specific Matroids

In this section we design FPT algorithm for MATROID GIRTH parameterized by rank in transversal matroid and a polynomial time algorithm to find MATROID GIRTH in a strict gammoid.

19.2.1 MATROID GIRTH on Transversal Matroid

We design an FPT algorithm for MATROID GIRTH parameterized by rank in transversal matroid. The problem is defined as follows.

<p>TRANSVERSAL MATROID GIRTH Input: A bipartite graph G with $V(G) = A \uplus B$ and an integer k Question: Does there exist a set $S \subseteq A$ of size at most k such that $N(S) < S$?</p>	<p>Parameter: $r = \text{rank}(M_G)$</p>
--	--

Since the size of a maximum matching in a bipartite graph is equal to the size of a minimum vertex cover of G , the rank of the transversal matroid M_G is equal to the size of the minimum vertex cover.

Lemma 19.1. *Let G be a bipartite graph with the vertex set $V(G)$ being partitioned into A and B and let $X \uplus Y$ be a minimum vertex cover (of size r) of G , where $X \subseteq A$ and $Y \subseteq B$. Let $X' \subseteq X$ and $Y' \subseteq Y$ such that $|X'| \leq k$, $|Y'| < k$, $N(X') \cap Y \subseteq Y'$ and $|N(X') \setminus Y| < k - |Y'|$. Then there exists a hall-set S of size k such that $S \cap X = X'$ and $N(S) \cap Y = Y'$ if and only if there exists a vertex set P of size $k - |X'|$ in $A \setminus X$ such that $N(P) \subseteq Y'$.*

Proof. (\Rightarrow) Let S be a hall-set of size k such that $S \cap X = X'$ and $N(S) \cap Y = Y'$. Note that $N(X') \cap Y \subseteq Y'$. Consider the set $P = S \setminus X$. Note that $P \subseteq A \setminus X$ and $|P| = k - |X'|$. Since $X \cup Y$ is a vertex cover, $N(P) \subseteq Y$ and since $N(S) \cap Y = Y'$, $N(P) \subseteq Y'$.

(\Leftarrow) Let $P \subseteq A \setminus X$ of size $k - |X'|$ such that $N(P) \subseteq Y'$. Now consider the set $S = X' \cup P$. Note that $|S| = k$ and $N(S) = N(X' \cup P) \subseteq Y' \cup (N(X') \setminus Y)$. Since $|N(X') \setminus Y| < k - |Y'|$, S is the required hall-set. \square

Theorem 19.2. TRANSVERSAL MATROID GIRTH can be solved in time $2^r |V(G)|^{\mathcal{O}(1)}$.

Proof. First we compute a minimum vertex cover $X \cup Y$ in G , where $X \subseteq A$ and $Y \subseteq B$. Now we guess $k' \leq k$, the size of a minimum sized hall-set. Let S be a minimum sized hall-set. We also guess $X' = S \cap X$ and $Y' = N(S) \cap Y$. Note that $|Y'|$ should be strictly less than k' and $|N(X') \setminus Y| < k' - |Y'|$. Now by Lemma 19.1, we know that there exist a hall-set S' of size k' such that $X' = S' \cap X$ and $Y' = N(S') \cap Y$ if and only if there exists a vertex set P of size $k' - |X'|$ in $A \setminus X$ such that $N(P) \subseteq Y'$. We can obtain the desired set P by just taking $k' - |X'|$ vertices in $A \setminus X$ whose neighborhood is contained in Y' . Since there are at most 2^r guesses for (X', Y') , the total running time is upper bounded by $2^r |V(G)|^{\mathcal{O}(1)}$. \square

19.2.2 MATROID GIRTH on Strict Gammoids

In this subsection we design a polynomial time algorithm to find MATROID GIRTH in a strict gammoid. The problem is formally defined as follows.

STRICT GAMMOID GIRTH

Input: A directed graph D , $S \subseteq V(D)$ and an integer k .

Question: Does there exist $Q \subseteq V(D)$ of size at most k such that Q is a circuit in the strict gammoid (D, S)

Theorem 19.3. STRICT GAMMOID GIRTH *can be solved in polynomial time.*

Proof. Let C be a circuit in (D, S) . We claim that there exists a vertex $u \in C$ such that there exists a $S - u$ cut, of size strictly less than $|C|$ in D . Since C is a circuit in (D, S) , the number of vertex disjoint paths from S to C is strictly less than $|C|$. Hence due to Menger's theorem, any minimum sized $S - C$ cut is strictly less than $|C|$. Let T be a minimum sized $S - C$ cut. Since $|C| > |T|$, there exists a vertex $u \in C \setminus T$. Hence T is also a $S - u$ cut.

Let $v \in V(D)$ and let $P \subseteq V(D)$ be a minimum sized $S - v$ cut in D . We claim that there is a circuit containing v in $P \cup \{v\}$ in the strict gammoid (D, S) . Towards the proof of the claim, we show that $P \cup \{v\}$ is dependent in (D, S) . Suppose not, then there exist $|P| + 1$ vertex disjoint paths from S to $P \cup \{v\}$ in D , which contradicts the fact that P is a minimum sized $S - v$ cut in D . Since P is a minimum sized

$S - v$ cut in D , there exists $|P|$ vertex disjoint paths from S to P (due to Menger's Theorem), that is P is independent in (D, S) . Hence there is a circuit containing v in $P \cup \{v\}$.

Now the algorithm is as follows. It computes a minimum sized $S - u$ cut for all $u \in V(D)$. If for any $u \in V(D)$ the size of minimum sized $S - u$ cut is less than k , then the algorithm outputs YES, otherwise algorithm outputs NO. Since we can find a minimum sized $S - u$ cut in deterministic polynomial time using flow techniques, STRICT GAMMOID GIRTH can be solved in polynomial time. \square

Chapter 20

Matroid Connectivity

In this chapter we study another problem related to matroids, called the MATROID CONNECTIVITY problem where the objective is to compute the connectivity of a given matroid. The problem is formally defined as,

MATROID CONNECTIVITY

Input: A linear matroid $M = (E, \mathcal{I})$ together with its representation matrix A_M of dimension $\text{rank}(M) \times |E|$ over a field \mathbb{F}_q , and a positive integer k .

Parameters: (1) k , (2) $\text{rank}(M)$ and (3) $\text{rank}(M)+q$

Question: Does M has a k -separation?

This problem generalizes the classical graph problem of computing the connectivity of a given graph. The notion of connectivity defined for matroids is slightly different from the standard one defined for graphs. It would be desirable to have a notion of connectivity from graphs extending to matroids. Unfortunately, the standard notion of edge-connectivity in graphs when extended to matroids does not fit well with the duals of these matroids. With these issues in mind, Tutte [115] proposed the above definition of connectivity for a matroid, which renders it dual-invariant. That is, a matroid is ℓ -connected if and only if its dual is. Finally, we note that Oxley [102] has shown how Tutte's definition of matroid connectivity can be modified to give a matroid concept that directly generalizes the notion of connectivity on graphs.

Although it is easy to observe that MATROID GIRTH admits an algorithm with running time $|E|^{\mathcal{O}(k)}$, it is not at all obvious that MATROID CONNECTIVITY admits an algorithm that is polynomial for every fixed integer k . Bixby and Cunningham, using an algorithm for matroid intersection, gave an algorithm for MATROID CONNECTIVITY running in time $|E|^{\mathcal{O}(k)}$ (see [15] for details). In Section 20.1 we show that MATROID CONNECTIVITY is not in P unless P=NP. In fact the proof of this result also gives us the results that MATROID CONNECTIVITY is not FPT unless $\text{FPT} = W[1]$, when parameterized by $\text{rank}(M)$ or k .

Therefore we choose our parameter to be $\text{rank}(M)+q$, where q is the size of the field in which the matroid is represented. In Section 20.2 we give a branching algorithm for MATROID CONNECTIVITY which has a single-exponential dependence on the rank. The main features of this algorithm are the use of the rank of matroids obtained in the subproblems as a measure to quantify the progress of the algorithm and application of the algorithm given by Theorem 19.1 to solve the base cases. Thus, one can view this algorithm as a parameterized Turing-reduction from MATROID CONNECTIVITY to MATROID GIRTH. Formally, we obtain the following theorem.

Theorem 20.1. MATROID CONNECTIVITY *can be solved in time*

$$\mathcal{O}\left(2^{\text{rank}(M)+k} \cdot \text{rank}(M)^2 \cdot |E| \cdot \left(q^{\text{rank}(M)\text{rank}(M) + |E|k^2}\right)\right).$$

20.1 Hardness

In this section we first show that MATROID CONNECTIVITY cannot be solved in polynomial time unless $P = NP$. Then we explain how the same proof gives $W[1]$ -hardness result for MATROID CONNECTIVITY where parameterized by $\text{rank}(M)$ or k . Towards the proof, we need to consider the complement of LINEAR DEGENERACY problem, named UNIFORM MATROID ISOMORPHISM problem.

UNIFORM MATROID ISOMORPHISM (UMI)

Input: A $k \times m$ matrix M of rank k .

Question: Is M isomorphic to $U_{m,k}$? I.e., is every k sized subset of columns of M linearly independent?

Khachiyan et al [73] showed that LINEAR DEGENERACY is NP-hard, by giving a reduction from SMALL SUBSET SUM. This implies that UMI is not in P unless $P = NP$. The following known lemma is needed to prove the hardness of MATROID CONNECTIVITY.

Lemma 20.1. *Let $M = (E, \mathcal{I})$ be a matroid of rank k and $m = |E| > 2k + 1$. Then*

1. *M is isomorphic to $U_{m,k}$, if and only if the girth of M is $k + 1$.*
2. *If $\kappa(M) = k + 1$ then $g(M) = k + 1$.*

Proof. We first prove the condition 1 in the lemma. If M is isomorphic to $U_{m,k}$, then the smallest dependent set size is $k + 1$ and hence its girth is $k + 1$. Now if

the girth of M is $k + 1$, then any k sized set of columns in M is independent. This implies that M is isomorphic to $U_{m,k}$.

Now we show the condition 2 in the lemma. Recall that M^* is the dual of M . Towards that we first show that $\kappa(M) \leq \min\{g(M), g(M^*)\}$. Let C be a smallest length circuit in M . Note that $r_M(C) + r_M(E \setminus C) - r_M(E) \leq r(C) \leq |C| - 1$. Hence, $\kappa(M) \leq g(M)$. Since, $\kappa(M) = \kappa(M^*)$ [115], we have that $\kappa(M) = \kappa(M^*) \leq g(M^*)$. Since, $\text{rank}(M) = k$, $g(M) \leq k + 1$. Furthermore, we have shown above that $\kappa(M) \leq \min\{g(M), g(M^*)\}$. This implies that if $\kappa(M) = k + 1$ then $g(M) = k + 1$. \square

Theorem 20.2. MATROID CONNECTIVITY is not in P unless P=NP.

Proof. We prove the theorem by designing a polynomial time algorithm for UMI, which uses an algorithm for MATROID CONNECTIVITY as a subroutine. Now given a $k \times m$ matrix M of rank k , we want to test whether M is isomorphic to $U_{m,k}$. We assume that $m > 2k + 1$. Note that M is a uniform matroid if and only if the dual of M is also a uniform matroid. So without loss of generality we can assume that $k \leq m - k$, otherwise instead of checking whether M is isomorphic to $U_{m,k}$, it is enough to check whether dual of M is isomorphic to $U_{m,m-k}$.

Now we describe an algorithm to solve UMI using an algorithm for MATROID CONNECTIVITY. Using an algorithm for MATROID CONNECTIVITY, find the least integer, $i \leq k + 1$, if it exists, such that M has i -separation. This implies that $\kappa(M) = i$. If $i = k + 1$ then we output YES, otherwise we output NO. Now we show that our algorithm is correct. If the algorithm outputs YES, then $\kappa(M) = k + 1$. Then by Lemma 20.1, we have that $g(M) = k + 1$ and M is isomorphic to $U_{m,k}$. Now consider the following claim.

Claim 20.1. $\kappa(U_{m,k}) = k + 1$

Proof. Let E be the ground set of $U_{m,k}$ and let r be the rank function of $U_{m,k}$. Let $C \subseteq E$ be an arbitrary subset of size $k + 1$. Note that $r(C) + r(E \setminus C) - r(E) \leq$

$r(C) \leq |C| - 1$ and $|C|, |E \setminus C| \geq k + 1$. Hence, $\kappa(U_{m,k}) \leq k + 1$. Next we show that there does not exist $i < k + 1$ such that $\kappa(U_{m,k}) = i$. We prove the statement via contradiction. Suppose there exists $i < k + 1$ such that $\kappa(U_{m,k}) = i$. Let $(A, E \setminus A)$ be an i -separation. Since $m > 2k$, we have that $|A| > k$ or $|E \setminus A| > k$. Assume without loss of generality that $|E \setminus A| > k$. Then $r(A) + r(E \setminus A) - r(E) = r(A) \geq i$ (since $|A| \geq i$ and $i \leq k$). This leads to a contradiction that $(A, E \setminus A)$ be an i -separation. This completes the proof of the lemma. \square

If M is isomorphic to $U_{m,k}$ then by Claim 20.1, we have that $\kappa(M) = k + 1$ and thus the algorithm will output YES. This completes the proof of correctness of our algorithm. Hence, if we do have a polynomial time algorithm for MATROID CONNECTIVITY then we do have a polynomial time algorithm for UMI. \square

We have mentioned in Chapter 19 that LINEAR DEGENERACY parameterized by k is $W[1]$ -hard. Thus, combining this fact with Theorem 20.2 we have the following theorem.

Theorem 20.3. *MATROID CONNECTIVITY parameterized by $\text{rank}(M)$ is not FPT unless $\text{FPT} = W[1]$.*

As $\kappa(M) \leq \text{rank}(M) + 1$, MATROID CONNECTIVITY parameterized by k is also not FPT unless $\text{FPT} = W[1]$.

