PARAMETERISED ALGORITHMS FOR COUNTING SUBGRAPHS, MATCHINGS, AND MONOCHROMATIC PARTITIONS

JESSICA LAURETTE RYAN

SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
Doctor of Philosophy

SCHOOL OF COMPUTING SCIENCE

COLLEGE OF SCIENCE AND ENGINEERING
UNIVERSITY OF GLASGOW

Abstract

Counting the number of solutions to a computational problem is at least as hard as deciding whether a solution exists. In fact, it is often much harder. However, in theory, as well as in practice, it is often of interest to determine the number of solutions to computational problems. In this thesis, we take advantage of the structure of some hard computational counting problems to develop efficient parameterised algorithms for the kinds of problem instances which we expect to see in practice.

The subgraph counting problem asks for the number of times that a "pattern graph" appears as a subgraph of a larger "host" graph. The subgraph counting problem is computationally hard for general pairs of host and pattern graphs. Our first result describes an efficient algorithm for counting small subgraphs in host graphs with a bounded number of high-degree vertices. Our work is motivated by practical applications of subgraph counting which involve counting copies of small pattern graphs in large host graphs with this structure.

Stable matching problems arise when we wish to match together a set of agents in such a way that no pair of agents would mutually prefer to deviate from the assignment. The problem of counting stable matchings is computationally hard even in the most basic stable marriage setting where agents' preference lists are strict and complete. Here, we study stable matching problems in the setting where agents belong to groups of similar agents called "types". We describe efficient parameterised algorithms for counting stable matchings in a number of different settings parameterised by the number of agent types.

Our final result concerns the problem of partitioning a large edge-coloured host graph into a small number of monochromatic subgraphs. Monochromatic partitioning problems are well-studied for specific classes of host graphs. Here, we consider the complexity of monochromatic partitioning problems for more general classes of host graphs. Specifically, we provide an efficient algorithm for counting partitions of edge-coloured graphs which are "tree-like" into monochromatic paths.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Kitty Meeks, whose vast knowledge and consistent encouragement and guidance have been invaluable throughout my PhD. I would also like to thank my second supervisor, Prof. David Manlove, for his support and helpful discussions on my research. I am extremely grateful to my viva examiners, Prof. Puck Romback and Dr Andrew Elliot, and my viva convenor Dr Kevin Bryson, for their thoughtful questions on my thesis, and for making the viva an enjoyable experience. Thank you also to Prof. David Manlove, Prof. Alice Miller and Dr. Angelos Marnerides for taking the time to conduct my annual progress reviews.

I am grateful to have been part of such a friendly and motivated research group. I would like to extend a special thanks to William for his friendship, and for sharing his wisdom on graph theory and algorithms with me, and also to Ciaran for sharing his expertise in search algorithms. Thank you to my officemates Blair and Simon for making the office such an enjoyable place to work. Thank you also to Stephen for providing much coffee, wit and proofreading throughout the final stages of thesis writing. Finally, I would like to thank my family and friends for their support and patience throughout this journey. I would like to especially thank my niece Ivy and my nephew Noah for bringing me so much joy.

Table of Contents

List of Figures

Chapter 1

Introduction

A *computational problem* is any problem that, in principle, can be solved by a computer. Each such problem can be seen as a collection of *problem instances*, with a solution for each particular instance. A *decision problem* is a computational problem for which the solution is always "yes" or "no". For example, the problem "is x a prime number?" is a decision problem that has the set of all integers as problem instances. A solution to this problem for a particular value of x is "yes" if x is prime, and "no" otherwise.

Computational complexity theory aims to classify computational problems according to their inherent difficulty. At the most basic level, problems are categorised into those which can be solved efficiently by an algorithm and those which cannot. A problem is considered *tractable* if there exists an algorithm that solves any instance of size n using at most a number of steps which is proportional to a polynomial function of n, written as $\mathcal{O}(n^c)$ where c is a fixed constant. Such problems are said to be solvable in *polynomial time*. The class P contains all decision problems which can be solved in polynomial time. The class NP (*nondeterministic polynomial time*) contains decision problems for which a proposed solution can be verified in polynomial time. Many natural problems in computer science belong to this class. Observe that the class P is a subset of the class NP. Whether the complexity class P is equal to the class NP is a major open problem in computer science, with a prize of $1 million offered to the person who solves it. It is a standard assumption in the literature that $\mathsf{P} \neq \mathsf{NP}$, and that is what will be assumed here. An NP-complete problem is at least as hard to solve as any problem in NP. Unless $\mathsf{P} = \mathsf{NP}$, any problem which is NP-complete cannot be solved in polynomial time in general. Such problems are considered *intractable*, since we do not expect to find a polynomial-time algorithm to solve any of these problems for all possible inputs.

A criticism of classical complexity theory is that there exist many computational problems which are classed as intractable but are efficiently solvable on many real-world data sets. This is usually because the problem instances that we face in practice have some specific structure

which means that they can be solved more quickly than the upper bound provided by classical complexity theory. Motivated by this shortcoming, *parameterised complexity theory* offers a more refined classification of intractable problems. Specifically, the parameterised approach investigates how the structure of a problem instance influences how easily it can be solved. A parameterised problem takes as input an instance x of a computational problem, together with the value of some parameter of the problem for the instance x. A parameterised problem is considered *tractable* if any instance of the problem can be solved in time $n^{\mathcal{O}(1)}f(k)$, where n is the size of the instance, k is the value of the parameter, and f is any computational function. Such problems are said to be *fixed parameter tractable* (FPT) *parameterised by* k.

Computational counting problems are concerned with determining the number of different solutions to an instance of a computational problem. Solving a counting problem is at least as hard as solving its decision counterpart since if we know the number of solutions to a problem, then we immediately know whether or not a solution exists. *Counting complexity theory* describes the computational difficulty of counting the number of solutions to an instance of a computational problem. A counting problem is solvable in polynomial time if we can count the number of solutions to an instance of the problem in time depending polynomially upon the size of the instance. A counting problem is said to belong to the class #P (pronounced "number P" or "sharp P") if its decision version belongs to NP. A counting problem that is #P-hard is at least as difficult as any counting problem belonging to #P. No #P-hard counting problem can be solved in polynomial time unless $\mathsf{P} = \mathsf{NP}$. We can also parameterise counting problems. A parameterised counting problem takes as input an instance of a computational problem, together with the value of a parameter of the problem for the instance. A parameterised counting problem is said to belong to the complexity class FPT parameterised by k if an instance of size n with parameter value k can be solved in time $n^{\mathcal{O}(1)}f(k)$ for some computable function f.

This thesis is concerned with finding effective and useful parameterisations of well-known computational counting problems. Namely, we consider computational counting problems belonging to the following three families: subgraph counting, stable matching, and monochromatic partitions of edge-coloured graphs. In each setting, we describe efficient parameterised algorithms for counting the number of solutions, or else indicate that such algorithms are unlikely to exist.

The subgraph counting problem asks for the number of ways in which one graph appears as a subgraph of another. Subgraph counting has many and varied applications in the analysis of large real-world networks. For instance, in [3], methods for subgraph counting are used to identify fraudulent tax claims, while in [4], the frequency of different small subgraphs in a host graph is studied in the context of gene transcription in cells. The subgraph counting problem is #P-hard in general [5]. In Chapter 3, we study subgraph counting in the setting where the host graph contains a very small number of vertices with high degree, and the

subgraph is much smaller than the host graph. We provide examples of large real world networks with this degree structure, and describe an FPT algorithm for subgraph counting in this setting parameterised by the size of the pattern graph.

An instance of a stable matching problem consists of a set of agents, together with an ordinal preference ordering for each agent over the set of their available partners. A solution, called a *stable matching*, is a way of matching the agents together so that no pair of agents would mutually prefer to deviate from the assignment. Practical examples of stable matching problems include the problem of assigning junior doctors to hospitals [6], or pairing children with adoptive families [7]. The *stable marriage* problem involves assigning men and women into man-woman pairs such that no man and woman would prefer to be together than with their assigned partners in the matching. The *stable roommates* problem is a generalisation of stable marriage in which agents may be matched with any other agent. *Hospitals/residents* is a second well-known generalisation of stable marriage in which a set of residents is assigned to a set of hospitals, and each hospital has a quota of how many residents it can admit.

In the basic model of stable matching, each agent considers all available partners as acceptable, and ranks their available partners in strict order of preference. Under this model, many important stable matching problems are solvable in polynomial time [8, 9]. A more realistic model of stable matching allows agents to declare some of their available partners as unacceptable (preference lists are *incomplete*), and to regard subsets of their available partners as equally desirable (preference lists contain *ties*). Under this model, even basic stable matching problems are intractable in general [10]. Here, we study a parameterisation of stable matching problems in which the set of agents belong to a small number of "types", where agents of the same type have identical preference lists and are regarded as equally desirable by their available partners. Under this parameterisation, Meeks and Rastegari [11] showed that several key stable matching problems which are computationally hard in general belong to FPT parameterised by the number of types. Here, we extend the work of Meeks and Rastegari in the counting setting. Counting stable matchings is computationally hard in general [12], even when agents' preference lists are complete and do not contain ties. In Chapter 4, we show that the problem of counting stable matchings can be efficiently solved for several key stable matching variants when we fix the number of allowable agent types. In Chapter 5, we describe an algorithm for approximately counting stable matchings in the stable marriage setting with a superior runtime bound to the exact algorithm described in Chapter 4.

An *edge-colouring* of a graph is an assignment of labels called colours to its edge set. An edge-coloured graph is called *monochromatic* if its edges all have the same colour. *Monochromatic partitioning problems* ask about the number of monochromatic subgraphs needed to partition the vertex set of a large edge-coloured host graph. Much of the literature on monochromatic partitioning problems has focused on solving monochromatic partitioning

problems for very specific host graphs. Of particular interest has been to find the minimum number of monochromatic subgraphs (specifically paths, cycles and trees) needed to partition the vertex set of a complete or complete bipartite graph whose edges have been coloured using a fixed number of colours. The problem of deciding whether the vertices of any edge-coloured graph can be partitioned into monochromatic paths is NP-complete even if only two colours are used to colour the edges [13]. In Chapter 6, we describe an FPT algorithm for counting partitions of edge-coloured graphs into k monochromatic paths parameterised by both the number of colours used and a measure of how "tree-like" the host graph is.

Chapter 2

Preliminaries

This chapter provides the definitions of key terms and describes the notation that will be used throughout this thesis. In Section 2.1, we describe the notation that will be used in relation to sets. In Section 2.2, we provide some basic graph theoretic definitions and describe the notation that will be used to describe graphs and their properties. In Section 2.3, we provide a brief introduction to the relevant areas of computational complexity theory, including parameterised complexity and counting complexity.

2.1 Set Theory

A *set* S is an unordered collection of distinct elements. Two sets are *equal* if they contain exactly the same elements. If a set S contains only elements a, b and c, then we may write this as $S = \{a, b, c\}$. If S contains an element a, then we say that a is a *member* of S, written $a \in S$. If an element d is not a member of S, then we write this as $d \notin S$. The *size* of a finite set S, denoted by $|S|$, is equal to the number of elements contained in S. A set S is said to be *empty*, written $S = \varnothing$, if it does not contain any elements. We may also define a set by stating the properties that each of its members must satisfy. Given a set X and property Φ, we may define the set $S = \{x : x \in X \text{ and } \Phi(x)\}$ containing all elements from the set X which satisfy the property Φ. We denote the set containing each of the natural numbers from 1 to n by $[n]$.

If every member of a set A is also a member of some set B, then we say that A is a *subset* of B, written as $A \subseteq B$. Equivalently, we may say that B is a *superset* of A, written as $B \supseteq A$. If B contains at least one element which does not appear in A, then we say that A is a *proper subset* of B, written as $A \subset B$ or $B \supset A$. The *union* of sets A and B, denoted by $A \cup B$, is the set containing all elements which are contained in either or both of A and B. The *intersection* of A and B, written $A \cap B$, is the set containing elements which are

common to both A and B. We say that sets A and B are *disjoint* if $A \cap B = \varnothing$. We use $A \setminus B$ to denote the set containing all elements of A which are not also contained in B. The *Cartesian product* of A and B, written $A \times B$, is the set containing all ordered pairs (a, b) such that $a \in A$ and $b \in B$. If $A = B$, then we may write this as A^2.