20.2 Algorithm for MATROID CONNECTIVITY

In this section we design a fast FPT algorithm for MATROID CONNECTIVITY parameterized by $\text{rank}(M) + q$. Our algorithm is a recursive algorithm and at the leaves of the search tree it runs the algorithm for MATROID GIRTH as a subroutine.

In what follows, we say that a partition (X, Y) of the ground set E of a matroid M , obeys the pair (X_1, Y_1) where $X_1, Y_1 \subseteq E$, if $X_1 \subseteq X$ and $Y_1 \subseteq Y$. We start with two lemmas which are useful for our algorithm.

Lemma 20.2. *Let $M = (E, \mathcal{I})$ be a matroid. Let $X, Y \subseteq E$ such that $X \cap Y = \emptyset$ and $|Y| \geq k$. Let $S_x = \text{cl}(X) \cap (E \setminus (X \cup Y))$. If there exist a k -separation (X', Y')*

obeying the pair (X, Y) , then there exist a k -separation (X'', Y'') obeying the pair $(X \cup S_x, Y)$.

Proof. Let (X', Y') be a k -separation obeying the pair (X, Y) . Then we know that $r(X') + r(Y') - r(E) \leq k - 1$. Now consider the partition $(X' \cup S_x, Y' \setminus S_x)$. We claim that $(X' \cup S_x, Y' \setminus S_x)$ is a k -separation because $|X' \cup S_x| \geq |X'| \geq k$, $|Y' \setminus S_x| \geq |Y| \geq k$ and $r(X' \cup S_x) + r(Y' \setminus S_x) - r(E) \leq r(X') + r(Y') - r(E) \leq k - 1$. The pair $(X' \cup S_x, Y' \setminus S_x)$ obeys $(X \cup S_x, Y)$ because $X \cup S_x \subseteq X' \cup S_x$ and $Y \subseteq Y' \setminus S_x$. \square

Lemma 20.3. *Let $M = (E, \mathcal{I})$ be a matroid. Let $X, Y \subseteq E$ such that $X \cap Y = \emptyset$, $|Y| \geq k$, $|X| < k$, $r(Y) = r(E)$, and $r(X) = |X|$. Then there exists a k -separation (X', Y') obeying (X, Y) if and only if there exists a circuit C in the matroid M/X of size at most $k - |X|$ contained in $E \setminus (X \cup Y)$.*

Proof. (\Rightarrow) Suppose there exists a k -separation (X', Y') obeying (X, Y) . We need to show that there exists a circuit C in the matroid M/X , of size at most $k - |X|$, contained in $E \setminus (X \cup Y)$. Since (X', Y') is a k -separation, we have that $r(X') + r(Y') - r(E) \leq k - 1$. This implies that $r(X') \leq k - 1$, because $r(Y') = r(E)$. Since (X', Y') is a k -separation, $|X'| \geq k$. Consider a minimum sized set $S \subseteq X'$ such that $X \subseteq S$ and $r(S) = |S| - 1$ (such a set S exists because $|X'| \geq k$, $r(X') \leq k - 1$ and $r(X) = |X|$). Also note that $|S| \leq k$. Since $r(S) = |S| - 1$, there exists a circuit C' in S .

Now we claim that there exists a circuit in M/X of size at most $k - |X|$ contained in $C' \setminus X$. It is easy to see that $r_{M/X}(C' \setminus X) \leq |C' \setminus X| - 1$. Now consider the size of $C' \setminus X$.

$$\begin{aligned} |C' \setminus X| &= |C'| - |X| + |X \setminus C'| \\ &\leq |S| - |X \setminus C'| - |X| + |X \setminus C'| \quad (\text{Since } C' \subseteq S \text{ and } X \subseteq S) \\ &\leq k - |X| \quad (\text{Since } |S| \leq k) \end{aligned}$$

Hence there exists a circuit in M/X of size at most $k - |X|$ contained in $C' \setminus X \subseteq E \setminus (X \cup Y)$.

(\Leftarrow) Suppose there exists a circuit C in M/X of size at most $k - |X|$, contained in $E \setminus (X \cup Y)$. Then we claim that $(C \cup X \cup S, E \setminus (C \cup X \cup S))$ is a k -separation obeying (X, Y) , where S is an arbitrary $k - |C \cup X|$ sized set in $E \setminus (C \cup X \cup Y)$. Note that $|C \cup X \cup S|, |E \setminus (C \cup X \cup S)| \geq k$ and

$$\begin{aligned}
r(C \cup X \cup S) + r(E \setminus (C \cup X \cup S)) - r(E) &\leq r(C \cup X \cup S) \\
&\leq r_{M/X}(C) + r(X) + r(S) \\
&\leq |C| - 1 + |X| + k - |C \cup X| \\
&\leq k - 1.
\end{aligned}$$

This completes the proof. □

Now we prove the Theorem [20.1](#).

Theorem 20.1. MATROID CONNECTIVITY *can be solved in time*

$$\mathcal{O}\left(2^{\text{rank}(M)+k} \cdot \text{rank}(M)^2 \cdot |E| \cdot \left(q^{\text{rank}(M)\text{rank}(M) + |E|k^2}\right)\right).$$

Proof. Let $r = \text{rank}(M)$. Since $\kappa(M) \leq \text{rank}(M)+1$, we can assume that $k \leq r$. We design a branching algorithm which gradually creates a solution (X, Y) starting from the pair (\emptyset, \emptyset) . At any point in the branching algorithm, we branch on a carefully chosen element from $E \setminus (X \cup Y)$. Our branching rules are the following, applied in the order in which they are listed.

- **Rule 1:** If there exists an element $e \in E \setminus (X \cup Y)$ such that $e \notin \text{cl}(X) \cup \text{cl}(Y)$, we branch on e by adding e to X or Y .
- **Rule 2:** If $|X|, |Y| < k$, then we branch on an arbitrary element $e \in$

$$E \setminus (X \cup Y).$$

In any node of the branching tree of the algorithm we have a potential partial solution (X, Y) , and we abort if $r(X) + r(Y) - r(E) \geq k$. Now we claim that there will not be an application of Rule 1 after an application of Rule 2. Consider a node of the branching tree of the algorithm, with a potential partial solution (X, Y) . We apply Rule 2, only if Rule 1 is not applicable, that is when for all $e \in E \setminus (X \cup Y)$, $e \in \text{cl}(X) \cup \text{cl}(Y)$. Hence for any $X' \supseteq X, Y' \supseteq Y$, for all $e \in E \setminus (X' \cup Y')$, $e \in \text{cl}(X') \cup \text{cl}(Y')$. This implies Rule 1 is not applicable after an application of Rule 2. Now consider any root to leaf path in the branching tree of the algorithm. If there exists an application of Rule 2 in this path then the length of the path is at most $2k$, because Rule 2 is applicable only if $|X|, |Y| < k$. Otherwise, we claim that the length of the path is at most $r + k$. Suppose not. Consider the leaf node and the potential partial solution (X, Y) associated with it. If the length of the path is more than $r + k$ and if we only used branching Rule 1, then $r(X) + r(Y) - r(E) > r + k - r(E) > k$, which is a contradiction (because we should have aborted this branch). Hence the height of the branching tree is at most $r + k$ (since $k \leq r$).

Now we explain how to compute a solution from a leaf node labeled (X, Y) , if there exists a solution obeying (X, Y) . Note that for all $e \in E \setminus (X \cup Y)$, $e \in \text{cl}(X) \cup \text{cl}(Y)$ because of Rule 1. Also note that either $|X| \geq k$ or $|Y| \geq k$. Without loss of generality assume that $|Y| \geq k$. Now we can apply Lemma 20.2 and add S_x to X where $S_x = \text{cl}(X) \cap (E \setminus (X \cup Y))$. Now if $|X \cup S_x| \geq k$, then we can output the partition $(X \cup S_x, E \setminus (X \cup S_x))$ as k -separation, because $r(X \cup S_x) + r(E \setminus (X \cup S_x)) - r(E) = r(X) + r(Y) - r(E) \leq k - 1$ (because we did not abort this branch). Otherwise $|X \cup S_x| < k$. For convenience, now we use (X, Y) to denote the partial solution $(X \cup S_x, Y)$. The properties of (X, Y) are $|X| < k, |Y| \geq k$, for all $e \in E \setminus (X \cup Y)$ $e \in \text{cl}(Y)$ and $e \notin \text{cl}(X)$. If $r(X) < |X|$, then we can output $(X \cup S, E \setminus (X \cup S))$ as a k -partition where S is an arbitrary set of $k - |X|$ elements

from $E \setminus (X \cup Y)$, because

$$\begin{aligned}
r(X \cup S) + r(E \setminus (X \cup S)) - r(E) &\leq r(X) + r(S) + r(E \setminus (X \cup S)) - r(E) \\
&\leq r(X) + k - |X| + r(E \setminus (X \cup S)) - r(E) \\
&\leq k - 1 \quad (\text{Since } r(X) < |X|)
\end{aligned}$$

If $r(X) = |X|$ and $r(Y) < r(E)$, then we can output the $(X \cup S, E \setminus (X \cup S))$ as a k -partition where S is an arbitrary set of $k - |X|$ elements from $E \setminus (X \cup Y)$, because

$$\begin{aligned}
r(X \cup S) + r(E \setminus (X \cup S)) - r(E) &\leq k + r(E \setminus (X \cup S)) - r(E) \\
&\leq k + r(Y) - r(E) \leq k - 1.
\end{aligned}$$

Now if $r(X) = |X|$, $|X| < k$, $r(Y) = r(E)$, $|Y| \geq k$, then we apply Lemma 20.3 and output YES if there exists a circuit of size at most $k - |X|$ in M/X , otherwise abort this particular branch. A linear representation of M/X and different case analysis explained above can be computed in time $\mathcal{O}((\text{rank}(M))^2|E|)$ field operations using Gaussian elimination. Since the height of the branching tree is at most $r + k$, the algorithm runs in $\mathcal{O}(2^{\text{rank}(M)+k} \cdot (\text{rank}(M))^2|E|(q^{\text{rank}(M)}\text{rank}(M) + |E|k^2))$ time. □

Part V

Steiner Tree

Chapter 21

Polynomial Space Single Exponential FPT Algorithm

In this chapter we give a polynomial space single exponential FPT algorithm for the STEINER TREE problem, which is define formally as,

STEINER TREE

Parameter: $k = |T|$

Input: A connected n -vertex graph, a non-negative weight function $w : E(G) \rightarrow \{1, 2, \dots, W\}$, and a set of terminal vertices $T \subseteq V(G)$.

Question: Find a minimum-weight connected subgraph ST of G containing all terminal nodes T .

STEINER TREE is one of the central and best-studied problems in Computer Science with various applications. We refer to the book of Prömel and Steger [111] for an overview of the results and applications of the Steiner tree problem. STEINER TREE is known to be APX-complete, even when the graph is complete and all edge costs are either 1 or 2 [13]. On the other hand the problem admits a constant factor approximation algorithm, the currently best such algorithm (after a long chain of improvements) is due to Byrka et al. and has approximation ratio $\ln 4 + \varepsilon < 1.39$ [26].

STEINER TREE is a fundamental problem in parameterized algorithms [41]. The classic algorithm for STEINER TREE of Dreyfus and Wagner [44] from 1971 might well be the first parameterized algorithm for *any* problem. The study of parameterized algorithms for STEINER TREE has led to the design of important techniques, such as Fast Subset Convolution [16] and the use of branching walks [101]. Research on the parameterized complexity of STEINER TREE is still on-going, with very recent significant advances for the planar version of the problem [107, 108].

Algorithms for STEINER TREE are frequently used as a subroutine in fixed-parameter

tractable (FPT) algorithms for other problems; examples include vertex cover problems [68], near-perfect phylogenetic tree reconstruction [20], and connectivity augmentation problems [8].

Motivation and earlier work. For more than 30 years, the fastest FPT algorithm for STEINER TREE was the $3^k \cdot \log W \cdot n^{\mathcal{O}(1)}$ -time dynamic programming algorithm by Dreyfus and Wagner [44]. Fuchs et al. [60] gave an improved algorithm with running time $\mathcal{O}((2 + \varepsilon)^k n^{f(1/\varepsilon)} \log W)$. For the unweighted version of the problem, Björklund et al. [16] gave a $2^k n^{\mathcal{O}(1)}$ time algorithm. All of these algorithms are based on dynamic programming and use exponential space.

Algorithms with high space complexity are in practice more constrained because the amount of memory is not easily scaled beyond hardware constraints whereas time complexity can be alleviated by allowing for more time for the algorithm to finish. Furthermore, algorithms with low space complexity are typically easier to parallelize and more cache-friendly. These considerations motivate a quest for algorithms whose memory requirements scale polynomially in the size of the input, even if such algorithms may be slower than their exponential-space counterparts. The first polynomial space $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ -time algorithm for the unweighted STEINER TREE problem is due to Nederlof [101]. This algorithm runs in time $2^k n^{\mathcal{O}(1)}$, matching the running time of the best known exponential space algorithm. Nederlof’s algorithm can be extended to the weighted case, unfortunately this comes at the cost of a $\mathcal{O}(W)$ factor both in the time and the space complexity. Lokshtanov and Nederlof [86] showed that the $\mathcal{O}(W)$ factor can be removed from the space bound, but with a factor $\mathcal{O}(W)$ in the running time. The algorithm of Lokshtanov and Nederlof [86] runs in $2^k \cdot n^{\mathcal{O}(1)} \cdot W$ time and uses $n^{\mathcal{O}(1)} \log W$ space. Note that both the algorithm of Nederlof [101] and the algorithm of Lokstanov and Nederlof [86] have a $\mathcal{O}(W)$ factor in their running time. Thus the running time of these algorithms depends exponentially on the input size, and therefore these algorithms are not FPT algorithms for weighted STEINER TREE.