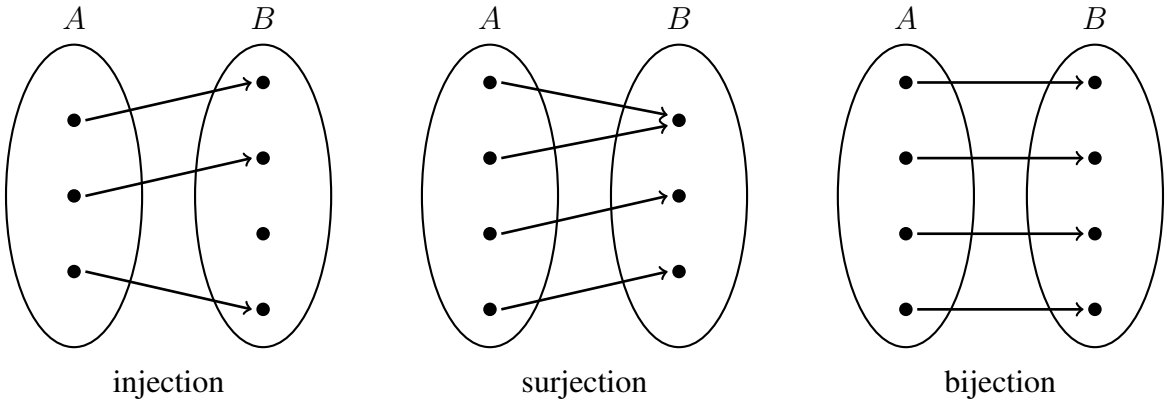

Figure 2.1: An illustration of an injection, a surjection, and a bijection

Given sets A and B, a *function* $f : A \to B$ from A to B maps each element of the set A to one or more elements of B. If f maps an element a in A to an element b in B, then we write this as $f(a) = b$. If a is the only element which is mapped to b, then we may also write $f^{-1}(b) = a$. Otherwise, we write $a \in f^{-1}(b)$. We say that f is *injective* if each element of B is mapped to by at most one element of A. We may also call f an *injection*. We say that f is *surjective* (or a *surjection*) if each element of B is mapped to by at least one element of A. If f is both injective and surjective (each element of B is mapped to by exactly one element of A), then we say that f is *bijective*, or a *bijection*.

There are several infinte sets with mathematical significance. Here, we discuss only some of these. The set $\mathbb{N} = \{0, 1, 2, \ldots\}$ contains all natural numbers. The set $\mathbb{Z}$ contains all integers. We use $\mathbb{Z}^+$ to denote the set containing all positive integers. The set $\mathbb{Q}$ is used to denote the set of rational numbers (those which can be written as a fraction a/b for some $a, b \in \mathbb{Z}$ with $b \neq 0$).

2.2 Graph Theory

We note that the notation described in this section has been adopted predominantly from Bollobás' *Modern Graph Theory* [14]. In graph theory, a *graph* $G = (V(G), E(G))$ consists of a finite set $V(G)$ of vertices and a set $E(G)$ of edges connecting pairs of vertices. We denote the number of vertices in G by $|V(G)|$ and the number of edges by $|E(G)|$. Two vertices in a graph are said to be *adjacent* if they are connected by an edge. We write uv to

denote the edge connecting vertices u and v. We call u and v the *endpoints* of the edge uv, and we say that u and v are *incident* to the edge uv. If uv is an edge in G, then we write this as $uv \in E(G)$. A graph with n vertices is said to have *order* n. A graph is *simple* if it does not contain multiple edges between a single pair of vertices, or any edges which start and end at the same vertex. Figure 2.2 illustrates an example of a simple graph. In this thesis, all graphs are assumed to be simple.

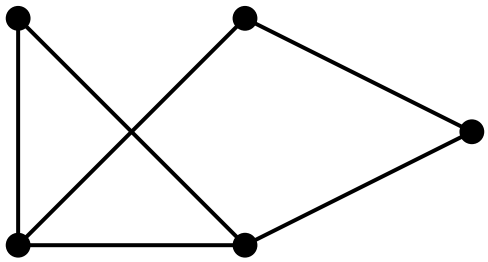

Figure 2.2: An example of a simple graph

Let $G = (V(G), E(G))$ be a graph. Any vertex which is adjacent to a vertex $v \in V(G)$ is called a *neighbour* of v. The set of neighbours of v in $V(G)$ is referred to as the *neighbourhood* of v. The *degree* of a vertex v, denoted by $deg(v)$, is equal to the number of neighbours of v. If every vertex in G has degree at most Δ, then G is said to have *maximum degree* Δ. An *independent set* of G is a subset U of $V(G)$ such that no pair of vertices in U are adjacent in G. The *independence number* $\alpha(G)$ of G is equal to the size of the largest independent set in $V(G)$. A *vertex cover* of G is a subset of $V(G)$ such that every edge in $E(G)$ is incident to at least one vertex in the subset. The *vertex cover number* of G is equal to the size of its smallest possible vertex cover. A *path* in G is a sequence of distinct adjacent vertices. A path is called a *cycle* if the sequence starts and ends at the same vertex. The *length* of a path is equal to the number of edges in the path. The *endpoints* of a path are the first and last vertices on the path. The *distance* between two vertices u and v in a graph is equal to the length of the shortest path from u to v. We say that a graph $G = (V(G), E(G))$ is *connected* if there is a path between every pair of vertices in $V(G)$. A *tree* is any graph which does not contain a cycle. A *forest* is a collection of disjoint trees. We say that two graphs $G = (V(G), E(G))$ and $G' = (V(G'), E(G'))$ are *vertex-disjoint* if $V(G) \cap V(G') = \varnothing$. In what follows, when we say that two graphs are *disjoint* we mean that they are vertex-disjoint.

A *subgraph* of a graph $G = (V(G), E(G))$ is a graph formed from a subset $V'(G)$ of $V(G)$ and a subset $E'(G)$ of $E(G)$ connecting pairs of vertices in $V'(G)$. We may use $G \setminus V'(G)$ to denote the subgraph of G formed by removing the subset $V'(G)$ of vertices from $V(G)$ and all incident edges of the vertices in $V'(G)$ from $E(G)$. An *induced subgraph* of G is a graph formed from a subset $V'(G)$ of $V(G)$ and a subset $E'(G)$ containing all edges present in $E(G)$ which connect pairs of vertices from $V'(G)$. A *spanning subgraph* of G

contains all vertices in $V(G)$. We say that a graph $G = (V(G), E(G))$ *contains a graph* $H = (V(H), E(H))$ *as a subgraph* if there is an injection $f : V(H) \to V(G)$ such that if $uv \in E(H)$, then $f(u)f(v) \in E(G)$. We say that G *contains* H *as an induced subgraph* if the function f is such that $uv \in E(H)$ if and only if $f(u)f(v) \in E(G)$. Two graphs $G = (V(G), E(G))$ and $G' = (V(G'), E(G'))$ are said to be *isomorphic* if there is a bijection $f : V(G) \to V(G')$ such that, for each pair $u, v \in V(G)$, we have that $uv \in E(G)$ if and only if $f(u)f(v) \in E(G')$. A graph G which contains an edge between every pair of

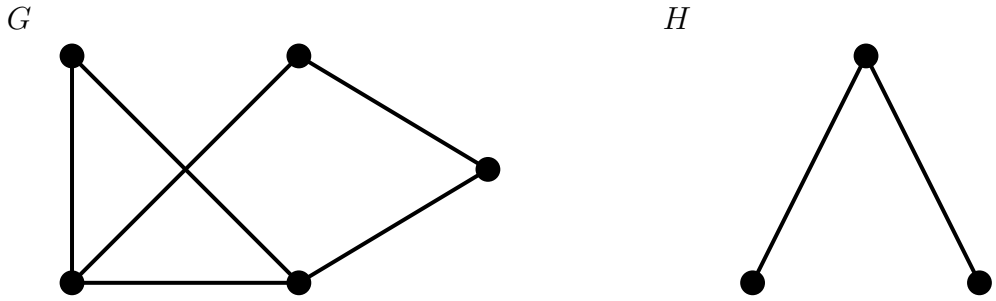

Figure 2.3: An example of a graph G and a subgraph H of G

vertices is called a *complete graph*. We may denote the complete graph on n vertices by K_n. A graph $G = (V(G), E(G))$ is called *bipartite* if there exists a partition of the vertices in $V(G)$ into disjoint sets U and V such that no pair of vertices within the same set are adjacent. We may denote such a graph by $G = ((U, V), E(G))$. If $|U| = |V|$, then we say that G is *balanced*. A *complete bipartite graph* $G = ((U, V), E(G))$ is a bipartite graph containing an edge between every pair of vertices from U and V. A complete bipartite graph $G = ((V, U), E(G))$ with $|V| = n_1$ and $|U| = n_2$ may be denoted by K_{n_1,n_2}. The complete balanced bipartite graph on $2n$ vertices is denoted by $K_{n,n}$.

2.3 Computational Complexity Theory

In this section, we provide an overview of the areas of computational complexity theory which are relevant to this thesis. A comprehensive introduction to computational complexity theory is provided in Garey and Johnson's *Computers and Intractability* [15]. Both *Fundamentals of Parameterized Complexity* by Downey and Fellows [16] and *Parameterized Complexity Theory* by Flum and Grohe [17] are excellent guides to parameterised complexity theory. *Parameterized Algorithms* by Cygan et al. [18] is another popular introductory text in this area.

2.3.1 Classical Complexity Theory

Complexity theory provides a means of classifying computational problems according to whether they can reasonably be solved by an algorithm. The *time complexity* (often simply called the *complexity*) of a problem describes the amount of time required to solve any instance of the problem using an algorithm. If any instance of the problem can be solved by an algorithm in time $\mathcal{O}(n^c)$, where n is the size of the instance and c is a fixed constant, then we say that the problem is solvable in *polynomial time*. Any problem which is solvable in polynomial time is considered tractable. The complexity class P contains all computational problems which are solvable in polynomial time. The class NP contains the set of decision problems for which a possible solution can be verified in polynomial time. A *reduction* is an

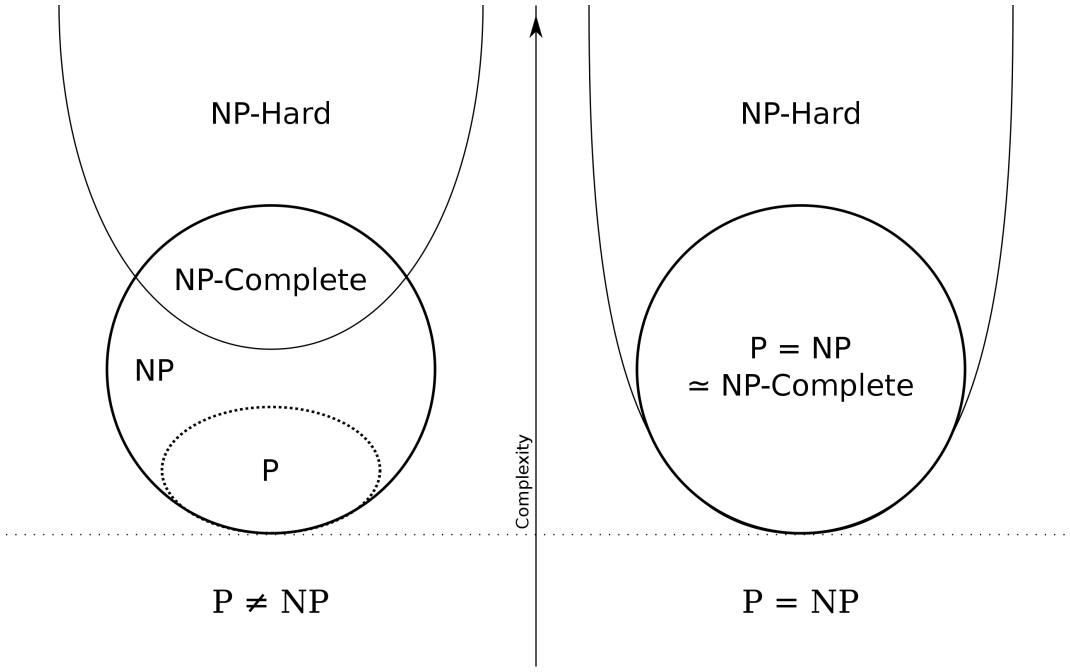

Figure 2.4: An illustration of the P versus NP debate [1]

algorithm that can be used to compare the complexity of two problems. For decision problems, a polynomial-time many-one reduction transforms any instance of one problem into an instance of another problem in polynomial time. In addition, the output of the algorithm for each instance of the first problem must be the same as the output for the corresponding instance of the second problem. A polynomial-time algorithm for solving one problem would then allow us to solve the other in polynomial time.

A decision problem is said to be NP-*complete* if it belongs to NP, and any other problem in NP can be reduced to this problem via a polynomial-time many-one reduction. In other words, NP-complete problems are at least as hard as all problems in NP since a polynomial-time algorithm for any NP-complete problem would allow us to solve any problem in NP

in polynomial time. The class of NP-hard problems contains the set of problems to which any problem in NP can be reduced, but which do not necessarily belong to the class NP. In other words, NP-hard problems are at least as hard as the hardest problem in NP. Note that an NP-hard problem need not be a decision problem. Figure 2.4 illustrates the relationship between these complexity classes in the case where we assume that $\mathsf{P} = \mathsf{NP}$, and in the case where we assume that $\mathsf{P} \neq \mathsf{NP}$.

An *optimisation problem* is a computational problem that asks for the "best' solution in some well-defined sense. An example of an optimisation problem is the problem of finding a maximum size matching in a graph. Any optimisation problem can be recast as an (easier) decision problem by fixing the size of a possible optimal solution. It follows that if the decision counterpart to an optimisation problem is NP-hard then the optimisation problem itself is NP-hard. In the other direction, a polynomial-time algorithm for an optimisation problem also provides a polynomial-time solution to any corresponding decision problem. Since the class NP contains only decision problems, an optimisation problem cannot belong to NP, nor can it be NP-complete.

2.3.2 Parameterised Complexity Theory

Parameterised complexity theory describes the complexity of computational problems as a function of the size of a problem instance, as well as one or more additional (computable) parameters of each instance. The aim is to provide a fuller picture of the relationship between the instances of a computational problem, and the computational difficulty involved in finding a solution. A *parameterised problem* consists of a collection of instances of a computational problem, together with the value of one or more computable parameters for each instance. A parameterised problem is said to be *fixed parameter tractable* (FPT) *parameterised by* k if any instance of size n with parameter value k is solvable in time $f(k)n^{\mathcal{O}(1)}$, where f is a computable function. The class FPT can be seen as the parameterised counterpart to the class P from the classical complexity setting.

An FPT-*reduction* is the parameterised equivalent of a polynomial-time many-one reduction. Specifically, an FPT-reduction from problem A to problem B is an FPT algorithm which, for every instance and parameter pair (x, k) of A, returns a corresponding a pair (x', k') of B in time $|x|^{\mathcal{O}(1)} f(k)$ for some computable function f such that

- (x, k) is a yes-instance if and only if (x', k') is a yes-instance, and
- $k' \leq g(k)$ for some computable function g.

It follows that if there exists an FPT-reduction from A to B, and B is known to belong to FPT, then we can also solve A via an FPT-algorithm.

The class W[1] of parameterised problems can be regarded as the parameterised counterpart to NP. In fact, there exists an entire family of complexity classes W[t], for positive integers t, which are known collectively as the W-*hierarchy*, where W[0] = FPT. It is widely believed that W[i] is a strict subset of W[j] for all pairs $i < j$ [17]. There are several equivalent formal definitions of the W-hierarchy available, all of them quite technical. For more information, see [19].

A parameterised problem which is W[i]-complete is at least as hard as any problem in W[i]. Formally, a problem that is W[i]-complete must belong to W[i], and can be reached via an FPT-reduction from any problem in the class W[i]. A problem is W[i]-hard if it can be reached via an FPT reduction by any problem in W[i] and may or may not belong to W[i]. Problems that are W[i]-hard are at least as difficult to solve as those belonging to W[i]. Many natural parameterised problems which belong to the W-hierarchy fall into one of W[1] or W[2]. For example, the problem of deciding whether a graph contains an independent set of size at least k is W[1]-complete parameterised by k [19]. The problem of deciding whether a graph contains a dominating set (a set of vertices such that all other vertices in the graph are adjacent to at least one vertex in the set) of size at most k is W[2]-complete parameterised by k [19].

A parameterised problem belongs to the class XP if an instance of size n with parameter value k is solvable in time $n^{f(k)}$ for some computable function f. Problems belonging to XP can be solved in polynomial time when the parameter is treated as a constant. The W-hierarchy forms a subset of XP.

2.3.3 Approximation Algorithms

Approximation algorithms are used to find an approximately optimal solution to an optimisation problem when finding an optimal solution is computationally hard. An approximation algorithm is said to be an $f(n)$-*approximation algorithm* if the solution returned by the algorithm for an instance of size n is guaranteed to be within a multiplicative factor of at most $f(n)$ times optimal. The value of $f(n)$ is called the *approximation ratio* of the algorithm. Note that if the function $f(n)$ is constant, then the approximation algorithm may be called a *constant-factor approximation algorithm*.

An optimisation problem is said to have a *polynomial-time approximation scheme* (PTAS) if there exists an algorithm which, given a problem instance of size n and a real number $\epsilon > 0$ as input, returns a solution which is within a factor $(1+\epsilon)$ of optimal for minimisation problems, and $(1-\epsilon)$ of optimal for maximisation problems, in time $f(n)$ for some polynomial function f of n. We call the value returned by such an algorithm an ϵ-*approximation* of the solution.

A *fully polynomial-time approximation scheme* (FPTAS) is an algorithm which takes an

instance of an optimisation problem and a real number $\epsilon > 0$ as input and returns an ϵ-approximation of the solution in time $f(n, 1/\epsilon)$ where n is the instance size and f is a polynomial function of n and $1/\epsilon$.

Randomised approximation algorithms offer a further tradeoff between certainty and efficiency. A *fully polynomial-time randomised approximation scheme* (FPRAS) [20] is an algorithm which takes an instance of an optimisation problem of size at most n and real numbers $\epsilon > 0$ and $0 < \delta < 1$ as input, and with probability at least $(1 - \delta)$ returns an ϵ-approximation of the solution in time $f(n, 1/\epsilon, \log(1/\delta))$ for some polynomial function f. If the output of the algorithm is outwith the $(1 \pm \epsilon)$ error bounds (note that this occurs with probability at most δ), then we say that the algorithm has failed. Hence, it is standard to call δ the *failure probability* of the algorithm.

A *fixed parameter tractable randomised approximation scheme* (FPTRAS) [21] is the parameterised analogue of an FPRAS. Given an instance of a parameterised optimisation problem with size at most n and parameter value k, as well as real numbers $\epsilon > 0$ and $0 < \delta < 1$ as input, an FPTRAS returns an ϵ-approximation of the solution with probability at least $(1-\delta)$ in time $g(k)f(n, 1/\epsilon, \log(1/\delta))$ for some computable function g and a polynomial function f.

2.3.4 Counting and Sampling

We begin by noting that, in the process of counting solutions to a computational problem, we must often handle values which are exponentially larger than the size of the input. As such, it is important to take into consideration the time needed to compute arithmetic operations on intermediate values. The addition of an a-bit and a b-bit number takes time $\mathcal{O}(\max(a, b))$. Multiplication or division involving an a-bit number and a b-bit number takes time $\mathcal{O}(ab)$. Note that in this thesis, we shall use $\log$ to mean $\log_2$. We use $\ln$ to denote the natural logarithm.

Like decision problems, counting problems are classified into a hierarchy of complexity classes according to the computational effort required to reach a solution. Given a decision problem A, we may use $\#A$ to denote the problem of counting the number of solutions to an instance of A. A counting problem belongs to FP if the number of solutions can be counted in time depending polynomially upon the size of the input. A counting problem belongs to the class #P if its decision version belongs to NP. More formally, #P contains the set of function problems f such that, for any input x, the value of $f(x)$ (the solution to the function problem f on x) is equal to the number of accepting paths of a nondeterministic polynomial-time Turing Machine on x [5]. Note that if $\#\mathsf{P} = \mathsf{FP}$, then $\mathsf{P} = \mathsf{NP}$.

Unlike reductions for decision problems, there is no singular definition of a reduction for

counting problems. Two main definitions used in the literature are the following.

- **Parsimonious reduction:** There is said to be a *parsimonious reduction* from a counting problem $\#A$ to another counting problem $\#B$ if there exists an algorithm which transforms any instance x of $\#A$ into an instance y of $\#B$. In particular, x and y must have the same number of solutions in $\#A$ and $\#B$ respectively.
- **Turing reduction:** There is a *Turing reduction* from $\#A$ to $\#B$ if there is an algorithm that can solve any instance of $\#A$ given access to an oracle for solving $\#B$.

Since Turing reductions do not require that the number of solutions to the original and transformed instance of the problem are maintained, this definition is less restrictive than parsimonious reductions and is therefore more commonly used in practice.

A counting problem is $\#\mathsf{P}$-complete if it can be reached from any problem in $\#\mathsf{P}$ via a polynomial-time counting reduction. Due to there being multiple methods for reductions in counting problems, it is necessary to specify the kind of reduction with respect to which a problem is $\#\mathsf{P}$-complete. Under both kinds of reduction, no $\#\mathsf{P}$-complete problems are solvable in polynomial time unless $\mathsf{P} = \mathsf{NP}$. In fact, Valiant [5, 22] showed that there exist problems whose decision version is solvable in polynomial time but for which the counting version is $\#\mathsf{P}$-complete. A classic example is the problem of counting perfect matchings in bipartite graphs [5], which is $\#\mathsf{P}$-complete under Turing reductions. A problem that is $\#\mathsf{P}$-hard is at least as difficult as any problem belonging to $\#P$ (can be reached via a polynomial-time counting reduction) but need not belong to the class $\#\mathsf{P}$ itself.

Many interesting counting problems are $\#\mathsf{P}$-hard. As in the decision setting, this has motivated the search for a richer classification framework for problems that are intractable in the classical sense. As in the decision setting, counting problems which can be solved in time $f(k)n^{\mathcal{O}(1)}$ for some parameter k belong to the class of fixed parameter tractable (FPT) problems. The class $\#\mathsf{W}[1]$ is the counting analogue of $\mathsf{W}[1]$ in the decision setting. The classes $\#\mathsf{W}[i]$, for each $i > 1$, are defined similarly.

Approximation algorithms can also be used to approximate the number of solutions to counting problems. Note that for counting problems, a solution is always a positive number. It follows that, for any $\epsilon > 1$, a trivial algorithm outputs 0 for all inputs. As such, it is standard to restrict the value of ϵ to the range $0 < \epsilon < 1$ for counting problems. A PTAS for a counting problem is an algorithm which, given a problem instance of size at most n and a real number $\epsilon > 0$ as input, returns an ϵ-approximation of the number of solutions in time $f(n)$ for some polynomial function f. Each of FPTAS, FPRAS and FPTRAS for counting are defined analogously.