For weighted STEINER TREE, the only known polynomial space FPT algorithm has a $2^{\mathcal{O}(k \log k)}$ running time dependence on the parameter k . This algorithm follows from combining a $(27/4)^k \cdot n^{\mathcal{O}(\log k)} \cdot \log W$ time, polynomial space algorithm by Fomin et al. [50] with the Dreyfus–Wagner algorithm. Indeed, one runs the algorithm of Fomin et al. [50] if $n \leq 2^k$, and the Dreyfus–Wagner algorithm if $n > 2^k$. If $n \leq 2^k$, the running time of the algorithm of Fomin et al. is bounded from above by $2^{\mathcal{O}(k \log k)}$. When $n > 2^k$, the Dreyfus–Wagner algorithm becomes a polynomial time (and space) algorithm.

Prior to our work the existence of a polynomial space algorithm with running time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)} \cdot \log W$, i.e a single exponential time polynomial space FPT algorithm, was an open problem asked explicitly in [50, 86].

Contributions and methodology. The starting point of our present algorithm is the $(27/4)^k \cdot n^{\mathcal{O}(\log k)} \cdot \log W$ -time, polynomial-space algorithm by Fomin et al. [50].

This algorithm crucially exploits the *balanced separation* of Steiner trees (for details see Lemma 21.1 below). Specifically, an optimal Steiner tree ST can be partitioned into two trees ST_1 and ST_2 containing the terminal sets T_1 and T_2 respectively, so that the following three properties are satisfied: (a) The two trees share exactly one vertex v and no edges. (b) Neither of the two trees ST_1 or ST_2 contain more than a $2/3$ fraction of the terminal set T . (c) The tree ST_1 is an optimal Steiner tree for the terminal set $T_1 \cup \{v\}$, and ST_2 is an optimal Steiner tree for the terminal set $T_2 \cup \{v\}$.

Dually, to find the optimal tree ST for the terminal set T it suffices to (a) guess the vertex v , (b) partition T into T_1 and T_2 , and (c) recursively find optimal trees for the terminal sets $T_1 \cup \{v\}$ and $T_2 \cup \{v\}$. Since there are n choices for v , and $\binom{k}{k/3}$ ways to partition T into two sets T_1 and T_2 such that $|T_1| = |T|/3$, the running time of the algorithm is essentially governed by the recurrence

$$T(n, k) \leq n \cdot \binom{k}{k/3} \cdot (T(n, k/3) + T(n, 2k/3)). \quad (21.1)$$

Unraveling (21.1) gives the $(27/4)^k \cdot n^{\mathcal{O}(\log k)} \cdot \log W$ upper bound for the running time, and it is easy to see that the algorithm runs in polynomial space. However, this algorithm is not an FPT algorithm because of the $n^{\mathcal{O}(\log k)}$ factor in the running time.

The factor $n^{\mathcal{O}(\log k)}$ is incurred by the factor n in (21.1), which in turn originates from the need to iterate over all possible choices for the vertex v in each recursive call. In effect the recursion tracks an $\mathcal{O}(\log k)$ -sized set S of *split vertices* (together with a subset T' of the terminal vertices T) when it traverses the recursion tree from the root to a leaf.

The key idea in our new algorithm is to redesign the recurrence for optimal Steiner trees so that we obtain control over the size of S using an alternation between

1. balanced separation steps (as described above), and
2. novel *resplitting* steps that maintain the size of S at no more than 3 vertices throughout the recurrence.

In essence, a resplit takes a set S of size 3 and splits that set into three sets of size 2 by combining each element in S with an arbitrary vertex v , while at the same time splitting the terminal set T' into three parts in all possible (not only balanced) ways. While the combinatorial intuition for resplitting is elementary (see Lemma 21.2 below), the implementation and analysis requires a somewhat careful combination of ingredients.

Namely, to run in polynomial space, it is not possible to use extensive amounts of memory to store intermediate results to avoid recomputation. Yet, if no memoization is used, the novel recurrence does not lead to an FPT algorithm, let alone to a single-exponential FPT algorithm. Thus neither a purely dynamic programming nor a

purely recursive implementation will lead to the desired algorithm. *A combination of the two will, however, give a single-exponential time algorithm that uses polynomial space.*

Roughly, our approach is to employ recursive evaluation over subsets T' of the terminal set T , but each recursive call with T' will compute and return the optimal solutions for every possible set S of split vertices. Since by resplitting we have arranged that S always has size at most 3, this hybrid evaluation approach will use polynomial space. Since each recursive call on T' yields the optimum weights for every possible S , we can use dynamic programming to efficiently combine these weights so that single-exponential running time results.

In precise terms, our main result is as follows:

Theorem 21.1. *STEINER TREE can be solved in time $\mathcal{O}(7.97^k n^4 \log nW)$ time using $\mathcal{O}(n^3 \log nW \log k)$ space.*

Whereas our main result seeks to optimize the polynomial dependency in n for both the running time and space usage, it is possible to trade between polynomial dependency in n and the single-exponential dependency in k to obtain faster running time as a function k , but at the cost of increased running time and space usage as a function of n . In particular, we can use larger (but still constant-size) sets S to avoid re-computation and to arrive at a somewhat faster algorithm:

Theorem 21.2. *There exists a polynomial-space algorithm for STEINER TREE running in $\mathcal{O}(6.751^k n^{O(1)} \log W)$ time.*

By using further balanced separators, but of large size, we show the following theorem.

Theorem 21.3. *For any $\epsilon > 0$, there is a $n^{\mathcal{O}(f'(\epsilon))} \log W$ space $4^{(1+\epsilon)k} n^{\mathcal{O}(f(\epsilon))} \log W$ time algorithm for STEINER TREE, where f and f' are computable functions depends only on ϵ .*

Organization of the Chapter. In Section 21.1 we define some notations specific to this chapter and define and state balanced separators used to design our algorithms. In Section 21.2 we prove Theorems 21.1 and 21.2. In Section 21.3 we prove Theorem 21.3.

21.1 Notations and Separators

Partition. A partition of a set U , is a collection nonempty, pairwise disjoint subsets of U whose union is U . The subsets are called *blocks*. For a set U , we write

$U_1 \uplus U_2 \uplus \dots \uplus U_\ell = U$ if U_1, U_2, \dots, U_ℓ is a partition of U . For a set U and a positive integer i , we use $\mathcal{P}(U)$ and $\mathcal{P}_i(U)$ to denote the set of all partitions of U and set of all partitions of U into i parts respectively. For a set U and a constant $\epsilon \geq 0$, we use $\mathcal{B}_\epsilon(U)$ to denote the set all partitions (U_1, U_2) of U into two parts such that $|U_1|, |U_2| \leq (\frac{1}{2} + \epsilon) |U|$. For a set $U, U' \subseteq U$ and a partition $P \in \mathcal{P}(U)$ we use $P[U']$ to denote the restriction of partition P on the set U' , i.e, blocks in $P[U']$ is obtained by deleting $U \setminus U'$ from the blocks of P . For a set U and partitions $P_1, P_2 \in \mathcal{P}(U)$, we say partition P_1 *refines* partition P_2 , denoted by $P_1 \preceq P_2$, if every block of P_1 is contained in some block of P_2 . We also use $P_1 \preceq P_2$, if P_1 is a restriction of a partition of $\mathcal{P}(U)$ which *refines* partition P_2 . That is, for a set $U, U' \subseteq U$ and partitions $P_1 \in \mathcal{P}(U')$ and $P_2 \in \mathcal{P}(U)$, we denote $P_1 \preceq P_2$, if each block of P_1 is contained in some block of P_2 . For two partitions P_1 and P_2 in $\mathcal{P}(U)$, the *join* of P_1 and P_2 , denoted by $P_1 \sqcup P_2$ is the smallest (with respect to \preceq) partition P refined by both P_1 and P_2 . For a graph G , we use P_G to denote the partition $\{V(C) \mid C \text{ is a connected component of } G\}$ of $V(G)$.

Separation and resplitting. A set of nodes S is called an α -*separator* of a graph G , $0 < \alpha \leq 1$, if the vertex set $V(G) \setminus S$ can be partitioned into sets V_L and V_R of size at most αn each, such that no vertex of V_L is adjacent to any vertex of V_R . We next define a similar notion, which turns out to be useful for Steiner trees. Given a Steiner tree ST on terminals T , an α -Steiner separator S of ST is a subset of nodes which partitions $ST - S$ in two forests \mathcal{R}_1 and \mathcal{R}_2 , each one containing at most αk terminals from T .

Lemma 21.1 (Separation). [22, 50] *Every Steiner tree ST on terminal set T , $|T| \geq 3$, has a $2/3$ -Steiner separator $S = \{s\}$ of size one.*

The following easy lemma enables us to control the size of the split S set at no more than 3 vertices.

Lemma 21.2 (Resplitting). *Let F be a tree and $S \in \binom{V(F)}{3}$. Then there is a vertex $v \in V(F)$ such that each connected component in $F - v$ contains at most one vertex of S .*

Proof. Let $S = \{s_1, s_2, s_3\}$. Let P_1 be the unique path between s_1 and s_3 in the tree F . Let P_2 be the unique path between s_3 and s_2 in the tree F . If P_1 and P_2 are edge disjoint then $V(P_1) \cap V(P_2) = \{s_3\}$ and $P_1 P_2$ is the unique path between s_1 and s_2 . Thus any connected component in $G - s_3$ will not contain both s_1 and s_2 . In this case s_3 is the required vertex. Suppose $V(P_1) \cap V(P_2) \neq \{s_3\}$. Consider the unique path $\overleftarrow{P_1}$ between s_3 and s_1 , which is the reverse of the path P_1 . Since F is a

tree these paths \overleftarrow{P}_1 and P_2 will be of the form $P_1 = Q\overleftarrow{P}'_1$ and $P_2 = QP'_2$. Note that Q is a path starting at s_3 . Let w be the last vertex in the path Q . Since F is a tree $V(\overleftarrow{P}'_1) \cap V(P'_2) = \{w\}$. Now consider the graph $G - w$. Any connected component in $G - w$ will not contain more than one from $\{s_1, s_2, s_3\}$, because the unique path between any pair of vertices in $\{s_1, s_2, s_3\}$ passes through w . \square

21.2 Algorithm

In this section we design an algorithm for STEINER TREE which runs in time $\mathcal{O}(7.97^k n^4 \log nW)$ using $\mathcal{O}(n^3 \log nW \log k)$ space. The minimum weight of a Steiner tree of G on terminals T is denoted by $st_G(T)$. Most algorithms for STEINER TREE, including ours, are based on recurrence relations that reduce finding the optimal Steiner tree to finding optimal Steiner trees in the same graph, but with a smaller terminal set. We will define four functions f_i for $i \in \{0, 1, 2, 3\}$. Each function f_i takes as input a vertex set S of size at most i and a subset T' of T . The function $f_i(S, T')$ returns a real number. We will *define* the functions using recurrence relations, and then prove that $f_i(S, T')$ is exactly $st_G(T' \cup S)$.

For $T' \subseteq T$, $i \in \{0, 1, 2, 3\}$, and $S \in \binom{V(G)}{\leq i}$, we define $f_i(S, T')$ as follows. When $|T'| \leq 2$, $f_i(S, T') = st_G(T' \cup S)$. For $|T'| \geq 3$, we define $f_i(S, T')$ using the following recurrences.

Separation. For $i \in \{0, 1, 2\}$, let us define

$$f_i(S, T') = \min_{(T_1, T_2) \in \mathcal{B}_{\frac{1}{6}}(T')} \min_{\substack{v \in V(G) \\ S_1 \uplus S_2 = S}} f_{i+1}(S_1 \cup \{v\}, T_1) + f_{i+1}(S_2 \cup \{v\}, T_2) \quad (21.2)$$

Resplitting. For $i = 3$, let us define

$$f_i(S, T') = \min_{(T_1, T_2, T_3) \in \mathcal{P}_3(T')} \min_{\substack{S_1 \uplus S_2 \uplus S_3 = S \\ |S_1|, |S_2|, |S_3| \leq i-2 \\ v \in V(G)}} \sum_{r=1}^3 f_{i-1}(S_r \cup \{v\}, T_r) \quad (21.3)$$

The recurrences (21.2) and (21.3) are recurrence relations for STEINER TREE:

Lemma 21.3. *For all $T' \subseteq T$, $0 \leq i \leq 3$, and $S \in \binom{V(G)}{\leq i}$ it holds that $f_i(S, T') = st_G(T' \cup S)$.*

Proof. We prove the lemma using induction on $|T'|$. For the base case $|T'| \leq 2$ the lemma holds by the definition of f_i . For inductive step, let us assume that the lemma holds for all T'' of size less than j . We proceed to show that $f_i(S, T') = st_G(T' \cup S)$ for all $T' \subseteq T$ with $|T'| = j$. We split into cases based on i and in each case establish inequalities $f_i(S, T') \leq st_G(T' \cup S)$ and $f_i(S, T') \geq st_G(T' \cup S)$ to conclude equality.

Case 1: $0 \leq i \leq 2$. By (21.2), we know that there is a vertex $v \in V(G)$, $S_1 \uplus S_2 = S$ and a partition $(T_1, T_2) \in \mathcal{B}_{1/6}(T')$ such that $f_i(S, T') = f_{i+1}(S_1 \cup \{v\}, T_1) + f_{i+1}(S_2 \cup \{v\}, T_2)$. Since $(T_1, T_2) \in \mathcal{B}_{1/6}(T')$ and $|T'| \geq 3$, we have that $|T_1|, |T_2| < |T'| \leq$. Then by induction hypothesis $f_{i+1}(S_1 \cup \{v\}, T_1) = st_G(T_1 \cup S_1 \cup \{v\})$ and $f_{i+1}(S_2 \cup \{v\}, T_2) = st_G(T_2 \cup S_2 \cup \{v\})$. So we have that $f_i(S, T') = st_G(T_1 \cup S_1 \cup \{v\}) + st_G(T_2 \cup S_2 \cup \{v\})$. Let ST_1 be an optimum Steiner tree for the set of terminals $T_1 \cup S_1 \cup \{v\}$ and ST_2 be an optimum Steiner tree for the set of terminals $T_2 \cup S_2 \cup \{v\}$. Note that $ST_1 + ST_2$ is a connected subgraph containing $T_1 \cup T_2 \cup S$ and $w(E(ST_1 + ST_2)) \leq st_G(T_1 \cup S_1 \cup \{v\}) + st_G(T_2 \cup S_2 \cup \{v\})$. This implies that $st_G(T' \cup S) \leq w(E(ST_1 + ST_2)) \leq st_G(T_1 \cup S_1 \cup \{v\}) + st_G(T_2 \cup S_2 \cup \{v\}) = f_i(S, T')$. Hence $f_i(S, T') \geq st_G(T' \cup S)$.