Counting and sampling are closely related problems computationally. Given an instance of a computational problem, a sampling algorithm returns a solution from the set of all solu-

tions according to some probability distribution. A *uniform random sampler* is an algorithm which, given an instance of a computational problem as input, samples from the solution set by selecting each solution with equal probability. An *almost uniform random sampler* is an algorithm which takes as input an instance of a computational problem and a real number $\epsilon > 0$, and selects each element from the solution set S with probability p such that $(1 - \epsilon)/|S| \leq p \leq (1 + \epsilon)/|S|$.

A *fully polynomial-time almost uniform sampler* (FPAUS) [20] is an algorithm which takes as input an instance of a computational problem with solution set S and a real number $\epsilon > 0$, and selects an element $s \in S$ with probability p such that $(1 - \epsilon)/|S| \leq p \leq (1 + \epsilon)/|S|$ in time $f(n, \log(1/\epsilon))$, where f is a polynomial function, and n is the size of the problem instance.

A *fixed parameter tractable almost uniform sampler* (FPTAUS) is the parameterised analogue of an FPAUS. An FPTAUS takes an instance of a parameterised problem and a real number $\epsilon > 0$ as input, and selects an element $s \in S$ with probability p such that $(1-\epsilon)/|S| \leq p \leq (1+\epsilon)/|S|$. The runtime of an FPTAUS is bounded by $g(k)f(n, \log(1/\epsilon))$, where f is a polynomial function, g is any computable function of the parameter value k, and n is the size of the problem instance.

Chapter 3

Subgraph Counting

3.1 Motivation

Given a pair of graphs H and G, the subgraph counting problem asks for the number of ways in which H appears as a subgraph of G. The graphs G and H are commonly referred to as the *host graph* and *pattern graph* respectively. The subgraph counting problem is a well-studied computational problem with wide-ranging applications and as many variations. A recent study by Sahu et al. [23] found that the problem of finding and counting fixed subgraphs was the fourth most popular graph query used in practice.

The subgraph counting problem is $\#\mathsf{P}$-hard in general [5]. In fact, even the decision version of the subgraph counting problem - *subgraph isomorphism* - is NP-complete, even when restricted to planar host graphs [15]. It follows that we do not expect to find an efficient algorithm for subgraph counting for all possible pairs of host and pattern graphs. However, in important practical applications of subgraph counting, the pattern graph is often significantly smaller than the host graph [24, 25, 26]. It is therefore reasonable to ask whether we can attain tractability by parameterising the problem by the order of the pattern graph. However, unless $\mathsf{W}[1] = \mathsf{FPT}$, there is no FPT algorithm for subgraph counting parameterised by the order of the pattern graph in general [27].

In practice, many large real-world networks contain a very small number of vertices with high degree (we present examples of such networks in Section 3.5). Here, we focus on subgraph counting in host graphs containing a constant number of high-degree vertices. It follows from a meta-theorem [28] that subgraph counting is in FPT parameterised by the order of the pattern graph in host graphs with this structure. However, the time bound provided by this meta-theorem contains impractically large constants. In this chapter, we describe a more practical FPT algorithm for subgraph counting in host graphs with a constant number of high-degree vertices, parameterised by the order of the pattern graph.

3.2 Definitions and Notation

Let $G = (V(G), E(G))$ and $H = (V(H), E(H))$ be graphs. An *embedding* of H into G is an injective function $f : V(H) \to V(G)$ such that if $uv \in E(H)$ then $f(u)f(v) \in E(G)$. Note that such a function exists only if G contains H as a subgraph. We use $\#Emb(H, G)$ to denote the number of embeddings of H into G. Automorphisms count symmetries within a graph. Formally, an *automorphism* of a graph G is an embedding of G into itself. The number of automorphisms of G is denoted by $\#Aut(G)$. The number of times that H appears as a subgraph of G, written as $\#Sub(H, G)$, is equal to the number of embeddings of H into G divided by the number of automorphisms of H. Thus, we have that

$$\#Sub(H, G) = \frac{\#Emb(H, G)}{\#Aut(H)}.$$

Let $G = (V(G), E(G))$ and $H = (\{v_1, \ldots, v_m\}, E(H))$ be graphs. Let $V_{v_i,G}$ be a subset of $V(G)$ for each $i \in [m]$, and let $V_{H,G} = \{V_{v_i,G}, \ldots, V_{v_m,G}\}$ be the set containing each of these subsets. We call an embedding $f : V(H) \to V(G)$ of H into G an *embedding of H into G with lists $V_{H,G}$* if $f(v_i) \in V_{v_i,G}$ for each $i \in [m]$. We denote the number of embeddings of H into G with lists $V_{H,G}$ by $\#Emb(H, G, V_{H,G})$. Note that if $V_{v_i,G} = V(G)$ for each $i \in [m]$, then we have that $\#Emb(H, G, V_{H,G}) = \#Emb(H, G)$. We say that a class $\mathcal{C}$ of graphs has *almost-bounded degree* if there exist constants Δ and ℓ such that every graph G in $\mathcal{C}$ contains at most ℓ vertices with degree exceeding Δ.

3.3 Literature Review

In this section, we survey existing results on the complexity of the subgraph counting problem. Due to the large volume of literature on this topic, we will cover only parameterised results, where the order of the pattern graph is taken as a parameter. It is standard in the subgraph counting literature to denote the pattern graph by H and the host graph by G. The numbers of vertices in H and G are denoted by m and n respectively.

It follows from a seminal paper due to Valiant [5] that the subgraph counting problem is #P-complete in general (with respect to Turing reductions), motivating the search for efficient parameterisations of the problem. It is worth noting that the naive brute-force method of checking all possible mappings of $V(H)$ to $V(G)$ gives an $\mathcal{O}(n^m)$ solution. It follows that the subgraph counting problem belongs to XP parameterised by the order of the pattern graph. However, under the same parameterisation, subgraph counting is #W[1]-hard even if the pattern graph is a cycle [29], path [29], clique [30] or matching [31]. In fact, the problem of counting matchings of size k is #W[1]-hard parameterised by k even when the host graph

is bipartite [27].

The cliquewidth [32] of a graph is a metric used to describe the complexity of the operations needed to construct the graph. Graph classes with constant cliquewidth include highly structured dense graphs such as cliques and complete bipartite graphs. Nowhere dense graph classes [33] are graph classes which are sparse in a well-defined sense. Nowhere dense graph classes include planar graphs, bounded degree graphs, and graph classes of bounded expansion [34]. *First-order logic* (also called FO logic) is a form of logic that allows us to express certain computational problems as logical sentences containing variables $x_1, x_2, \ldots$, and logical symbols (such as $\exists, \forall, \wedge, \vee, \neg$), as well as both functions and relations over the variables. The subgraph counting problem can be written as a sentence in first-order logic [17] whose length is independent of the size of the input. As such, it follows from two celebrated meta-theorems that subgraph counting is in FPT parameterised by the order of the pattern graph for host graphs which are nowhere-dense [28], or which have bounded cliquewidth [35]. Since graphs of bounded treewidth and bounded degree are nowhere dense, it follows that subgraph counting is also tractable for host graphs from each of these classes.

Alon et al. [36] showed that counting cycles of length ≤ 7 can be achieved in time $\mathcal{O}(n^\omega)$, where ω is the exponent of matrix multiplication (known to be at most 2.37188 [37]). In a breakthrough paper, Curticapean and Marx [27, 38] proved a dichotomy result for subgraph counting parameterised by the order of the pattern graph - if H is drawn from a class of graphs with bounded vertex cover number then subgraph counting is in FPT, otherwise it is $\#\mathsf{W}[1]$-hard. Björklund et al. proved that the problem of counting connected subgraphs is in FPT parameterised by the order of the pattern graph and the maximum degree of the host graph for pattern graphs with a fixed-size "balancer" [39]. As a corollary, it is shown that the problems of counting paths, trees and cycles are in FPT parameterised by the order of the subgraph and the maximum degree of the host graph.

3.4 Contributions

It follows from a meta-theorem due to Grohe et al. [28] that any graph problem which can be expressed in FO logic is in FPT parameterised by the length of the expression for any class of graphs which is nowhere dense. The subgraph counting problem can be written as a FO logic sentence whose length is a function of the size of the pattern graph (an example for subgraph isomorphism is provided in [17]; a straightforward generalisation to counting can be made using the generalised quantifiers $\exists^{=c}$ defined in [40]). Moreover, the class of almost-bounded degree graphs is a member of the nowhere dense graph classes. It follows that the subgraph counting problem is in FPT parameterised by the order of the pattern graph in host graphs from the class of graphs with almost-bounded degree. However, the algorithm

provided by the meta-theorem contains very large constants, which prevents it from being useful in practice.

In Section 3.5, we provide examples of large real-world networks containing few vertices with high degree, motivating the search for practical algorithms for subgraph counting in this setting. In Section 3.6, we describe a practical FPT algorithm for subgraph counting in host graphs from the class of graphs with almost-bounded degree parameterised by the order of the pattern graph.

3.5 Real-World Networks with Few High-Degree Vertices

In this section, we present examples of real-world networks containing only a small number of vertices with high degree. The data set used to derive each of these networks has been obtained from the Stanford Large Network data set Collection [41]. Each of the networks in the data set is directed. For simplicity, here we regard the presence of one or more directed edges between a pair of vertices as a single undirected edge. Note that the data sets discussed here have been selected due to their diversity and degree distributions. Indeed, we do not claim that all relational data sets are expected to have very few high-degree vertices.

The first data set which we examine contains information about items purchased together (by the same customer at the same time) from Amazon.com on 12$^{\text{th}}$ March 2003 [41, 42]. Here, the vertices represent individual items available for purchase on Amazon; two vertices are joined by a (directed) edge if one is frequently purchased with the other in a single transaction. The second data set contains user voting data from Wikipedia [41, 43, 44, 45] over a period of 7 years. Each time that a Wikipedia user is nominated for a promotion to administrator, other users on the site are invited to vote for or against the candidate in an election. In this setting, the vertices represent individual users on the site who participated in an election either as a nominee or as a voter, and the edges represent votes between users. The final two data sets discussed in this section are web graphs representing pages from the Stanford University and University of Notre Dame websites, respectively [41, 45, 46]. In this setting, each webpage is represented by a single vertex. Two vertices are connected by an edge if one page contains a link to the other.

For each of the data sets, we present a plot (Figures 3.1, 3.3 and 3.4) of the maximum degree of the network against the number of (highest degree) vertices removed from the graph. In each case, we describe the structure of the curve produced from the data set and make suggestions about the cause of the steep drop-off in the degree distribution.

We begin by examining the network representing co-purchased products on Amazon.com.

Figure 3.1 shows a plot of the maximum degree of this network against the number of vertices removed from the network in descending order of degree. This network has 400,727 vertices and 2,349,869 edges. The maximum degree of the network is 2,747. Removing the 10 highest degree vertices from the graph results in a decrease of 1,974, or 72%, of the original maximum degree. In practice, this steep drop-off implies that a small number of products are purchased very frequently, while most products available on the website are purchased relatively infrequently - over 50% of the vertices in the graph have degree at most 10. As can be seen in Figure 3.1, the steep drop-off of the curve plateaus after the 10 highest degree vertices have been removed. The authors of [2] generated a visualisation (Figure 3.2) of a network of products which were co-purchased from Amazon.com in August 2003. We can see in this visualisation that the network contains a small number of vertices (most notably those circled in red) which are connected to a very large number of others, and a very large number of vertices (most notably those circled in green) with a very small number of neighbours. In particular, the neighbourhood of each cluster of vertices circled in green is very small, further supporting our suggestion that a small number of products appear in the vast majority of purchases from Amazon.com.