Conversely, let ST be an optimum Steiner tree for the set of terminals $T' \cup S$. Thus ST is also a Steiner tree for the set of terminals T' . Hence by Lemma 21.1, we know that there is a $2/3$ -Steiner separator $\{v\}$ of size one. Let F_1 and F_2 be two forests created by the separator $\{v\}$, such that $V(F_r) \cap T' \leq 2|T'|/3$ for each $1 \leq r \leq 2$. If $v \in T'$ and $|T_1| \leq |T_2|$, then we replace T_1 with $T_1 \cup \{v\}$. If $v \in T'$ and $|T_1| > |T_2|$, then we replace T_2 with $T_2 \cup \{v\}$. Note that (T_1, T_2) is a partition of T' . Since $\{v\}$ is a $2/3$ -Steiner separator and $|T'| \geq 3$, we have that $|T_1|, |T_2| \leq 2|T'|/3 \leq (\frac{1}{2} + \frac{1}{6})|T'| < |T'|$. Hence $(T_1, T_2) \in \mathcal{B}_{1/6}(T')$. Let $S_r = V(ST_r) \cap S$ and $T_r = V(F_r) \cap T'$, $1 \leq r \leq 2$. Thus $f_{i+1}(S_1 \cup \{v\}, T_1) + f_{i+1}(S_2 \cup \{v\}, T_2) \geq f_i(S, T')$. Note that $ST_1 = ST[V(F_1) \cup \{v\}]$ and $ST_2 = ST[V(F_2) \cup \{v\}]$ are subtrees of ST . By the induction hypothesis, we have that $f_{i+1}(S_1 \cup \{v\}, T_1) = st_G(T_1 \cup S_1 \cup \{v\})$ and $f_{i+1}(S_2 \cup \{v\}, T_2) = st_G(T_2 \cup S_2 \cup \{v\})$. Since ST_1 and ST_2 are trees containing

$T_1 \cup S_1 \cup \{v\}$ and $T_2 \cup S_2 \cup \{v\}$ respectively, we have $w(E(ST_1)) + w(E(ST_2)) \geq st_G(T_1 \cup S_1 \cup \{v\}) + st_G(T_2 \cup \{v\}) = f_{i+1}(S_1 \cup \{v\}, T_1) + f_{i+1}(S_2 \cup \{v\}, T_2) \geq f_i(S, T')$. Since $V(ST_1) \cap V(ST_2) = \{v\}$ and $T' \cup S \subseteq V(ST_1) \cup V(ST_2)$, we have that $st_G(T' \cup S) = w(E(ST_1)) + w(E(ST_2))$. Thus $f_i(S, T') \leq st_G(T' \cup S)$.

Case 2: $i = 3$. By (21.3), there is $v \in V(G)$, $S_1, S_2, S_3 \in \binom{V(G)}{\leq 1}$, $S_1 \uplus S_2 \uplus S_3 = S$, and a partition $(T_1, T_2, T_3) \in \mathcal{P}_3(T')$ such that $f_3(S, T') = \sum_{r=1}^3 f_2(S_r \cup \{v\}, T_r)$. We have shown (in Case 1) that $f_2(S_r \cup \{v\}, T_r) = st_G(T_r \cup S_r \cup \{v\})$ for all $1 \leq r \leq 3$. Therefore $f_3(S, T') = \sum_{r=1}^3 st_G(T_r \cup S_r \cup \{v\})$. Let ST_r be an optimum Steiner tree for the set of terminals $T_r \cup S_r \cup \{v\}$ for all r . Note that $ST_1 + ST_2 + ST_3$ is a connected subgraph containing $T_1 \cup T_2 \cup T_3 \cup S$ and $w(E(ST_1 + ST_2 + ST_3)) \leq \sum_{r=1}^3 st_G(T_r \cup S_r \cup \{v\})$. Thus $st_G(T' \cup S) \leq w(E(ST_1 + ST_2 + ST_3)) \leq \sum_{r=1}^3 st_G(T_r \cup S_r \cup \{v\}) = f_3(S, T')$. Thus, $f_3(S, T') \geq st_G(T' \cup S)$.

Conversely, let ST be an optimum Steiner tree for the set of terminals $T' \cup S$. By Lemma 21.2, there is a vertex $v \in V(ST)$ such that each connected component C in $ST - v$ contains at most one vertex from S . Let be $\mathcal{C}_1, \mathcal{C}_2$ and \mathcal{C}_3 be a partition of connected components of $ST - v$ such that for each $|V(\mathcal{C}_r) \cap S| \leq 1$ for all $1 \leq r \leq 3$. For each r , let $T_r = T' \cap V(\mathcal{C}_r)$. If $v \in T'$, then we replace T_1 with $T_1 \cup \{v\}$. Note that (T_1, T_2, T_3) is a partition of T' . Hence $(T_1, T_2, T_3) \in \mathcal{P}_3(T')$. For each r , let $S_r = (S \setminus \{v\}) \cap V(\mathcal{C}_r)$. Since each \mathcal{C}_r contains at most one vertex from S , $|S_r| \leq 1$. This implies $\sum_{r=1}^3 f_2(S_r \cup \{v\}, T_r) \geq f_3(S, T')$. Note that $ST_r = ST[V(\mathcal{C}_r) \cup \{v\}]$ is a tree for each r . Since $V(\mathcal{C}_1) \cup V(\mathcal{C}_2) \cup V(\mathcal{C}_3) \cup \{v\} = V(ST)$ and for all $1 \leq r_1 \neq r_2 \leq 3$ it holds that $V(\mathcal{C}_{r_1}) \cap V(\mathcal{C}_{r_2}) = \{v\}$, we have $st_G(T' \cup S) = w(E(ST)) = \sum_{r=1}^3 w(E(ST_r)) \geq \sum_{r=1}^3 f_2(S_r \cup \{v\}, T_r) \geq f_3(S, T')$. Thus $f_3(S, T') \leq st_G(T' \cup S)$. \square

Our algorithm uses (21.2) and (21.3) to compute $f_0(\emptyset, T)$, which is exactly the cost of an optimum Steiner tree. A naïve way of turning the recurrences into an algorithm would be to simply make one recursive procedure for each f_i , and apply (21.2) and (21.3) directly. However, this would result in a factor $n^{O(\log k)}$ in the running time, which we seek to avoid. As the naïve approach, our algorithm has one recursive procedure F_i for each function f_i . The procedure F_i takes as input a subset T' of

Algorithm 4: Implementation of procedure F_i for $i \in \{0, 1, 2\}$

Input: $T' \subseteq T$
Output: $st_G(T' \cup S)$ for all $S \in \binom{V(G)}{\leq i}$

- 1 **if** $|T'| \leq 2$ **then**
- 2 **for** $S \in \binom{V(G)}{\leq 3}$ **do**
- 3 $A[S] \leftarrow st_G(T' \cup S)$ (compute using the Dreyfus–Wagner algorithm)
- 4 **return** A
- 5 **for** $S \in \binom{V(G)}{\leq i}$ **do**
- 6 $A[S] \leftarrow \infty$
- 7 **for** $T_1, T_2 \in \mathcal{B}_{1/6}(T')$ **do**
- 8 $A_1 \leftarrow F_{i+1}(T_1)$
- 9 $A_2 \leftarrow F_{i+1}(T_2)$
- 10 **for** $S_1 \uplus S_2 \in \binom{V(G)}{\leq i}$ such that $|S_2| \leq |S_1|$ and $v \in V(G)$ **do**
- 11 **if** $A[S_1 \uplus S_2] > A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}]$ **then**
- 12 $A[S_1 \uplus S_2] \leftarrow A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}]$
- 13 **return** A .

the terminal set, and returns an array that, for every $S \in \binom{V(G)}{\leq i}$, contains $f_i(S, T')$.

The key observation is that if we seek to compute $f_i(S, T')$ for a fixed T' and *all* choices of $S \in \binom{V(G)}{\leq i}$ using recurrence (21.2) or (21.3), we should not just iterate over every choice of S and then apply the recurrence to compute $f_i(S, T')$ because it is much faster to compute all the entries of the return array of F_i simultaneously, by iterating over every eligible partition of T , making the required calls to F_{i+1} (or F_{i-1} if we are using recurrence (21.3)), and updating the appropriate array entries to yield the return array of F_i . Next we give pseudocode for the procedures F_0, F_1, F_2, F_3 .

The procedure F_i for $0 \leq i \leq 2$ operates as follows. (See [algorithm 4](#).) Let $T' \subseteq T$ be the input to the procedure F_i . If $|T'| \leq 2$, then F_i computes $st_G(T' \cup S)$ for all $S \in \binom{V(G)}{\leq i}$ using the Dreyfus–Wagner algorithm and returns these values. The procedure F_i has an array A indexed by $S \in \binom{V(G)}{\leq i}$. At the end of the procedure F_i , $A[S]$ will contain the value $st_G(T' \cup S)$ for all $S \in \binom{V(G)}{\leq i}$. For each $(T_1, T_2) \in \mathcal{B}_{1/6}(T')$ ([line 7](#)), F_i calls $F_{i+1}(T_1)$ and $F_{i+1}(T_2)$ and it returns two sets of values $\{f_{i+1}(S, T_1) \mid S \in \binom{V(G)}{\leq i+1}\}$ and $\{f_i(S, T_2) \mid S \in \binom{V(G)}{\leq i}\}$, respectively. Let A_1 and A_2 be two arrays used to store the return values of $F_{i+1}(T_1)$ and $F_{i+1}(T_2)$ respectively. That is, $A_1[S] = f_{i+1}(S, T_1)$ for all $S \in \binom{V(G)}{\leq i+1}$ and $A_2[S'] = f_i(S', T_2)$ for all $S' \in \binom{V(G)}{\leq i}$. Now we update A as follows. For each $S_1 \uplus S_2 \in \binom{V(G)}{\leq i}$ and $v \in V(G)$ ([line 10](#)), if $A[S_1 \uplus S_2] > A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}]$, then we update the entry $A[S_1 \uplus S_2]$, with the value $A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}]$. So at the end the inner **for**

Algorithm 5: Implementation of procedure F_3

Input: $T' \subseteq T$
Output: $st_G(T' \cup S)$ for all $S \in \binom{V(G)}{\leq 3}$

```

1 if  $|T'| \leq 2$  then
2   for  $S \in \binom{V(G)}{\leq 3}$  do
3      $A[S] \leftarrow st_G(T' \cup S)$  (compute using the Dreyfus–Wagner algorithm)
4   return  $A$ 
5 for  $S \in \binom{V(G)}{\leq 3}$  do
6    $A[S] \leftarrow \infty$ 
7 for  $T_1, T_2, T_3 \in \mathcal{P}_3(T')$  do
8    $A_1 \leftarrow F_2(T_1)$ 
9    $A_2 \leftarrow F_2(T_2)$ 
10   $A_3 \leftarrow F_2(T_3)$ 
11  for  $S_1, S_2, S_3 \in \binom{V(G)}{\leq 1}$  and  $v \in V(G)$  do
12    if  $A[S_1 \cup S_2 \cup S_3] > A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}] + A_3[S_3 \cup \{v\}]$  then
13       $A[S_1 \cup S_2 \cup S_3] \leftarrow A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}] + A_3[S_3 \cup \{v\}]$ 
14 return  $A$ .
```

loop, $A[S]$ contains the value

$$\min_{\substack{v \in V(G) \\ S_1 \uplus S_2 = S}} f_{i+1}(S_1 \cup \{v\}, T_1) + f_i(S_2 \cup \{v\}, T_2).$$

Since we do have a outer **for** loop which runs over $(T_1, T_2) \in \mathcal{B}_{1/6}(T')$, we have updated $A[S]$ with

$$\min_{(T_1, T_2) \in \mathcal{B}_{1/6}(T')} \min_{\substack{v \in V(G) \\ S_1 \uplus S_2 = S}} f_{i+1}(S_1 \cup \{v\}, T_1) + f_i(S_2 \cup \{v\}, T_2).$$

at the end of the procedure. Then F_i will return A .