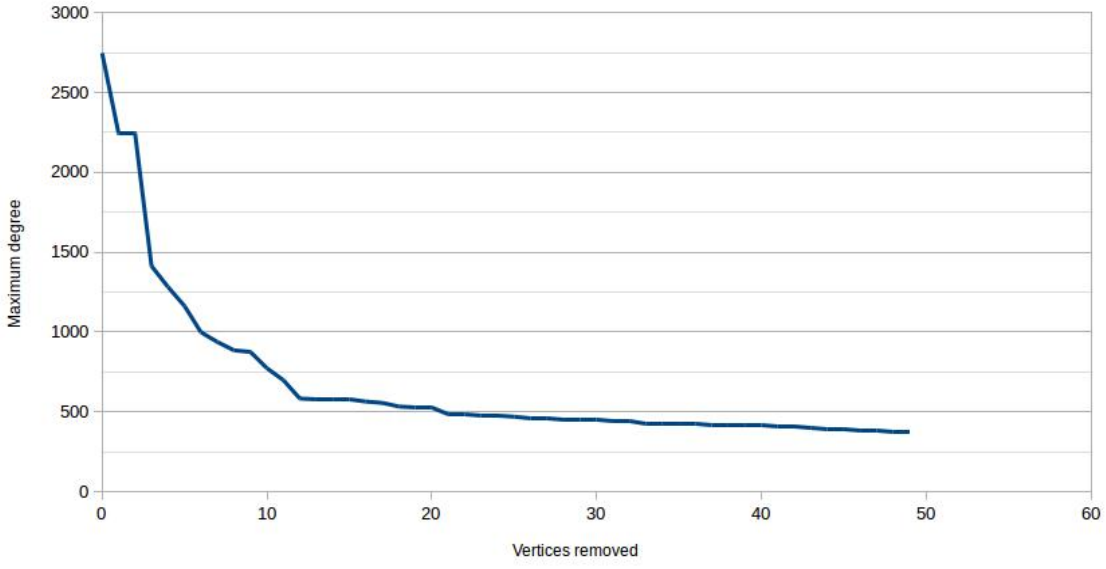

Figure 3.1: A plot of the maximum degree of a graph formed from data on co-purchased products on Amazon.com as vertices are removed from the graph in descending order of degree

We will now discuss the structure of the network formed from Wikipedia voting data. The voting data used to construct this network was obtained from a total of 2,794 elections of Wikipedia administrators, with a combined total of 103,663 total votes made or received by 7,066 users. Figure 3.3 contains a plot of the maximum degree of this network as vertices are

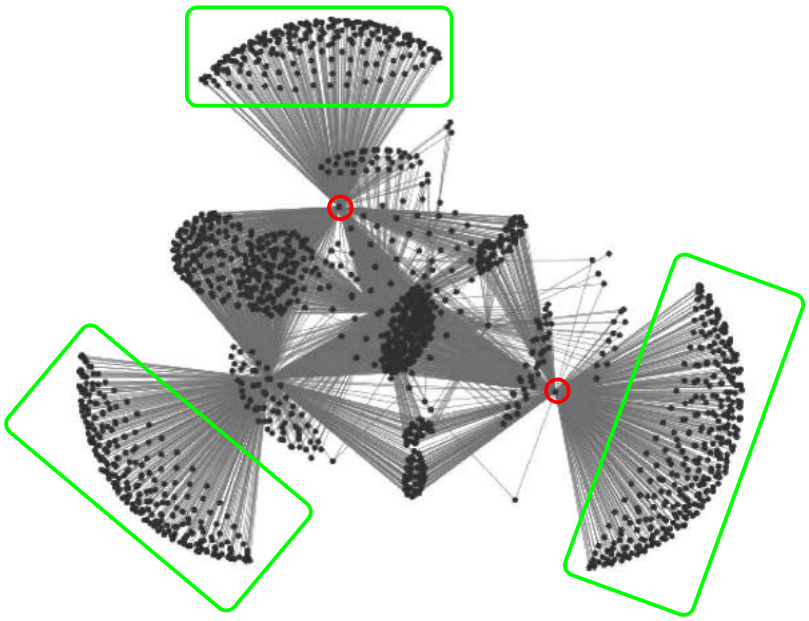

Figure 3.2: A visualisation of a network formed from co-purchased products on Amazon.com in August 2003 [2]

removed in descending order of degree. The network contains 7,115 vertices[1], and 100,762 edges. The maximum degree of the network is 1,065. In the plot, the curve is steepest between values 0 and 1 on the x-axis. Removing the single highest degree vertex from this graph reduces the maximum degree from 1,065 to 773, indicating that a single user received a very large number of votes in many of the elections. Removing the 10 highest degree vertices results in a decrease of 588, or 55%, in the maximum degree. As in the Amazon.com co-purchasing network described above, the large majority (61%) of vertices in the network have very low degree (10 or less). In this setting, this implies that the majority of users cast and received very few votes.

Figure 3.4 contains plots of the maximum degree of two web graphs against the number of high-degree vertices removed. The plot at the top of Figure 3.4 is derived from the Stanford University website (domain stanford.edu) and the plot in the bottom of Figure 3.4 is derived from the University of Notre Dame website (domain nd.edu). The web graph of the Stanford University website contains 281,903 vertices and 1,992,636 edges. The maximum degree of the graph is 38,625, and the average degree is 14. As in Figure 3.3, the greatest drop-off in the maximum degree occurs when the single highest degree vertex is removed from the graph. Specifically, removing the highest degree vertex from the stanford.edu webgraph decreases the maximum degree from 38,625 to 21,923. Removing the 10 highest degree vertices from the graph results in a drop of 19,694, or 51%, from the original maximum degree. Removing another 6 vertices results in a reduction of over 78% of the original maximum degree.

[1]Note the discrepancy between the number of users participating in the elections and the number of vertices; a possible explanation is that some users who appear in the network did not vote in any election.

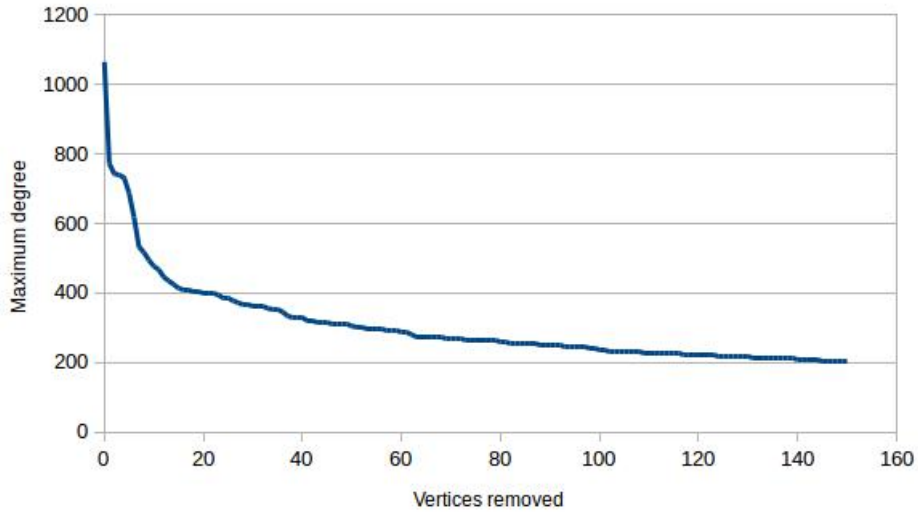

Figure 3.3: A plot of the maximum degree of a graph formed from Wikipedia administrator voting data as vertices are removed from the graph in descending order of degree

The web graph of the Notre Dame university website contains 325,728 vertices and 1,117,563 edges. The maximum degree of this network is 10,721. Once again, a significant drop-off in the maximum degree occurs when the highest degree vertex is removed. Specifically, the maximum degree drops by 3097, or 29%, of the original maximum degree. Removing the 10 highest degree vertices from the network reduces the original maximum degree by 7157, or 67%. In this setting, this structure of the webgraphs suggests that most webpages on each website are linked to a very small collection of central pages.

A network is said to have a *power-law degree distribution* if the proportion of vertices in the network having degree k is inversely related to k [47]. Note that a network with a power-law degree structure necessarily has a (proportionately) very small number of vertices with high-degree. It is well known [48] that the general webgraph formed from hyperlinks between pages on the web follows a power-law degree distribution. In [49], Dill et al. observe that "cohesive collections of Web pages" also tend to exhibit a power-law degree structure. It follows then that we should expect many websites, including those studied here, to exhibit a power-law degree distribution, and hence to have only a small number of vertices with high degree.

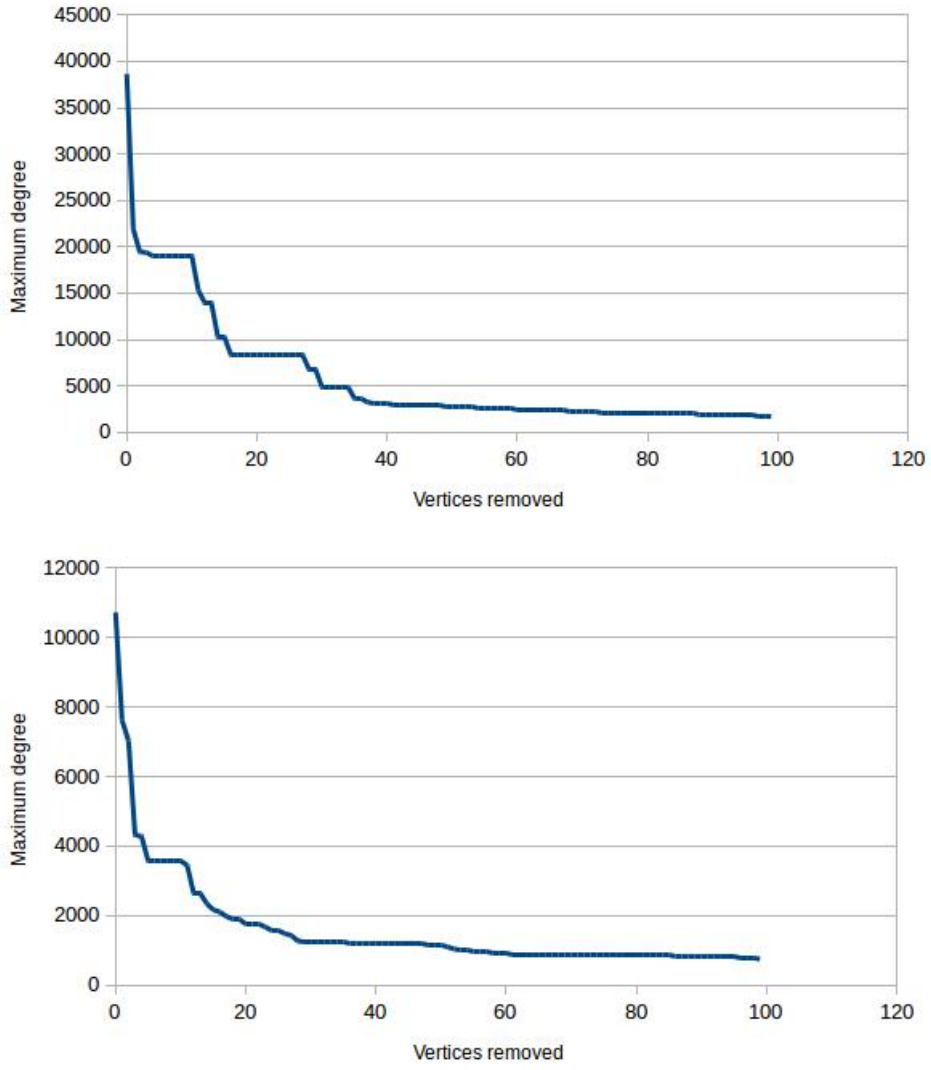

Figure 3.4: Plots of the maximum degree of web graphs representing the Stanford University (top) and University of Notre Dame (bottom) websites as vertices are greedily removed from the graph

3.6 An FPT Subgraph Counting Algorithm

Let $G = (V(G), E(G))$ be a graph of order n, and let $H = (V(H), E(H))$ be a graph of order m. We define the subgraph counting problem parameterised by m as follows.

SUBGRAPH COUNTING	
Input:	A graph $G = (V(G), E(G))$ of order n and a graph $H = (V(H), E(H))$ of order m.
Parameter:	m.
Question:	How many times does H appear as a subgraph of G?