The procedure F_3 works as follows. (See [algorithm 5](#).) Let $T' \subseteq T$ be the input to the procedure F_3 . If $|T'| \leq 2$, then F_3 computes $st_G(T' \cup S)$ for all $S \in \binom{V(G)}{\leq 3}$ using the Dreyfus–Wagner algorithm and returns these values. The procedure F_3 has an array A indexed by $S \in \binom{V(G)}{\leq 3}$. At the end of the procedure F_3 , $A[S]$ will contain the value $st_G(T' \cup S)$ for all $S \in \binom{V(G)}{\leq 3}$. For each $(T_1, T_2, T_3) \in \mathcal{P}_3(T')$ ([line 7](#)), F_3 calls $F_2(T_1)$, $F_2(T_2)$ and $F_2(T_3)$, and it returns three sets of values $\{f_2(S, T_1) \mid S \in \binom{V(G)}{\leq 2}\}$, $\{f_2(S, T_2) \mid S \in \binom{V(G)}{\leq 2}\}$ and $\{f_2(S, T_3) \mid S \in \binom{V(G)}{\leq 2}\}$, respectively. Let A_1 , A_2 and A_3 be three arrays used to store the outputs of $F_2(T_1)$, $F_2(T_2)$ and $F_2(T_3)$ respectively. That is, $A_r[S] = f_2(S, T_r)$ for $r \in \{1, 2, 3\}$. Now we update A as follows. For each $S_1, S_2, S_3 \in \binom{V(G)}{\leq 1}$ and $v \in V(G)$ ([line 11](#)), if

$A[S_1 \cup S_2 \cup S_3] > A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}] + A_3[S_3 \cup \{v\}]$, then we update the entry $A[S_1 \cup S_2 \cup S_3]$, with the value $A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}] + A_3[S_3 \cup \{v\}]$. So at the end the inner **for** loop, $A[S]$ contains the value

$$\min_{\substack{S_1 \cup S_2 \cup S_3 = S \\ |S_1|, |S_2|, |S_3| \leq 1 \\ v \in V(G)}} \sum_{r=1}^3 f_2(S_r \cup \{v\}, T_r).$$

Since we do have a outer **for** loop which runs over $(T_1, T_2, T_3) \in \mathcal{P}_3(T')$, we have updated $A[S]$ with

$$\min_{(T_1, T_2, T_3) \in \mathcal{P}_3(T')} \min_{\substack{S_1 \cup S_2 \cup S_3 = S \\ |S_1|, |S_2|, |S_3| \leq 1 \\ v \in V(G)}} \sum_{r=1}^3 f_2(S_r \cup \{v\}, T_r).$$

at the end of the procedure. Then F_3 will return A as the output.

In what follows we prove the correctness and analyze the running time and memory usage of the call to the procedure $F_0(T)$.

Lemma 21.4. *For every $i \leq 3$, $T' \subseteq T$ the procedure $F_i(T')$ outputs an array that for every $S \in \binom{V(G)}{\leq i}$, contains $f_i(S, T')$.*

Proof. Correctness of Lemma 21.4 follows directly by an induction on $|T|$. Indeed, assuming that the lemma statement holds for the recursive calls made by the procedure F_i , it is easy to see that each entry of the output table is exactly equal to the right hand side of recurrence (21.2) (recurrence (21.3) in the case of F_3). \square

Observation 21.1. *The recursion tree of the procedure $F_0(T)$ has depth $\mathcal{O}(\log k)$.*

Proof. For every $i \leq 2$ the procedure $F_i(T')$ only makes recursive calls to $F_{i+1}(T'')$ where $|T''| \leq 2|T'|/3$. The procedure $F_3(T')$ makes recursive calls to $F_2(T'')$ where $|T''| \leq |T'|$. Therefore, on any root-leaf path in the recursion tree, the size of the considered terminal set T' drops by a constant factor every second step. When the terminal set reaches size at most 2, no further recursive calls are made. Thus any root-leaf path has length at most $\mathcal{O}(\log k)$. \square

Lemma 21.5. *The procedure $F_0(T)$ uses $\mathcal{O}(n^3 \log n W \log k)$ space.*

Proof. To upper bound the space used by the procedure $F_0(T)$ it is sufficient to upper bound the memory usage of every individual recursive call, not taking into account the memory used by its recursive calls, and then multiply this upper bound by the depth of the recursion tree.

Each individual recursive call will at any point of time keep a constant number of tables, each containing at most $\mathcal{O}(n^3)$ entries. Each entry is a number less than or equal to nW , therefore each entry can be represented using at most $\mathcal{O}(\log nW)$ bits. Thus each individual recursive call uses at most $\mathcal{O}(n^3 \log nW)$ bits. Combining this with Observation 21.1 proves the lemma. \square

Next we analyze the running time of the algorithm. Let $\tau_i(k)$ be the total number of arithmetic operations of the procedure $F_i(T')$ for all $i \leq 3$, where $k = |T'|$ on an n -vertex graph. It follows directly from the structure of the procedures F_i for $i \leq 2$, that there exists a constant C such that the following recurrences hold for τ_i , $i \leq 2$:

$$\begin{aligned} \tau_i(k) &\leq \sum_{\frac{k}{3} \leq j \leq \frac{2k}{3}} \binom{k}{j} (\tau_{i+1}(j) + \tau_{i+1}(k-j) + Cn^3) \\ &\leq 2 \sum_{\frac{k}{3} \leq j \leq \frac{2k}{3}} \binom{k}{j} (\tau_{i+1}(j) + Cn^3) \leq 2k \max_{\frac{k}{3} \leq j \leq \frac{2k}{3}} \binom{k}{j} (\tau_{i+1}(j) + Cn^3) \end{aligned} \quad (21.4)$$

Let $\binom{k}{i_1, i_2, i_3}$ be the number of partitions of k distinct elements into sets of sizes i_1 , i_2 , and i_3 . It follows directly from the structure of the procedure F_3 , that there exists a constant C such that the following recurrence holds for τ_3 :

$$\begin{aligned} \tau_3(k) &= \sum_{i_1+i_2+i_3=k} \binom{k}{i_1, i_2, i_3} (\tau_2(i_1) + \tau_2(i_2) + \tau_2(i_3) + Cn^4) \\ &\leq \sum_{i_1 \geq i_2, i_3} \binom{k}{i_1, i_2, i_3} 3 \cdot (\tau_2(i_1) + Cn^4) \leq 3 \sum_{i_1 \geq \frac{k}{3}} \binom{k}{i_1} 2^{k-i_1} \cdot (\tau_2(i_1) + Cn^4) \\ &\leq 3k \max_{i_1 \geq \frac{k}{3}} \binom{k}{i_1} 2^{k-i_1} \cdot (\tau_2(i_1) + Cn^4) \end{aligned} \quad (21.5)$$

Now we will bound $\tau_3(k)$ from above using (21.4) and (21.5). The following facts are required for the proof.

Fact 21.1. *By Stirling's approximation, $\binom{k}{\alpha k} \leq (\alpha^{-\alpha}(1-\alpha)^{(\alpha-1)})^k$ [112].*

Fact 21.2. For every fixed $x \geq 4$, function $f(y) = \frac{x^y}{y^y(1-y)^{1-y}}$ is increasing on interval $(0, 2/3]$.

Lemma 21.6. There exists a constant C such that $\tau_3(k) \leq C \cdot 11.7899^k n^4$

Proof. We prove that $\tau_2(k) \leq \hat{C}k^{(c \log k)}9.78977^k n^4$ and $\tau_3(k) \leq \hat{C}k^{(c \log k)}11.7898^k n^4$, by induction on k . We will pick \hat{C} to be a constant larger than the constants of (21.4) and (21.5), and sufficiently large so that the base case of the induction holds. We prove the inductive step. By the induction hypothesis and (21.4), we have that

$$\begin{aligned} \tau_2(k) &\leq 2k \max_{\frac{1}{3} \leq \alpha \leq \frac{2}{3}} \binom{k}{\alpha k} \left(\hat{C}(\alpha k)^{(c \log \alpha k)} 11.7898^{\alpha k} n^4 + \hat{C}n^3 \right) \\ &\leq 2k \left(\frac{11.7898^{2/3}}{(2/3)^{2/3}(1/3)^{1/3}} \right)^k \cdot \left(\hat{C} \left(\frac{2k}{3} \right)^{(c \log 2k/3)} n^4 + \hat{C}n^3 \right) \quad (\text{Fact 21.1, 21.2}) \\ &\leq (9.78977)^k \cdot 2k \cdot \left(\hat{C} \left(\frac{2k}{3} \right)^{(c \log 2k/3)} n^4 + \hat{C}n^3 \right) \\ &\leq 9.78977^k \cdot \hat{C}k^{(c \log k)} n^4 \end{aligned}$$

The last inequality holds if c is a sufficiently large constant (independent of k). By the induction hypothesis and (21.5), we have that

$$\begin{aligned} \tau_3(k) &\leq 3k \max_{\frac{1}{3} \leq \alpha \leq \frac{1}{2}} \binom{k}{\alpha k} 2^{(1-\alpha)k} \cdot \left(9.78977^{\alpha k} \cdot \hat{C}(\alpha k)^{(c \log \alpha k)} n^4 + \hat{C}n^4 \right) \\ &\leq 3k \max_{\frac{1}{3} \leq \alpha \leq \frac{1}{2}} (\alpha^{-\alpha}(1-\alpha)^{(\alpha-1)} 2^{(1-\alpha)} 9.78977^\alpha)^k \cdot \left(\hat{C}(\alpha k)^{(c \log \alpha k)} + \hat{C}n^4 \right) \\ &\leq 11.7898^k \cdot \hat{C}k^{(c \log k)} n^4 \end{aligned}$$

The last inequality holds for sufficiently large constants \hat{C} and c . For a sufficiently large constant C it holds that

$$C \cdot 11.7899^k n^4 \geq 11.7898^k \cdot \hat{C}k^{(c \log k)} n^4,$$

completing the proof. □

Lemma 21.7. For every $i \leq 2$ and constants C_{i+1} and $\beta_{i+1} \geq 4$ such that for

every $k \geq 1$ we have $\tau_{i+1}(k) \leq C_{i+1}\beta_{i+1}^k n^4$, there exists a constant C_i such that $\tau_i(k) \leq C_i \cdot 1.8899^k \cdot \beta_{i+1}^{2k/3} \cdot n^4$.

Proof. By (21.4) we have that

$$\begin{aligned} \tau_i(k) &\leq 2k \max_{\frac{k}{3} \leq i \leq \frac{2k}{3}} \binom{k}{j} (\tau_{i+1}(j) + Cn^3) \\ &\leq (2k + C) \max_{\frac{k}{3} \leq i \leq \frac{2k}{3}} \binom{k}{j} (C_{i+1}\beta_{i+1}^j n^4) \\ &\leq C_{i+1} \cdot (2k + C) \cdot \left(\frac{3}{2^{2/3}}\right)^k \cdot \beta_{i+1}^{2k/3} \cdot n^4 \\ &\leq C_i \cdot 1.8899^k \cdot \beta_{i+1}^{2k/3} \cdot n^4 \end{aligned}$$

The last inequality holds for a sufficiently large C_i depending on C_{i+1} and β_{i+1} but not on k . \square

Lemma 21.8. *The procedure $F_0(T)$ uses $\mathcal{O}(7.97^k n^4 \log nW)$ time.*

Proof. We show that $\tau_0(k) = \mathcal{O}(7.9631^k n^4)$. Since each arithmetic operation takes at most $\mathcal{O}(\log nW)$ time the lemma follows. Applying Lemma 21.7 on the upper bound for $\tau_3(k)$ from Lemma 21.6 proves that

$$\tau_2(k) = \mathcal{O}(1.8899^k \cdot 11.7899^{2k/3} n^4) = \mathcal{O}(9.790^k n^4).$$

Re-applying Lemma 21.7 on the above upper bound for $\tau_2(k)$ yields

$$\tau_1(k) = \mathcal{O}(1.8899^k \cdot 9.790^{2k/3} n^4) = \mathcal{O}(8.6489^k n^4).$$

Re-applying Lemma 21.7 on the above upper bound for $\tau_1(k)$ yields

$$\tau_0(k) = \mathcal{O}(1.8899^k \cdot 8.6489^{2k/3} n^4) = \mathcal{O}(7.9631^k n^4).$$

This completes the proof. \square

We are now in position to prove our main theorem.

Theorem 21.1. *STEINER TREE can be solved in time $\mathcal{O}(7.97^k n^4 \log nW)$ time using $\mathcal{O}(n^3 \log nW \log k)$ space.*

Proof. The algorithm calls the procedure $F_0(T)$ and returns the value stored for $f_0(\emptyset, T)$. By Lemma 21.4 the procedure $F_0(T)$ correctly computes $f_0(\emptyset, T)$, and by Lemma 21.3 this is exactly equal to the cost of the optimal Steiner tree. By Lemma 21.5 the space used by the algorithm is at most $\mathcal{O}(n^3 \log nW \log k)$, and by Lemma 21.8 the time used is $\mathcal{O}(7.97^k n^4 \log nW)$. \square

21.2.1 Obtaining better parameter dependence

The algorithm from Theorem 21.1 is based on defining and computing the functions f_i , $0 \leq i \leq 3$. The functions f_i , $i \leq 2$ are defined using recurrence (21.2), while the function f_3 is defined using recurrence (21.3). For every constant $t \geq 4$ we could obtain an algorithm for STEINER TREE by defining functions f_i , $0 \leq i \leq t - 1$ using (21.2) and f_t using (21.3). A proof identical to that of Lemma 21.3 shows that $f_i(S, T') = ST_G(S \cup T')$ for every $i \leq t$.

We can now compute $f_0(\emptyset, T)$ using an algorithm almost identical to the algorithm of Theorem 21.1, except that now we have $t + 1$ procedures, namely a procedure F_i for each $i \leq t$. For each i and terminal set $T' \subseteq T$ a call to the procedure $F_i(T')$ computes an array containing $f_i(S, T')$ for every set S of size at most i .

For $i < t$, the procedure F_i is based on (21.2) and is essentially the same as Algorithm 4. Further, the procedure F_t is based on (21.3) and is essentially the same as Algorithm 5. The correctness of the algorithm and an $\mathcal{O}(n^t \log(nW))$ upper bound on the space usage follows from arguments identical to Lemma 21.4 and Lemma 21.5 respectively.

For the running time bound, an argument identical to Lemma 21.6 shows that $\tau_t(k) = \mathcal{O}(11.7899^k n^{t+1})$. Furthermore, Lemma 21.7 now holds for $i \leq t - 1$. In the proof of Lemma 21.8 the bound for $\tau_0(k)$ is obtained by starting with the $\mathcal{O}(11.7899^k n^4)$ bound for τ_3 and applying Lemma 21.7 three times. Here we can upper bound $\tau_0(k)$ by starting with the $\mathcal{O}(11.7899^k n^{t+1})$ bound for τ_t and applying Lemma 21.7 t times. This yields a $C_0 \cdot \beta_0^k$ upper bound for $\tau_0(k)$, where

$$\beta_0 = (11.7899^{(2/3)^t}) 1.8899^{\sum_{i=0}^{t-1} (2/3)^i}$$

It is easy to see that as t tends to infinity, the upper bound for β_0 tends to a number between 6.75 and 6.751. This proves Theorem 21.2.