In what follows, we describe an FPT algorithm for subgraph counting in classes of host

graphs with almost-bounded degree parameterised by the order of the pattern graph.

3.6.1 Algorithm Overview

Let $G = (V(G), E(G))$ be a graph of order n with at most c vertices with degree exceeding Δ. Let $H = (V(H), E(H))$ be a graph of order m. Our algorithm works by first considering each possible way to embed a subgraph of H into the high-degree part of G. For each such possibility, we then count the number of ways to assign the remaining vertices in $V(H)$ to the bounded degree part of G. This is done by splitting the remaining part of H into its connected components, and counting non-overlapping copies of these components in the bounded degree part of G with lists (determined by the assignment of a subset of $V(H)$ to the high-degree vertices in $V(G)$) via a careful inclusion-exclusion argument. We will see that splitting H up in this way allows us to take advantage of the almost-bounded degree structure of G in the runtime bound. This method so far counts the number of embeddings of H into G. To obtain the number of times that H appears as a subgraph of G, we count the number of automorphisms of H using a brute-force method, and then divide $\#Emb(H, G)$ by $\#Aut(H)$.

3.6.2 Algorithm

The following two lemmas bound the complexity of counting embeddings with lists of a connected graph in a bounded degree host graph. We will make use of these results when counting embeddings of each of the connected components. We note that both results are folklore, but we include a proof of each for completeness. Let $H = (V(H), E(H))$ be a graph. A *vertex ordering* of $V(H)$ is a bijective numbering $\sigma : V(H) \to \{1, \ldots, |V(H)|\}$, where the value of $\sigma(i)$ provides the position of v_i in the ordering. In the following lemma, we prove that there exists an ordering over the vertices of a connected graph H such that every vertex except the first is preceded (in the ordering) by one of its neighbours. Let $G = (V(G), E(G))$ be a graph with maximum degree Δ. Recall that in any embedding f of H into G, we require that $f(u)f(v) \in E(G)$ for any pair $u, v \in V(H)$ with $uv \in E(H)$. Hence, by assigning vertices from H to vertices in $V(G)$ in this order, we can restrict the number of potential embeddings that we need to consider.

Lemma 3.1. *Let $H = (V(H), E(H))$ be a connected graph with order m. We can construct a vertex ordering $\sigma : V(H) \to [m]$ of $V(H)$ such that every vertex except the first is preceded by one of its neighbours in the ordering in time $\mathcal{O}(m^3)$.*

Proof. We construct σ as follows. We may select the first vertex arbitrarily. We then add the remaining vertices in $V(H)$ one by one. At each stage, we select a vertex v from the

set of vertices which are not already in the (partial) ordering such that the ordering contains at least one neighbour of v. We can search for such a vertex in time $\mathcal{O}(m^2)$ by checking, for each available vertex, whether it has a neighbour among the vertices in the partial ordering. Suppose for a contradiction that there is no such vertex. It follows that the set of vertices in $V(H)$ which are contained in the (partial) ordering and those which are not, form disconnected components in the graph. Since we assumed that H is connected, we have a contradiction. Since there are at most m vertices in $V(H)$, it follows that we can construct the ordering in time $\mathcal{O}(m^3)$. □

The following lemma uses Lemma 3.1 to bound the time needed to count labelled embeddings of a connected graph into a graph with bounded degree.

Lemma 3.2. *Let $H = (\{v_1, \ldots, v_m\}, E(H))$ be a connected graph, and let $G = (V(G), E(G))$ be a graph of order n with maximum degree Δ. Let $V_{H,G} = \{V_{v_1,G}, \ldots, V_{v_m,G}\}$ be a set of m subsets of $V(G)$. We can count the number $\#Emb(H, G, V_{H,G})$ of embeddings of H into G with lists $V_{H,G}$ in time $\mathcal{O}(n^2 m \Delta^m)$.*

Proof. We count the number of embeddings of H into G with lists $V_{H,G}$ using a depth-first search. Specifically, each node in the search tree (excluding the root node) corresponds to a possible assignment of a vertex in $V(H)$ to a particular vertex in $V(G)$. The depth of the tree is at most one more than the size m of $V(H)$, and the number of embeddings of H into G with lists $V_{H,G}$ is equal to the number of leaves in the tree at depth $m+1$. By Lemma 3.1, we can construct an ordering $\sigma : V(H) \to [m]$ of $V(H)$ such that every vertex except the first is preceded by one of its neighbours in time $\mathcal{O}(m^3)$. The order in which we assign vertices from $V(H)$ along any branch of the search tree is then given by σ. Suppose that at some node of the search tree we assign a vertex v_H in $V(H)$ to an available vertex v_G from the list $V_{v_H,G}$. We must now update the lists of each unassigned neighbour of v_H to include only neighbours of v_G. We must also remove v_G from the lists of all unassigned vertices in $V(H)$. Since H has order m and G has order n, there are at most m lists to update, and each list has length at most n. Since G has degree at most Δ, it follows that v_G has at most Δ neighbours. Hence, we can remove all elements in the lists of unassigned neighbours of v_H which are not neighbours of v_G in time $\mathcal{O}(mn\Delta)$. We can remove v_G from the lists of all unassigned vertices in $V(H)$ in time $\mathcal{O}(mn)$. Hence, updating the lists takes time $\mathcal{O}(mn\Delta)$. Note that this updating lists operation occurs once at each non-root node of the search tree.

We now bound the number of nodes in our search tree. Since there are n vertices in G, it follows that there are at most n ways to assign the first vertex v in $V(H)$ (according to the ordering σ) to vertices from $V_{v,G}$. For each other vertex $v_H \in V(H)$, when we come to assign v_H, we will have already assigned at least one neighbour of v_H. Since G has maximum degree Δ, it follows that there are at most Δ elements remaining in the list $V_{v_H,G}$, and hence

there are at most Δ ways to assign v_H. Since H has order m, it follows that there are at most $n\Delta^{m-1}$ nodes in the search tree. It follows that we can count the number of embeddings of H into G with lists $V_{H,G}$ in time

$$\begin{aligned}&\mathcal{O}(m^3 + n\Delta^{m-1} \times (mn\Delta + ((m-1)\log\Delta + \log n))\\ &= \mathcal{O}(n^2 m\Delta^m).\end{aligned}$$

□

In the next lemmas, we describe how to count non-overlapping embeddings with lists of a set of connected components into a graph with bounded degree. We first describe how to count the total number of embeddings. Let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a set of disjoint connected graphs $C_i = (V(C_i), E(C_i))$, and let $G = (V(G), E(G))$ be a graph. For each $C_i \in \mathcal{C}$, let $V_{C_i,G}$ be a set of $|V(C_i)|$ subsets of $V(G)$ (one for each vertex in $V(C_i)$). Let $V_{\mathcal{C},G} = \{V_{C_1,G}, \ldots, V_{C_\ell,G}\}$. An *embedding of $\mathcal{C}$ into G with lists $V_{\mathcal{C},G}$* is a function $f : \bigcup_{i\in[\ell]} V(C_i) \to V(G)$ such that the restriction of f to each C_i is an embedding of C_i into G with lists $V_{C_i,G}$. Note that f is not necessarily an injection i.e. we do allow $f(u) = f(v)$ for a pair $u \in V(C_i)$ and $v \in V(C_j)$ with $i \neq j$. We denote the number of embeddings of $\mathcal{C}$ into G with lists $V_{\mathcal{C},G}$ by $\#Emb(\mathcal{C}, G, V_{\mathcal{C},G})$. In the following lemma, we show that the number of embeddings of $\mathcal{C}$ into G with lists $V_{\mathcal{C},G}$ is equal to the product, over all graphs $C_i \in \mathcal{C}$, of the number of embeddings of C_i into G with lists $V_{C_i,G}$.

Lemma 3.3. *Let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a set of ℓ disjoint connected graphs, and let $G = (V(G), E(G))$ be a graph. Let $V_{C_i,G}$ be a set of $|V(C_i)|$ subsets of $V(G)$ for each $i \in [\ell]$, and let $V_{\mathcal{C},G} = \{V_{C_1,G}, \ldots, V_{C_\ell,G}\}$. We have that*

$$\#Emb(\mathcal{C}, G, V_{\mathcal{C},G}) = \prod_{i\in[\ell]} \#Emb(C_i, G, V_{C_i,G}).$$

Proof. By definition, the value of $\#Emb(\mathcal{C}, G, V_{\mathcal{C},G})$ is equal to the number of functions from the set $\bigcup_{i\in[\ell]} V(C_i)$ to the set $V(G)$ which meet the definition of an embedding of C_i into G with lists $V_{C_i,G}$. Since $V(C_i) \cap V(C_j) = \varnothing$ for all pairs $i \neq j$, the result follows.

□

We now use the relationship described in Lemma 3.3 to bound the time needed to count embeddings with lists of $\mathcal{C}$ into G.

Lemma 3.4. *Let m be a positive integer and let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a set of ℓ disjoint connected graphs with $\sum_{i\in[\ell]} |V(C_i)| \leq m$ and $\ell \leq m$. Let $G = (V(G), E(G))$ be a graph of order n with maximum degree Δ. Let $V_{C_i,G}$ be a set of $|V(C_i)|$ subsets of $V(G)$ for each*

$i \in [\ell]$, and let $V_{\mathcal{C},G} = \{V_{C_1,G}, \ldots, V_{C_\ell,G}\}$. We can compute the value of $\#Emb(\mathcal{C}, G, V_{\mathcal{C},G})$ in time $\mathcal{O}(n^2m^2\Delta^m)$.

Proof. By Lemma 3.3, we have that

$$\#Emb(\mathcal{C}, G, V_{\mathcal{C},G}) = \prod_{i\in[\ell]} \#Emb(C_i, G, V_{C_i,G}).$$

Since $|V(C_i)| \leq m$ for each $i \in [\ell]$, it follows from Lemma 3.2 that we can compute the value of $\#Emb(C_i, G, V_{C_i,G})$ in time $\mathcal{O}(n^2m\Delta^m)$ for each $i \in [\ell]$. The value of $\#Emb(\mathcal{C}, G, V_{\mathcal{C},G})$ is at most n^m, and hence can be represented using at most $m \log n$ bits. Since $\ell \leq m$, it follows that we can compute the value of $\#Emb(\mathcal{C}, G, V_{\mathcal{C},G})$ in time

$$\begin{aligned}&\mathcal{O}(m \times (n^2m\Delta^m + (m \log n)^2))\\ &= \mathcal{O}(n^2m^2\Delta^m).\end{aligned}$$

□

We now describe how to count the number of overlapping embeddings (with lists) of the components. We can then subtract this value from the total number of embeddings to obtain the number of non-overlapping embeddings. We first define formally what it means for a set of components to overlap.

Let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a set of ℓ disjoint connected graphs, and let $G = (V(G), E(G))$ be a graph. Let $V_{C_i,G}$ be a set of $|V(C_i)|$ subsets of $V(G)$ for each $i \in [\ell]$, and let $V_{\mathcal{C},G} = \{V_{C_1,G}, \ldots, V_{C_\ell,G}\}$. Let f be an embedding of $\mathcal{C}$ into G with lists $V_{\mathcal{C},G}$. We say that f is an *overlapping embedding of $\mathcal{C}$ into G with lists $V_{\mathcal{C},G}$* if $f(u) = f(v)$ for some pair $u \in V(C_i)$ and $v \in V(C_j)$ with $i \neq j$. We denote the number of such embeddings by $\#Emb\text{-}overlap(\mathcal{C}, G, V_{\mathcal{C},G})$. We call f a *non-overlapping embedding of $\mathcal{C}$ into G with lists $V_{\mathcal{C},G}$* if there are no pairs of vertices $u \in V(C_i)$ and $v \in V(C_j)$ with $i \neq j$ such that $f(u) = f(v)$. The number of such embeddings is denoted by $\#Emb\text{-}none(\mathcal{C}, G, V_{\mathcal{C},G})$. We make the following observation.