21.3 $4^{(1+\epsilon)k}n^{\mathcal{O}(f(\epsilon))}$ time algorithm

In this section, for any $\epsilon > 0$, we design a $4^{(1+\epsilon)k}n^{\mathcal{O}(f(\epsilon))} \log W$ time, $n^{\mathcal{O}(f'(\epsilon))} \log W$ space algorithm for STEINER TREE, where f and f' are computable functions depends only on ϵ . Towards that we need to explain SUBSET STEINER FOREST problem and show that the algorithm in Section 21.2 can be generalized to an algorithm for SUBSET STEINER FOREST. In Subsection 21.3.1 we explain SUBSET STEINER FOREST. Then in Subsection 21.3.2 we give faster polynomial space algorithm for STEINER TREE.

21.3.1 SUBSET STEINER FOREST

In this Subsection we generalize our parameterized single exponential time polynomial space algorithm (algorithm in Section 21.2) to a general version of the STEINER TREE problem, named SUBSET STEINER FOREST. Towards that we first define *subset Steiner forest* and then the problem SUBSET STEINER FOREST.

Definition 21.1. *Let G be a graph, $w : E(G) \rightarrow \{1, 2, \dots, W\}$ be a non-negative weight function, \mathcal{S} be a family of set of vertices and $T \subseteq V(G)$ be a set of terminals. A subgraph G' of G is called a subset Steiner forest of G on the family \mathcal{S} and the terminal set T , if the following conditions holds.*

- $T \cup (\bigcup_{S \in \mathcal{S}} S) \subseteq V(G')$, and
- for all $S \in \mathcal{S}$, there is a connected component C in G' such that $S \subseteq V(C)$.

We use $sf_G(\mathcal{S}, T)$ to denote the minimum weight of a subset Steiner forest of G on the family \mathcal{S} and the terminal set T .

The problem SUBSET STEINER FOREST is formally defined as,

<p>SUBSET STEINER FOREST</p> <p>Input: An undirected graph G, a non-negative weight function $w : E(G) \rightarrow \{1, 2, \dots, W\}$, a family \mathcal{S} of sets of vertices and a set of terminals T where $\bigcup_{S \in \mathcal{S}} S$ is a constant</p> <p>Question: A minimum weight subset Steiner forest of G on the family \mathcal{S} and the terminal set T</p>	<p>Parameter: $k = T$</p>
---	---

The recurrence relations defined for STEINER TREE (Equations 21.2 and 21.3) can be generalized to get recurrence relations for SUBSET STEINER FOREST. That is we

define four functions f_i for $i \in \{0, 1, 2, 3\}$. In what follows let $\mathcal{S} = \{S_1, \dots, S_r\}$ is a family of vertex sets such that for all j , $|S_j| \leq c$, where c and r are constants. Each function f_i takes as input a family \mathcal{S} of vertex subsets, of size r such that for all $S \in \mathcal{S}$, $|S| \leq c + i$ and a subset T' of T . These functions f_i s are recurrence relations for SUBSET STEINER FOREST. That is $f_i(\mathcal{S}, T')$ is exactly $sf_G(\mathcal{S}, T')$. Now we define these functions. When $|T'| \leq c + 3$, $f_i(\mathcal{S}, T') = sf_G(\mathcal{S}, T')$. For $|T'| > c + 3$, we define $f_i(\mathcal{S}, T')$ using the following recurrences.

Separation. For $i \in \{0, 1, 2\}$, let us define

$$f_i(\mathcal{S}, T') = \min_{\substack{(T_1, T_2) \in \mathcal{B}_1(T') \\ v_1, \dots, v_r \in V(G) \\ \forall j: S_j^{(1)} \uplus S_j^{(2)} = S_j}} \sum_{\ell=1}^2 f_{i+1}(\{S_1^{(\ell)} \cup \{v_1\}, \dots, S_r^{(\ell)} \cup \{v_r\}\}, T_\ell) \quad (21.6)$$

Resplitting. For $i = 3$, let us define

$$f_i(\mathcal{S}, T') = \min_{\substack{(T_1, T_2, T_3) \in \mathcal{P}_3(T') \\ \forall j: S_j^{(1)} \uplus S_j^{(2)} \uplus S_j^{(3)} = S_j \\ |S_j^{(1)}|, |S_j^{(2)}|, |S_j^{(3)}| \leq c+1 \\ v_1, \dots, v_r \in V(G)}} \sum_{\ell=1}^3 f_{i-1}(\{S_1^{(\ell)} \cup \{v_1\}, \dots, S_r^{(\ell)} \cup \{v_r\}\}, T_\ell) \quad (21.7)$$

The proof of correctness of above recurrence relation is exact generalization of the proof of correctness of the recurrence relations of STEINER TREE (Equations (21.2) and (21.3)). Note that when $|T'| \leq c + 3$, we can compute $sf_G(\mathcal{S}, T')$ in polynomial time by solving many instances of STEINER TREE using the Dreyfus–Wagner algorithm. As like in Section 21.2, a recursive algorithm using Equations (21.6) and (21.7) can be designed.

Theorem 21.4. SUBSET STEINER FOREST *can be solved in $\mathcal{O}(7.97^k n^{r(c+4)} \log nW)$ time using $n^{r(c+3)} \log nW \cdot \log k$ space.*

21.3.2 Algorithm for Steiner Tree

In this Subsection we design a faster polynomial space algorithm for STEINER TREE using SUBSET STEINER FOREST. We design a new recurrence relation for subset Steiner forest using a notion of α -Steiner separator for subset Steiner forest. Given a subset Steiner forest SF on terminals T and family \mathcal{S} , an α -Steiner separator S of SF is a subset of nodes which partitions $SF - S$ in two forests \mathcal{R}_1 and \mathcal{R}_2 , each one containing at most $\alpha|T|$ terminals from T . The following lemma follows from Lemma 1 (Sharp Separation) of [51].

Lemma 21.9 (*c*-Separation). *For any constant $c \geq 0$, every subset Steiner forest SF on terminal set T and family \mathcal{S} , has a $(\frac{1}{2} + \frac{1}{2^{c/2}})$ -Steiner separator S of size at most c .*

We design a recurrence relation $g^{(c)}$ for subset Steiner forest for any constant $c \geq 2$. The function $g^{(c)}$ takes inputs $S \subseteq V(G)$, $P_S \in \mathcal{P}(S)$ and a set of terminals T . When $|T| \leq c$, the value of $g^{(c)}(S, P_S, T)$ is defined to be $sf_G(P_S, T)$. Otherwise $g^{(c)}(S, P_S, T)$ is defined by the following recurrence relation. Let us fix $\alpha = \frac{1}{2^{(c/2)}}$.

$$g^{(c)}(S, P_S, T) = \min_{\substack{(T_1, T_2) \in \mathcal{B}_\alpha(T), \\ S' \supseteq S: |S' \setminus S| \leq c, \\ P_1, P_2 \in \mathcal{P}(S'), \\ P_S \preceq P_1 \sqcup P_2}} g^{(c)}(S', P_1, T_1) + g^{(c)}(S', P_2, T_2) \quad (21.8)$$

We need to show that Equation 21.8 is a recurrence relation for Subset Steiner forest and we prove it in Lemma 21.11. The following lemma is useful for proving Lemma 21.11.

Lemma 21.10. *Let G be a graph, $S', T \subseteq V(G)$ and $P_1, P_2 \in \mathcal{P}(S')$. Let F_1 and F_2 be subset Steiner forests for the pairs (P_1, T) and (P_2, T) respectively. Let $S \subseteq S'$ and $P_S \in \mathcal{P}(S)$ such that $P_S \preceq P_1 \sqcup P_2$. Then $F_1 + F_2$ is a subset Steiner forest for the pair (P_S, T) .*

Proof. Let $F = F_1 + F_2$. Since F_1 is a subset Steiner forest for the pair (P_1, T) and $P_1 \in \mathcal{P}(S)$, we have that $S' \cup T \subseteq V(F_1)$. This implies that $S \cup T \subseteq V(F)$. Now we need to show that for any $Q \in P_S$, there is a component C in F such that $Q \subseteq V(C)$. Since F_1 and F_2 are subset Steiner forests for the pairs (P_1, T) and (P_2, T) respectively, the graph $F = F_1 + F_2$ is a subset Steiner forest for the pair $(P_1 \sqcup P_2, T)$. Since $P_S \preceq P_1 \sqcup P_2$ any $Q \in P_S$ is completely contained in a block Q' in $P_1 \sqcup P_2$. Since F is a subset Steiner forest for the pair $(P_1 \sqcup P_2, T)$, there is a component C in F such that $Q' \subseteq V(C)$. This implies that $Q \subseteq V(C)$. This completes the proof of the Lemma. \square

Now we show that the above recurrence (Equation 21.8) is indeed a recurrence relation for subset Steiner forest:

Lemma 21.11. *For any $T \subseteq V(G)$, a partition P_S of vertex subset S of G and a constant $c \geq 2$ it holds that $g^{(c)}(S, P_S, T) = sf_G(P_S, T)$.*

Proof. We prove the lemma using induction on $|T|$. For the base case, when $|T| \leq c$ the lemma holds by the definition of $g^{(c)}$. For inductive step, let us assume that the lemma holds for all T' of size less than j and any $S' \subseteq V(G)$ and any partition $P_{S'} \in \mathcal{P}(S')$. We now show that $g^{(c)}(S, P_S, T) = sf_G(P_S, T)$ for all $T \subseteq \binom{V(G)}{j}$, $S \subseteq V(G)$ and $P_S \in \mathcal{P}(S)$. Fix a set $T \subseteq \binom{V(G)}{j}$, $S \subseteq V(G)$ and $P_S \in \mathcal{P}(S)$. Let $\alpha = \frac{1}{2^{(c/2)}}$.

First we show that $g^{(c)}(S, P_S, T) \geq sf_G(P_S, T)$. By (21.8), we know there exists $(T_1, T_2) \in \mathcal{B}_\alpha(T)$, $S' \supseteq S$ and $P_1, P_2 \in \mathcal{P}(S')$ such that $|S' \setminus S| \leq c$, $P_S \preceq P_1 \sqcup P_2$ and

$$g^{(c)}(S, P_S, T) = g^{(c)}(S', P_1, T_1) + g^{(c)}(S', P_2, T_2).$$

Since $(T_1, T_2) \in \mathcal{B}_\alpha(T)$ and $|T| \geq 2$, we have that $|T_1|, |T_2| < |T|$. Then by induction hypothesis $g^{(c)}(S', P_1, T_1) = sf_G(P_1, T_1)$ and $g^{(c)}(S', P_2, T_2) = sf_G(P_2, T_2)$. So we have that $g^{(c)}(S, P_S, T) = sf_G(P_1, T_1) + sf_G(P_2, T_2)$. Let SF_1 and SF_2 be optimum subset Steiner forests for the pairs (P_1, T_1) and (P_2, T_2) respectively. Hence, by Lemma 21.10, $G' = SF_1 + SF_2$ is a subset Steiner forest for the pair (P_S, T) . Thus we have shown that G' is subset Steiner forest of the pair (P_S, T) of weight $g^{(c)}(S, P_S, T)$. This implies that $g^{(c)}(S, P_S, T) \geq sf_G(P_S, T)$.

Conversely, let SF be an optimum subset Steiner forest for the pair (P_S, T) . By Lemma 21.9 we know that there exists an α -Steiner separator S'' of SF such that $|S''| \leq c$. Let $S' = S'' \cup S$. Since $S' \supseteq S''$, S' is also an α -Steiner separator of SF . Let \mathcal{R}_1 and \mathcal{R}_2 be the forests created by S' such that $|V(\mathcal{R}_1) \cap T| \leq (\frac{1}{2} + \alpha)|T|$ and $|V(\mathcal{R}_2) \cap T| \leq (\frac{1}{2} + \alpha)|T|$. Let $T_1 = V(\mathcal{R}_1) \cap T$ and $T_2 = V(\mathcal{R}_2) \cap T$. If $T_1 \cup T_2 \neq T$, then arbitrarily add each vertex in $T \setminus (T_1 \cup T_2)$ to either T_1 or T_2 such that $|T_r| \leq (\frac{1}{2} + \alpha)|T|$ for $r \in \{0, 1\}$. Note that $(T_1, T_2) \in \mathcal{B}_\alpha(T)$. Since S' is a separator for \mathcal{R}_1 and \mathcal{R}_2 in SF , there is no edge in $E(SF)$ which is incident

to both \mathcal{R}_1 and \mathcal{R}_2 . Let E_1 be the set of edges in $E(SF)$ which are incident to \mathcal{R}_1 and let $E_2 = E(SF) \setminus E_1$. Consider the graphs $F_1 = (V(\mathcal{R}_1) \cup S', E_1)$ and $F_2 = (V(\mathcal{R}_2) \cup S', E_2)$. The graphs F_1 and F_2 are subset Steiner forests for the pairs $(P_{F_1}[S'], T_1)$ and $(P_{F_2}[S'], T_2)$ respectively. Thus we have that

$$w(SF) = w(F_1) + w(F_2) \geq sf_G(P_{F_1}[S'], T_1) + sf_G(P_{F_2}[S'], T_2) \quad (21.9)$$

Since $F_1 + F_2 = SF$ and $P_S \preceq P_{SF}$, we have that $P_S \preceq P_{F_1}[S'] \sqcup P_{F_2}[S']$. We have that $|S' \setminus S| \leq c$, $(T_1, T_2) \in \mathcal{B}_\alpha(T)$ and $P_{F_1}[S'], P_{F_2}[S'] \in \mathcal{P}(S')$ such that $P_S \preceq P_{F_1}[S'] \sqcup P_{F_2}[S']$. Hence by induction hypothesis and our recurrence relation (Equation 21.8), we have that $sf_G(P_{F_1}[S'], T_1) + sf_G(P_{F_2}[S'], T_2) \geq g^{(c)}(S, P_S, T)$. Combining this with Equation 21.9, we get $g^{(c)}(S, P_S, T) \leq w(SF) = sf_G(P_S, T)$. \square

Now for any $\epsilon > 0$, we explain a $n^{\mathcal{O}(f'(\epsilon))} \log W$ space $4^{(1+\epsilon)k} n^{\mathcal{O}(f(\epsilon))} \log W$ time algorithm for STEINER TREE. We fix (later) two constants $c \geq 4$ and d based on ϵ . Let $\alpha = \frac{1}{2^{(c/2)}}$ and $\beta = (\frac{1}{2} + \alpha)^d$. The algorithm is a recursive algorithm based on Equation 21.8. Whenever the cardinality of set of terminal in a recursive call is bounded by βk , then the algorithm uses Theorem 21.4 as a black box, otherwise it branches according to Equation 21.8. Initial call to the recurrence is on the set of terminals T and family \emptyset . Since each time Equation 21.8 reduces the cardinality of the set of terminals by a factor of $(\frac{1}{2} + \alpha)$, the depth of the recurrence tree is bounded by d . Hence the total number of vertices in the family when our algorithm invoke Theorem 21.4 is bounded by $d \cdot c$. This implies that the space usage of our algorithm is bounded by $(n^{dc(dc+1)} + d) \log W$.