Observation 3.5. Let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a set of disjoint connected graphs, and let $G = (V(G), E(G))$ be a graph. Let $V_{C_i,G}$ be a set of $|V(C_i)|$ subsets of $V(G)$ for each $C_i \in \mathcal{C}$, and let $V_{\mathcal{C},G} = \{V_{C_1,G}, \ldots, V_{C_\ell,G}\}$. We have that

$$\#Emb\text{-}none(\mathcal{C}, G, V_{\mathcal{C},G}) = \#Emb(\mathcal{C}, G, V_{\mathcal{C},G}) - \#Emb\text{-}overlap(\mathcal{C}, G, V_{\mathcal{C},G}).$$

Informally, we extract the number of non-overlapping embeddings of the components from the total number of embeddings by considering each way that the components can be "merged"

together to form a new set of connected components. We then subtract the number of overlapping embeddings of each set of these merged components from the total number of embeddings.

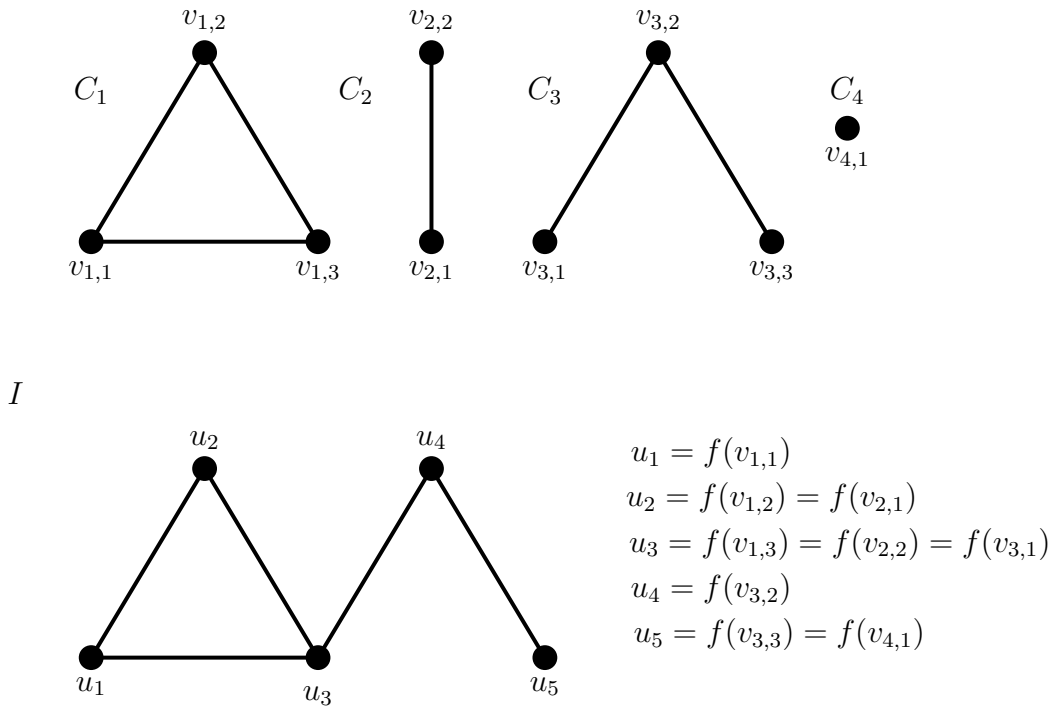

Figure 3.5: An example of an intersection (I, f) of graphs $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$.

Formally, let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a set of ℓ disjoint connected graphs. Let $V_{C_i,G}$ be a set of $|V(C_i)|$ subsets of $V(G)$ for each $i \in [\ell]$, and let $V_{\mathcal{C},G} = \{V_{C_1,G}, \ldots, V_{C_\ell,G}\}$. Let $\mathcal{P} = \{P_1, \ldots, P_{\ell'}\}$ be a partition of $\mathcal{C}$ into ℓ' sets for some $\ell' < \ell$. Note that we say that a partition $\mathcal{P} = \{P_1, \ldots, P_{\ell'}\}$ of $\mathcal{C}$ is *finer* than a partition $\mathcal{P}' = \{P'_1, \ldots, P'_{\ell''}\}$ if $\ell' > \ell''$ and, for each $i \in \{1, \ldots, \ell''\}$, we have that $P'_i = \bigcup_{j \in s} P_j$ for some subset s of $[\ell']$.

Let $\mathcal{P} = \{P_1, \ldots, P_{\ell'}\}$ be a partition of $\mathcal{C}$, and let $\{C_a, \ldots, C_b\}$ be the components in some part P_i of $\mathcal{P}$. An *intersection* of $\{C_a, \ldots, C_b\}$ is a pair (I, f) where $I = (V(I), E(I))$ is a connected graph, and $f : \bigcup_{i \in \{a,\ldots,b\}} V(C_i) \to V(I)$ is a function such that

- for each $i \in \{a, \ldots, b\}$, if $uv \in E(C_i)$ then $f(u)f(v) \in E(I)$, and
- if $uv \in E(I)$ then there exists some $i \in \{a, \ldots, b\}$ and some pair $u', v' \in V(C_i)$ such that $u' \in f^{-1}(u)$ and $v' \in f^{-1}(v)$, and $u'v' \in E(C_i)$.

Figure 3.5 illustrates an example of an intersection of a set of disjoint connected graphs. We define the set $V_{I,G}$ of subsets of $V(G)$ from $\{V_{C_a,G}, \ldots, V_{C_b,G}\}$ as follows. For each $v \in V(I)$, we set $V_{v,G} = \bigcap_{u \in f^{-1}(v)} V_{u,G}$. We say that two intersections (I, f) and (I', f') of $\{C_a, \ldots, C_b\}$ are equal if and only if $|V(I)| = |V(I')|$ and there exists a bijection $h : V(I) \to V(I')$ such that, for all $v \in \bigcup_{i \in \{a,\ldots,b\}} V(C_i)$, we have that $f'(v) = h(f(v))$.

Let (I_i, f_i) be an intersection of the graphs in P_i for each $P_i \in \mathcal{P}$. We call the set $\mathcal{I} = \{(I_1, f_1), \ldots, (I_{\ell'}, f_{\ell'})\}$ an *intersection set* of $\mathcal{C}$. By construction, the set of graphs in $\mathcal{I}$ are disjoint, and each graph is connected. We call ℓ' the *size* of the intersection set. Two intersection sets of $\mathcal{C}$ are equal if they contain exactly the same intersections.

In the following lemma, we describe an upper bound on the number of possible intersection sets of $\mathcal{C}$.

Lemma 3.6. *Let m be a positive integer, and let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a set of ℓ disjoint connected graphs with $\ell \leq m$ and $\sum_{i \in [\ell]} |V(C_i)| \leq m$. There are at most m^m intersection sets of $\mathcal{C}$.*

Proof. The number of intersection sets of $\mathcal{C}$ is at most the number of functions from the set of (at most m) vertices in $\bigcup_{i \in [\ell]} V(C_i)$ to a set of size m. It follows that the number of intersection sets is at most m^m. □

Let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a set of disjoint connected graphs, and let $G = (V(G), E(G))$ be a graph. Let $V_{C_i,G}$ be a set of $|V(C_i)|$ subsets of $V(G)$ for each $i \in [\ell]$, and let $V_{\mathcal{C},G} = \{V_{C_1,G}, \ldots, V_{C_\ell,G}\}$. In the following lemma, we describe a relationship between the number of overlapping embeddings of $\mathcal{C}$ into G with lists $V_{\mathcal{C},G}$, and the number of non-overlapping embeddings of the graphs in each intersection set $\mathcal{I}$ of $\mathcal{C}$ with lists $V_{\mathcal{I},G}$.

Lemma 3.7. *Let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a set of disjoint connected graphs, and let $G = (V(G), E(G))$ be a graph. Let $V_{C_i,G}$ be a set of $|V(C_i)|$ subsets of $V(G)$ for each $i \in [\ell]$, and let $V_{\mathcal{C},G} = \{V_{C_1,G}, \ldots, V_{C_\ell,G}\}$. We have that*

$$\#Emb\text{-}overlap(\mathcal{C}, G, V_{\mathcal{C},G}) = \sum_{\mathcal{I}=\{(I_1,f_1),\ldots,(I_{\ell'},f_{\ell'})\}} \#Emb\text{-}none(\{I_1, \ldots, I_{\ell'}\}, G, V_{\mathcal{I},G}).$$

Proof. In what follows, we describe a bijection between the set of overlapping embeddings with lists of $\mathcal{C}$ into G, and the set of non-overlapping embeddings with lists of the graphs in each of the intersection sets of $\mathcal{C}$ into G.

In the first direction, let g be an overlapping embedding of $\mathcal{C}$ into G with lists $V_{\mathcal{C},G}$. Let $\{I_1, \ldots, I_{\ell'}\}$ be the set of connected graphs formed from $\mathcal{C}$ and g as follows. Let $\mathcal{P} = \{P_1, \ldots, P_{\ell'}\}$ be the (unique) finest partition of $\mathcal{C}$ such that there are no pairs $u \in V(C_i)$ and $v \in V(C_j)$ with $C_i \in P_{i'}$ and $C_j \in P_{j'}$ such that $i' \neq j'$ and $g(u) = g(v)$. Since g is an overlapping embedding, it follows that $\ell' < \ell$. For each $i \in [\ell']$, we construct the intersection set (I_i, f_i) of the graphs in P_i from g as follows. Let $\{C_a, \ldots, C_b\}$ denote the set of graphs in P_i. Let t denote the size of the codomain of g for the subset $\bigcup_{i \in \{a,\ldots,b\}} V(C_i)$ of $\bigcup_{i \in [\ell]} V(C_i)$. Let $I_i = (V(I_i), E(I_i))$ be a connected graph on t vertices, and let $f_i : \bigcup_{i \in \{a,\ldots,b\}} V(C_i) \to V(I)$ be a function such that

- for each $j, k \in \{a, \ldots, b\}$, and each pair $u \in V(C_j)$ and $v \in V(C_k)$, we have that $g(u) = g(v)$ if and only if $f_i(u) = f_i(v)$, and
- for each $j \in \{a, \ldots, b\}$, if $uv \in E(C_j)$ then $f_i(u)f_i(v) \in E(I_i)$, and
- if $uv \in E(I_i)$ then there exists some $j \in \{a, \ldots, b\}$ and some pair $u', v' \in V(C_j)$ such that $u' \in f_i^{-1}(u)$ and $v' \in f_i^{-1}(v)$, and $u'v' \in E(C_j)$.

Observe that (I_i, f_i) is an intersection of $\{C_a, \ldots, C_b\}$. Moreover, (I_i, f_i) is the only intersection which meets these requirements. To construct the lists $V_{I_i,G}$ from $V_{\mathcal{C},G}$, we set $V_{v,G} = \bigcap_{u \in f_i^{-1}(v)} V_{u,G}$ for each $v \in V(I_i)$. Finally, we construct an embedding $h : V(I_i) \to V(G)$ of I_i into G for each $i \in [\ell']$ as follows. For each $v \in V(I_i)$, set $h_i(v) = g(u)$ for arbitrary $u \in f^{-1}(v)$ (we may choose arbitrarily since it follows from our construction that $g(u) = g(u')$ for any $u, u' \in f^{-1}(v)$). Let $\mathcal{I} = \{(I_1, f_1), \ldots, (I_{\ell'}, f_{\ell'})\}$ be the intersection set formed from the pairs (I_i, f_i), and let $V_{\mathcal{I},G}$ denote the set of lists $\{V_{I_1,G}, \ldots, V_{I_{\ell'},G}\}$. Let $h : \bigcup_{i \in [\ell']} V(I_i) \to V(G)$ be the embedding of the graphs in $\mathcal{I}$ into G with lists $V_{\mathcal{I},G}$ formed by applying h_i to $V(I_i)$ for each $i \in [\ell']$. It follows from our construction that h is a non-overlapping embedding of the graphs in $\mathcal{I}$ into G with lists $V_{\mathcal{I},G}$.