Now we bound the running time of the algorithm. Let $T(k)$ be the running time of the algorithm for an n -vertex graph.

Lemma 21.12. *There exists a constant C such that for any $k' \leq k$,*

$$T(k') = C \cdot (dc)^{(2dc)d'} n^{cd'} n^{cd(cd+4)} 2^{d'} \log nW \cdot (7.97)^{\beta k} 4^{k'} 2^{\frac{2k'}{1-2\alpha}},$$

where $d' = \log_{\frac{1}{2} + \alpha} \frac{\beta k}{k'}$.

Proof. Let C be a constant such that the algorithm in Theorem 21.4 runs in time $C \cdot 7.97^k n^{r(c+4)} \log nW$. We prove the lemma by induction on k' . When $k' \leq \beta k$,

then $T(k')$ is bounded by $C \cdot 7.97^{k'} n^{dc(dc+4)} \log nW$. Assume that the lemma holds for all values of $k' < k$. Now we need to bound $T(k)$. According to Equation 21.8,

$$T(k) = n^c c^{2c} 2^k \cdot 2 \cdot T\left(\left(\frac{1}{2} + \alpha\right)k\right) \quad (21.10)$$

Let $\gamma = \left(\frac{1}{2} + \alpha\right)$. Now by induction hypothesis, we simplify Equation 21.10 as

$$\begin{aligned} T(k) &= n^c c^{2c} 2^k \cdot 2 \cdot T(\gamma k) \\ &= n^c c^{2c} 2^{k+1} \cdot C (dc)^{(2dc)d'} n^{cd'} n^{cd(cd+4)} 2^{d'} \log nW \cdot (7.97)^{\beta k} 4^{\gamma k} 2^{\frac{2\gamma k}{1-2\alpha}} \end{aligned} \quad (21.11)$$

where $d' = \log_{\gamma} \frac{\beta k}{\gamma k} \leq d - 1$.

Substituting $d' = d - 1$ in Equation 21.11,

$$T(k) = C \cdot (dc)^{(2dc)d} n^{cd} n^{cd(cd+4)} 2^d \log nW \cdot 2^k (7.97)^{\beta k} 4^{\gamma k} 2^{\frac{2\gamma k}{1-2\alpha}} \quad (21.12)$$

Claim 21.1. $2^k 4^{\gamma k} 2^{\frac{2\gamma k}{1-2\alpha}} \leq 4^k 2^{\frac{2k}{1-2\alpha}}$

Proof.

$$\begin{aligned} 2^k 4^{\gamma k} 2^{\frac{2\gamma k}{1-2\alpha}} &\leq 2^k \cdot 2^{(1+2\alpha)k} \cdot 2^{\frac{(1+2\alpha)k}{1-2\alpha}} \\ &\leq 4^k \cdot 2^{\left(2\alpha + \frac{1+2\alpha}{1-2\alpha}\right)k} \end{aligned} \quad (21.13)$$

Here $\alpha = \frac{1}{2(c/2)}$. For $c \geq 4$, $\left(2\alpha + \frac{1+2\alpha}{1-2\alpha}\right) \leq \frac{2}{1-2\alpha}$. This completes the proof of the claim. \square

By applying above claim in Equation 21.12, we get

$$T(k) = C \cdot (dc)^{(2dc)d} n^{cd} n^{cd(cd+4)} 2^d \log nW \cdot (7.97)^{\beta k} 4^k 2^{\frac{2k}{1-2\alpha}}$$

\square

Thus, for any $\epsilon > 0$, by choosing constants c and d such that $(7.97)^{\beta k} 2^{\frac{2k}{1-2\alpha}} \leq 4^{\epsilon k}$,

we can get the following theorem.

Theorem 21.3. *For any $\epsilon > 0$, there is a $n^{\mathcal{O}(f'(\epsilon))} \log W$ space $4^{(1+\epsilon)k} n^{\mathcal{O}(f(\epsilon))} \log W$ time algorithm for STEINER TREE, where f and f' are computable functions depends only on ϵ .*

Part VI

Conclusion

Chapter 22

Conclusion

We gave an efficient algorithm for computing a representative family of a family of independent sets in a linear matroid. For the special case where the underlying matroid is uniform we developed an even faster algorithm. Also we have developed faster algorithm for representative family of product families both in uniform matroids and linear matroids. We also showed interesting links between representative families of matroids and the design of single-exponential parameterized and exact exponential algorithms. We believe that these connections have a potential for a wide range of applications. We list some of the natural open problems below.

- What is the best possible running time of an algorithm that computes a q -representative family of size at most $\binom{p+q}{p}$ for a p -family \mathcal{F} of independent sets of a linear matroid? Does an algorithm with linear dependence of the running time on $|\mathcal{F}|$ exist, or is it possible to prove superlinear lower bounds?
- It would be interesting to find faster algorithms even for special classes of linear matroids. Uniform matroids and graphic matroids are especially interesting in this regard.
- There is a randomized polynomial time algorithm to find a linear representation of a gammoid. Can we derandomize it? Can we compute a small representative family of a family of independent sets in a gammoid without using its linear representation?
- What are the natural set families like product families for which we can find representative sets faster?
- Are there deterministic algorithms for k -PATH and MULTILINEAR MONOMIAL DETECTION running in time $2^k n^{\mathcal{O}(1)}$?

We would like to remark that recently Zehavi [121] has announced a further improvement for k -PATH algorithm using representative families and divide and color

technique. The algorithm presented in [121] runs in time $2.597^k \cdot n^{\mathcal{O}(1)}$.

In Part IV we studied some problems on matroids like MATROID GIRTH and MATROID CONNECTIVITY. We showed that MATROID GIRTH and MATROID CONNECTIVITY in a linear matroid when parameterized by $\text{rank}(M)$ are not FPT unless $\text{FPT} = W[1]$, but FPT when parameterized by $\text{rank}(M)+q$, where q is the field size. Other than the EVEN SET problem which remains notoriously open, we draw attention to the following interesting open problem arising from our work. Is MATROID GIRTH FPT on gammoids when parameterized by $\text{rank}(M)$?

In Part V we give the first single exponential time polynomial space algorithm for STEINER TREE running in time $\mathcal{O}(7.97^k n^4 \log nW)$ where W is the largest weight of an edge in the n -vertex input graph. We also give an algorithm running in time $4^{(1+\epsilon)k} n^{\mathcal{O}(f(\epsilon))} \log W$ and space $n^{\mathcal{O}(f'(\epsilon))}$. Can we get rid of the ϵ dependence from the power of n in the time complexity or space complexity?

Bibliography

- [1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004. [23](#)
- [2] J. Akiyama and M. Kano. Factors and factorizations of graphs - a survey. *Journal of Graph Theory*, 9(1):1–42, 1985. [139](#)
- [3] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986. [22](#)
- [4] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. Assoc. Comput. Mach.*, 42(4):844–856, 1995. [24](#), [34](#), [85](#), [86](#), [97](#), [117](#), [118](#)
- [5] O. Amini, F. V. Fomin, and S. Saurabh. Counting subgraphs via homomorphisms. *SIAM J. Discrete Math.*, 26(2):695–717, 2012. [86](#)
- [6] R. P. Anstee. An algorithmic proof of Tutte’s f -factor theorem. *J. Algorithms*, 6(1):112–131, 1985. [139](#)
- [7] J. Bang-Jensen and G. Gutin. *Digraphs*. Springer Monographs in Mathematics. Springer-Verlag London Ltd., London, second edition, 2009. Theory, algorithms and applications. [12](#), [131](#), [132](#)
- [8] M. Basavaraju, F. V. Fomin, P. A. Golovach, P. Misra, M. S. Ramanujan, and S. Saurabh. Parameterized algorithms to preserve connectivity. In *Proceedings of the 41st International Colloquium of Automata, Languages and Programming (ICALP)*, volume 8572 of *Lecture Notes in Comput. Sci.*, pages 800–811. Springer, 2014. [244](#)
- [9] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, Jan. 1962. [7](#)
- [10] R. Bellman and W. Karush. Mathematical programming and the maximum transform. *Journal of the Society for Industrial and Applied Mathematics*, 10(3):pp. 550–567, 1962. [109](#)
- [11] R. Bellman and W. Karush. On the maximum transform and semigroups of transformations. *Bull. Amer. Math. Soc.*, 68(5):516–518, 09 1962. [109](#)

- [12] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978. [225](#)
- [13] M. W. Bern and P. E. Plassmann. The Steiner problem with edge lengths 1 and 2. *Inf. Process. Lett.*, 32(4):171–176, 1989. [243](#)
- [14] A. Bhattacharyya, P. Indyk, D. P. Woodruff, and N. Xie. The complexity of linear dependence problems in vector spaces. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 496–508, 2011. [226](#)
- [15] R. E. Bixby and W. H. Cunningham. Matroid optimization and algorithms. In R. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of combinatorics*, volume 1, pages 550–609. MIT Press, Cambridge, MA, 1996. [233](#)
- [16] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: Fast subset convolution. In *Proceedings of the 39th annual ACM Symposium on Theory of Computing (STOC 2007)*, page to appear, New York, 2007. ACM Press. [109](#), [243](#), [244](#)
- [17] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Narrow sieves for parameterized paths and packings. *CoRR*, abs/1007.1161, 2010. [85](#), [86](#), [97](#), [98](#), [118](#)
- [18] A. Björklund, T. Husfeldt, and S. Khanna. Approximating longest directed paths and cycles. In *Proceedings of the 31st International Colloquium, Automata, Languages and Programming (ICALP 2004)*, volume 3142 of *Lecture Notes in Comput. Sci.*, pages 222–233. Springer, 2004. [73](#)
- [19] A. Björklund, P. Kaski, and L. Kowalik. Probably optimal graph motifs. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, pages 20–31, 2013. [118](#)
- [20] G. E. Blelloch, K. Dhamdhere, E. Halperin, R. Ravi, R. Schwartz, and S. Sridhar. Fixed parameter tractability of binary near-perfect phylogenetic tree reconstruction. In *Proceedings of the 33rd International Automata, Languages and Programming, Colloquium (ICALP 2006)*, volume 4051 of *Lecture Notes in Computer Science*, pages 667–678. Springer, 2006. [244](#)
- [21] H. L. Bodlaender. On linear time minor tests with depth-first search. *J. Algorithms*, 14(1):1–23, 1993. [85](#), [88](#)
- [22] H. L. Bodlaender. A partial k -arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998. [247](#)
- [23] H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In *Automata, Languages, and Programming - 40th International*

Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I, pages 196–207, 2013. [36](#), [37](#), [128](#), [201](#)

- [24] B. Bollobás. On generalized graphs. *Acta Math. Acad. Sci. Hungar.*, 16:447–452, 1965. [35](#)
- [25] J. Bunch and J. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974. [128](#)
- [26] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6, 2013. [243](#)
- [27] J. Chen, Q. Feng, Y. Liu, S. Lu, and J. Wang. Improved deterministic algorithms for weighted matching and packing problems. *Theor. Comput. Sci.*, 412(23):2503–2512, 2011. [98](#)
- [28] J. Chen, D. Friesen, W. Jia, and I. Kanj. Using nondeterminism to design efficient deterministic algorithms. *Algorithmica*, 40(2):83–97, 2004. [98](#)
- [29] J. Chen, X. Huang, I. A. Kanj, and G. Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8):1346–1367, 2006. [7](#)
- [30] J. Chen, J. Kneis, S. Lu, D. Mölle, S. Richter, P. Rossmanith, S.-H. Sze, and F. Zhang. Randomized divide-and-conquer: improved path, matching, and packing algorithms. *SIAM J. Comput.*, 38(6):2526–2547, 2009. [85](#), [86](#)
- [31] J. Chen, J. Kneis, S. Lu, D. Molle, S. Richter, P. Rossmanith, S. H. Sze, and F. Zhang. Randomized divide-and-conquer: Improved path, matching, and packing algorithms. *SIAM J. on Computing*, 38(6):2526–2547, 2009. [98](#)
- [32] J. Chen, Y. Liu, S. Lu, S.-H. Sze, and F. Zhang. Iterative expansion and color coding: An improved algorithm for 3D-matching. *ACM Transactions on Algorithms*, 8(1):6, 2012. [98](#), [99](#)
- [33] J. Chen, S. Lu, S.-H. Sze, and F. Zhang. Improved algorithms for path, matching, and packing problems. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007)*, pages 298–307. SIAM, 2007. [86](#)
- [34] S. Chen and Z. Chen. Faster deterministic algorithms for packing, matching and t-dominating set problems. *CoRR*, abs/1306.3602, 2013. [98](#)
- [35] N. Cohen, F. V. Fomin, G. Gutin, E. J. Kim, S. Saurabh, and A. Yeo. Algorithm for finding k -vertex out-trees and its application to k -internal out-branching problem. *J. Comput. System Sci.*, 76(7):650–662, 2010. [86](#)
- [36] T. H. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., second edition, 2001. [3](#), [131](#)
- [37] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015. [7](#)

- [38] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS 2011)*. IEEE, 2011. [201](#)
- [39] R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, third edition, 2005. [12](#)
- [40] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999. [272](#)
- [41] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. [7](#), [189](#), [226](#), [243](#)
- [42] R. G. Downey, M. R. Fellows, and M. Koblitz. Techniques for exponential parameterized reductions in vertex set problems. (Unpublished, reported in [40], §8.3). [97](#), [98](#)
- [43] R. G. Downey, M. R. Fellows, A. Vardy, and G. Whittle. The parametrized complexity of some fundamental problems in coding theory. *SIAM J. Comput.*, 29(2):545–570, 1999. [226](#)
- [44] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971. [243](#), [244](#)
- [45] J. Edmonds. Optimum branchings. *J. Res. Nat. Bur. Standards Sect. B*, 71B:233–240, 1967. [137](#)
- [46] M. R. Fellows, C. Knauer, N. Nishimura, P. Ragde, F. A. Rosamond, U. Stege, D. M. Thilikos, and S. Whitesides. Faster fixed-parameter tractable algorithms for matching and packing problems. *Algorithmica*, 52(2):167–176, 2008. [97](#), [98](#)
- [47] H. Fleischner. *Eulerian Graphs and Related Topics, Part 1, Volume 1*. Annals of Discrete Mathematics 45. Amsterdam, 1990. [139](#)
- [48] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006. [7](#)
- [49] F. V. Fomin, P. Golovach, F. Panolan, and S. Saurabh. Editing to connected f -degree graph. *To appear in STACS 2016*. [10](#)
- [50] F. V. Fomin, F. Grandoni, D. Kratsch, D. Lokshtanov, and S. Saurabh. Computing optimal Steiner trees in polynomial space. *Algorithmica*, 65(3):584–604, 2013. [244](#), [247](#)
- [51] F. V. Fomin, F. Grandoni, D. Lokshtanov, and S. Saurabh. Sharp separation and applications to exact and parameterized algorithms. *Algorithmica*, 63(3):692–706, 2012. [259](#)

- [52] F. V. Fomin, P. Kaski, D. Lokshtanov, F. Panolan, and S. Saurabh. Parameterized single-exponential time polynomial space algorithm for steiner tree. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 494–505, 2015. [10](#)
- [53] F. V. Fomin and D. Kratsch. *Exact exponential algorithms*. Springer, 2011. [7](#)
- [54] F. V. Fomin, D. Lokshtanov, F. Panolan, and S. Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. *Submitted*. [9](#), [10](#)
- [55] F. V. Fomin, D. Lokshtanov, F. Panolan, and S. Saurabh. Representative sets of product families. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737, pages 443–454, 2014. [9](#), [10](#)
- [56] F. V. Fomin, D. Lokshtanov, V. Raman, S. Saurabh, and B. V. R. Rao. Faster algorithms for finding and counting subgraphs. *J. Comput. System Sci.*, 78(3):698–706, 2012. [86](#), [118](#)
- [57] F. V. Fomin, D. Lokshtanov, and S. Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 142–151, 2014. [9](#), [42](#), [85](#)
- [58] P. Frankl. An extremal problem for two families of sets. *European J. Combin.*, 3(2):125–127, 1982. [35](#)
- [59] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $0(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984. [21](#), [22](#), [23](#)
- [60] B. Fuchs, W. Kern, D. Mölle, S. Richter, P. Rossmanith, and X. Wang. Dynamic programming for minimum Steiner trees. *Theory of Computing Systems*, 41(3):493–500, 2007. [244](#)
- [61] H. N. Gabow and S. Nie. Finding a long directed cycle. *ACM Transactions on Algorithms*, 4(1), 2008. [73](#)
- [62] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979. [97](#)
- [63] S. Gaspers, E. J. Kim, S. Ordyniak, S. Saurabh, and S. Szeider. Don’t be strict in local search! In *AAAI*, 2012. [226](#)
- [64] P. A. Golovach. Editing to a connected graph of given degrees. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 8635 of *Lecture Notes in Comput. Sci.*, pages 324–335. Springer, 2014. [141](#)

- [65] P. Goyal, N. Misra, and F. Panolan. Faster deterministic algorithms for r -dimensional matching using representative sets. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, volume 24, pages 237–248, 2013. [9](#), [98](#), [99](#)
- [66] P. Goyal, N. Misra, F. Panolan, and M. Zehavi. Deterministic algorithms for matching and packing problems based on representative sets. *SIAM Journal on Discrete Mathematics*, 29(4):1815–1836, 2015. [9](#)
- [67] S. Guillemot and F. Sikora. Finding and counting vertex-colored subtrees. *Algorithmica*, 65(4):828–844, 2013. [118](#)
- [68] J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of generalized vertex cover problems. In *Proceedings of the 9th International Workshop Algorithms and Data Structures (WADS 2005)*, volume 3608 of *Lecture Notes in Computer Science*, pages 36–48. Springer, 2005. [244](#)
- [69] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 1961 16th ACM National Meeting*, ACM '61, pages 71.201–71.204, New York, NY, USA, 1961. ACM. [7](#)
- [70] H. T. Hsu. An algorithm for finding a minimal equivalent graph of a digraph. *J. Assoc. Comput. Mach.*, 22:11–16, 1975. [131](#)
- [71] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity. *Journal of Computer and System Sciences*, 63(4):512–530, 2001. [73](#)
- [72] S. Jukna. *Extremal combinatorics*. Springer Verlag Berlin Heidelberg, 2011. [42](#)
- [73] L. Khachiyan. On the complexity of approximating extremal determinants in matrices. *Journal of Complexity*, 11(1):138 – 153, 1995. [226](#), [234](#)
- [74] T. Kirkman. On a problem in combinatorics. *Cambridge and Dublin Math. J.*, 2:191–204, 1847. [139](#)
- [75] J. Kleinberg and E. Tardos. *Algorithm design*. Addison-Wesley, 2005. [3](#)
- [76] T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. [202](#)
- [77] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. Divide-and-color. In *Proceedings of the 34th International Workshop Graph-Theoretic Concepts in Computer Science (WG 2008)*, volume 4271 of *Lecture Notes in Computer Science*, pages 58–67. Springer, 2008. [85](#)
- [78] M. Kouider and P. D. Vestergaard. Connected factors in graphs - a survey. *Graphs and Combinatorics*, 21(1):1–26, 2005. [139](#)

- [79] I. Koutis. A faster parameterized algorithm for set packing. *Information Processing Letters*, 94:7–9, 2005. [97](#), [98](#)
- [80] I. Koutis. Faster algebraic algorithms for path and packing problems. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586, 2008. [85](#), [97](#), [98](#), [117](#), [118](#)
- [81] I. Koutis. Constrained multilinear detection for faster functional motif discovery. *Information Processing Letters*, 112(22):889 – 892, 2012. [117](#), [118](#)
- [82] I. Koutis and R. Williams. Limits and applications of group algebras for parameterized problems. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, volume 5555 of *Lecture Notes in Computer Sci.*, pages 653–664. Springer, 2009. [86](#), [97](#), [98](#), [117](#), [118](#)
- [83] Y. Liu, J. Chen, and J. Wang. Parameterized algorithms for weighted matching and packing problems. In *Theory and Applications of Models of Computation, 4th International Conference, TAMC 2007, Shanghai, China, May 22-25, 2007, Proceedings*, pages 692–702, 2007. [98](#)
- [84] Y. Liu, S. Lu, J. Chen, and S.-H. Sze. Greedy localization and color-coding: improved matching and packing algorithms. In *International Workshop on Parameterized and Exact Computation IWPEC*, volume 4169 of *LNCS*, pages 84–95. Springer, 2006. [99](#)
- [85] D. Lokshtanov, P. Misra, F. Panolan, and S. Saurabh. Deterministic truncation of linear matroids. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 922–934, 2015. [18](#), [128](#), [129](#)
- [86] D. Lokshtanov and J. Nederlof. Saving space by algebraization. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 321–330. ACM, 2010. [244](#)
- [87] L. Lovász. Flats in matroids and geometric graphs. In *In Combinatorial surveys (Proc. Sixth British Combinatorial Conf., Royal Holloway Coll., Egham)*, pages 45–86. Academic Press, London, 1977. [35](#), [36](#), [124](#)
- [88] L. Lovász and M. D. Plummer. *Matching theory*. AMS Chelsea Publishing, Providence, RI, 2009. Corrected reprint of the 1986 original [MR0859549]. [139](#)
- [89] J. Macwilliams. A theorem on the distribution of weights in a systematic code. *Bell System Technical Journal*, 42(1):79–94, 1963. [228](#)
- [90] S. Martello. An algorithm for finding a minimal equivalent graph of a strongly connected digraph. *Computing*, 21(3):183–194, 1978/79. [131](#)
- [91] S. Martello and P. Toth. Finding a minimum equivalent graph of a digraph. *Networks*, 12(2):89–100, 1982. [131](#)

- [92] D. Marx. Parameterized coloring problems on chordal graphs. *Theor. Comput. Sci.*, 351(3):407–424, 2006. [34](#), [35](#)
- [93] D. Marx. A parameterized view on matroid optimization problems. *Theor. Comput. Sci.*, 410(44):4471–4479, 2009. [17](#), [19](#), [35](#), [125](#)
- [94] L. Mathieson and S. Szeider. Editing graphs to satisfy degree constraints: A parameterized approach. *J. Comput. Syst. Sci.*, 78(1):179–191, 2012. [140](#)
- [95] S. T. McCormick. *A Combinatorial Approach to Some Sparse Matrix Problems*. PhD thesis, Stanford University, CA Systems Optimization, Lab, 1983. [225](#)
- [96] M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005. [8](#), [49](#)
- [97] B. Monien. How to find long paths efficiently. In *Analysis and design of algorithms for combinatorial problems (Udine, 1982)*, volume 109 of *North-Holland Math. Stud.*, pages 239–254. North-Holland, Amsterdam, 1985. [34](#), [35](#), [85](#)
- [98] D. M. Moyses and G. L. Thompson. An algorithm for finding a minimum equivalent graph of a digraph. *J. ACM*, 16(3):455–460, July 1969. [131](#), [132](#)
- [99] K. Murota. *Matrices and matroids for systems analysis*, volume 20. Springer, 2000. [125](#)
- [100] M. Naor, L. J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS 1995)*, pages 182–191. IEEE, 1995. [21](#), [26](#), [27](#), [29](#), [30](#), [42](#)
- [101] J. Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica*, 65(4):868–884, 2013. [243](#), [244](#)
- [102] J. G. Oxley. On a matroid generalization of graph connectivity. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 90, pages 207–214. Cambridge Univ Press, 1981. [233](#)
- [103] J. G. Oxley. *Matroid theory*, volume 3. Oxford University Press, 2006. [15](#), [16](#), [17](#), [19](#)
- [104] F. Panolan, M. S. Ramanujan, and S. Saurabh. On the parameterized complexity of girth and connectivity problems on linear matroids. In *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, pages 566–577, 2015. [10](#)
- [105] C. H. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of the V-C dimension. *J. Comput. Syst. Sci.*, 53(2):161–170, 1996. [85](#)

- [106] J. Petersen. Die Theorie der regulären graphs. *Acta Math.*, 15(1):193–220, 1891. [139](#)
- [107] M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E. J. van Leeuwen. Subexponential-time parameterized algorithm for Steiner tree on planar graphs. In *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 353–364, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. [243](#)
- [108] M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E. J. van Leeuwen. Network sparsification for Steiner problems on planar and bounded-genus graphs. In *Proceedings of the 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–285. IEEE, 2014. [243](#)
- [109] J. Plehn and B. Voigt. Finding minimally weighted subgraphs. In *Proceedings of the 16th Workshop on Graph-Theoretic Concepts in Computer Science (WG 1991)*, volume 484 of *Lecture Notes in Comput. Sci.*, pages 18–29. Springer, 1991. [85](#)
- [110] M. D. Plummer. Graph factors and factorization: 1985-2003: A survey. *Discrete Mathematics*, 307(7-8):791–821, 2007. [139](#)
- [111] H. J. Prömel and A. Steger. *The Steiner tree problem*. Advanced Lectures in Mathematics. Friedr. Vieweg & Sohn, Braunschweig, 2002. [243](#)
- [112] H. Robbins. A remark on Stirling’s formula. *Amer. Math. Monthly*, 62:26–29, 1955. [189](#), [254](#)
- [113] J. P. Schmidt and A. Siegel. The spatial complexity of oblivious k-probe hash functions. *SIAM J. Comput.*, 19(5):775–786, 1990. [21](#)
- [114] H. Shachnai and M. Zehavi. Representative families: A unified tradeoff-based approach. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737, pages 786–797, 2014. [42](#)
- [115] W. Tutte. Connectivity in matroids. *Canad. J. Math*, 18:1301–1324, 1966. [233](#), [235](#)
- [116] A. Vardy. The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, 43(6):1757–1766, 1997. [225](#)
- [117] J. Wang and Q. Feng. Improved parameterized algorithms for weighted 3-set packing. In *Proc. COCOON*, pages 130–139, 2008. [99](#)
- [118] J. Wang and Q. Feng. An $O^*(3.52^{3k})$ parameterized algorithm for 3-set packing. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation, TAMC’08*, pages 82–93, Berlin, Heidelberg, 2008. Springer-Verlag. [99](#)

- [119] R. Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009. [85](#), [86](#), [117](#), [118](#)
- [120] V. V. Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC 2012)*, pages 887–898. ACM, 2012. [11](#)
- [121] M. Zehavi. Mixing color coding-related techniques. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 1037–1049, 2015. [267](#), [268](#)