In the other direction, let $\mathcal{I} = \{(I_1, f_1), \ldots, (I_{\ell'}, f_{\ell'})\}$ be an intersection set of $\mathcal{C}$ for some $\ell' < \ell$, and let h be a non-overlapping embedding of the graphs $\{I_1, \ldots, I_{\ell'}\}$ into G with lists $V_{\mathcal{I},G}$. We construct an overlapping embedding g of $\mathcal{C}$ with lists $V_{\mathcal{C},G}$ from h as follows. For each $C_i \in \mathcal{C}$, let $j \in [\ell']$ be the value such that $f_j(v) = u$ for some $v \in V(C_i)$ and some $u \in V(I_j)$. For each $v \in V(C_i)$, set $g(v) = h(f_j(v))$. It follows directly from the definition of an intersection set that g is an overlapping embedding of $\mathcal{C}$ into G with lists $V_{\mathcal{C},G}$. The result follows. □

We say that an intersection set $\mathcal{I}' = \{(I'_1, f'_1), \ldots, (I'_{\ell'}, f'_{\ell''})\}$ of $\mathcal{C}$ is *contained* in another intersection set $\mathcal{I} = \{(I_1, f_i), \ldots, (I_\ell, f_{\ell'})\}$, written as $\mathcal{I}' \subset \mathcal{I}$, if $\ell'' < \ell$ and, for each $i \in [\ell']$, there exists some $j \in [\ell'']$ and a function $g_j : V(I_i) \to V(I'_j)$ such that for every $u \in V(I_i)$ and every $v \in f^{-1}(u)$ we have that $g_j(f_i(v)) = f'_j(v)$. Observe that the set $\{(I'_1, g_1), \ldots, (I'_{\ell'}, g_{\ell''})\}$ is an intersection set of the graphs $\{I_1, \ldots, I_{\ell'}\}$.

The following result describes how to obtain the number of non-overlapping embeddings of the graphs in an intersection set of $\mathcal{C}$ into G. In particular, we describe a relationship between the number of non-overlapping embeddings of the graphs in a particular intersection set $\mathcal{I}$ of $\mathcal{C}$, and the number of non-overlapping embeddings of the graphs in each intersection set $\mathcal{I}'$ of $\mathcal{C}$ contained in $\mathcal{I}$.

Lemma 3.8. *Let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a set of ℓ disjoint connected graphs, and let $G = (V(G), E(G))$ be a graph. Let $V_{C_i,G}$ be a set of $|V(C_i)|$ subsets of $V(G)$ for each $i \in [\ell]$, and let $V_{\mathcal{C},G} = \{V_{C_1,G}, \ldots, V_{C_\ell,G}\}$. Let $\mathcal{I} = \{(I_1, f_1), \ldots, (I_{\ell'}, f_{\ell'})\}$ be an intersection set of*

$\mathcal{C}$. We have that

$$\#Emb\text{-}none(\{I_1,\ldots,I_{\ell'}\},G,V_{\mathcal{I},G}) = \prod_{i\in[\ell']} \#Emb(I_i,G,V_{I_i,G}) - \sum_{\mathcal{I}'=\{(I'_1,f'_1),\ldots,(I'_{\ell''},f'_{\ell''})\}\subset\mathcal{I}} \#Emb\text{-}none(\{I'_1,\ldots,I'_{\ell''}\},G,V_{\mathcal{I}',G}).$$

Proof. It follows from Observation 3.5 that

$$\#Emb\text{-}none(\{I_1,\ldots,I_{\ell'}\},G,V_{\mathcal{I},G}) = \#Emb(\{I_1,\ldots,I_{\ell'}\},G,V_{\mathcal{I},G}) - \#Emb\text{-}overlap(\{I_1,\ldots,I_{\ell'}\},G,V_{\mathcal{I},G}).$$

By Lemma 3.3, we have that

$$\#Emb(\{I_1,\ldots,I_{\ell'}\},G,V_{\mathcal{I}}) = \prod_{i\in[\ell']} \#Emb(I_i,G,V_{I_i,G}).$$

Finally, it follows from Lemma 3.7 and from the definition of containment among intersection sets that

$$\#Emb\text{-}overlap(\{I_1,\ldots,I_{\ell'}\},G,V_{\mathcal{I},G}) = \sum_{\mathcal{I}'=\{(I'_1,f'_1),\ldots,(I'_{\ell''},f'_{\ell''})\}\subset\mathcal{I}} \#Emb\text{-}none(\{I'_1,\ldots,I'_{\ell''}\},G,V_{\mathcal{I}',G}).$$

The result follows. □

The following lemma uses the relationships described in Lemmas 3.7 and 3.8 to bound the time needed to count the number of overlapping embeddings of the components in $\mathcal{C}$.

Lemma 3.9. *Let m be a positive integer, and let $\mathcal{C} = \{C_1,\ldots,C_\ell\}$ be a set of ℓ graphs with $\ell \leq m$ and $\sum_{i\in[\ell]} |V(C_i)| \leq m$. Let $G = (V(G),E(G))$ be a graph of order n with maximum degree Δ. Let $V_{C_i,G}$ be a set of $|V(C_i)|$ subsets of $V(G)$ for each $i \in [\ell]$, and let $V_{\mathcal{C},G} = \{V_{C_1,G},\ldots,V_{C_\ell,G}\}$. We can compute the value of $\#Emb\text{-}overlap(\mathcal{C},G,V_{\mathcal{C},G})$ in time $\mathcal{O}(n^2m^{2m}\Delta^m)$.*

Proof. By Lemma 3.7, we have that

$$\#Emb\text{-}overlap(\mathcal{C},G,V_{\mathcal{C},G}) = \sum_{\mathcal{I}=\{(I_1,f_1),\ldots,(I_{\ell'},f_{\ell'})\}} \#Emb\text{-}none(\{I_1,\ldots,I_{\ell'}\},G,V_{\mathcal{I},G}). \tag{3.1}$$

Let $\mathcal{I} = \{(I_1, f_1), \ldots, (I_\ell, f_\ell)\}$ be an intersection set of $\mathcal{C}$. By Lemma 3.8, we have that

$$\#Emb\text{-}none(\{I_1, \ldots, I_{\ell'}\}, G, V_{\mathcal{I},G}) = \prod_{i \in [\ell']} \#Emb(I_i, G, V_{I_i,G}) - \sum_{\mathcal{I}' = \{(I'_1, f'_1), \ldots, (I'_{\ell''}, f'_{\ell''})\} \subset \mathcal{I}} \#Emb\text{-}none(\{I'_1, \ldots, I'_{\ell''}\}, G, V_{\mathcal{I}',G}). \tag{3.2}$$

By definition, each intersection set that is contained in $\mathcal{I}$ is smaller than $\mathcal{I}$. It follows that if we compute the number of non-overlapping embeddings for each intersection set in ascending order of the size of the intersection set, then we know the value of $\#Emb\text{-}none(\{I'_1, \ldots, I'_{\ell''}\}, G, V_{\mathcal{I}',G})$ for each $\mathcal{I}' \subset \mathcal{I}$ when we come to compute the value of $\#Emb\text{-}none(\{I_1, \ldots, I_{\ell'}\}, G, V_{\mathcal{I},G})$. Note that the value of both the sum and the product in (3.2) have value at most n^m each since they are each at most the number of embeddings of $\mathcal{C}$ in G. It follows from the definition of an intersection set that $\sum_{i \in [\ell']} V(I_i) < \sum_{j \in [\ell]} V(C_i)$. Hence, it follows from Lemma 3.4 that we can compute the value of the product in (3.2) in time

$$\mathcal{O}(n^2 m^2 \Delta^m). \tag{3.3}$$

It follows from Lemma 3.6 that there are at most m^m intersection sets contained in $\mathcal{I}$. It follows that once we know the value of $\#Emb\text{-}none(\{I'_1, \ldots, I'_{\ell''}\}, G, V_{\mathcal{I}',G})$ for each $\mathcal{I}' \subset \mathcal{I}$, then we can compute the value of the sum in (3.2) in time

$$\begin{aligned} &\mathcal{O}(m^m \times (m \log n)) \\ &= \mathcal{O}(m^{m+1} \log n). \end{aligned} \tag{3.4}$$

It follows from (3.3) and (3.4) that, once we know the value of $\#Emb\text{-}none(\{I'_1, \ldots, I'_{\ell''}\}, G, V_{\mathcal{I}',G})$ for each $\mathcal{I}' \subset \mathcal{I}$, we can compute the value of $\#Emb\text{-}none(\{I_1, \ldots, I_{\ell'}\}, G, V_{\mathcal{I},G})$ in time

$$\begin{aligned} &\mathcal{O}(n^2 m^2 \Delta^m + m^{m+1} \log n + m \log n) \\ &= \mathcal{O}(n^2 m^m \Delta^m). \end{aligned} \tag{3.5}$$

Finally, it remains to bound the time needed to compute the value of the sum in (3.1). By definition, the value of the sum in (3.1) is at most n^m. Hence, it follows from (3.5) that we

can compute the value of the sum in time

$$\begin{aligned}&\mathcal{O}(m^m \times (n^2 m^m \Delta^m + m \log n))\\ &= \mathcal{O}(n^2 m^{2m} \Delta^m).\end{aligned}$$

The result follows. □

Using Lemmas 3.5 and 3.9, we can now bound the time needed to compute the number of non-overlapping embeddings of the components.

Lemma 3.10. *Let m be a positive integer, and let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a set of ℓ graphs with $\ell \leq m$ and $\sum_{i \in [\ell]} |V(C_i)| \leq m$. Let $G = (V(G), E(G))$ be a graph of order n with maximum degree Δ. Let $V_{C_i,G}$ be a set of $|V(C_i)|$ subsets of $V(G)$ for each $i \in [\ell]$, and let $V_{\mathcal{C},G} = \{V_{C_1,G}, \ldots, V_{C_\ell,G}\}$. We can compute the value of $\#Emb\text{-}none(\mathcal{C}, G, V_{\mathcal{C},G})$ in time $\mathcal{O}(n^2 m^{2m} \Delta^m)$.*

Proof. By Observation 3.5, we have that

$$\#Emb\text{-}none(\mathcal{C}, G, V_{\mathcal{C},G}) = \#Emb(\mathcal{C}, G, V_{\mathcal{C},G}) - \#Emb\text{-}overlap(\mathcal{C}, G, V_{\mathcal{C},G}).$$

It follows from Lemma 3.4 that we can compute the value of $\#Emb(\mathcal{C}, G, V_{\mathcal{C},G})$ in time $\mathcal{O}(n^2 m^2 \Delta^m)$. By definition, the value of $\#Emb(\mathcal{C}, G, V_{\mathcal{C},G})$ is at most n^m. It follows from Lemma 3.9 that we can compute the value of $\#Emb\text{-}overlap(\mathcal{C}, G, V_{\mathcal{C},G})$ in time $\mathcal{O}(n^2 m^{2m} \Delta^m)$. The value of $\#Emb\text{-}overlap(\mathcal{C}, G, V_{\mathcal{C},G})$ is also at most n^m. It follows that we can compute the value of $\#Emb\text{-}none(\mathcal{C}, G, V_{\mathcal{C},G})$ in time

$$\begin{aligned}&\mathcal{O}(n^2 m^2 \Delta^m + n^2 m^{2m} \Delta^m + m \log n)\\ &= \mathcal{O}(n^2 m^{2m} \Delta^m).\end{aligned}$$

□

We are now ready to prove our main result.

Theorem 3.11. *Let $H = (V(H), E(H))$ be a graph of order m, and let $G = (V(G), E(G))$ be a graph of order n which contains at most c vertices with degree exceeding Δ. We can count the number of times that H appears as a subgraph of G in time $\mathcal{O}(c^c n^2 m^{2m+c} \Delta^m)$.*

Proof. We first count the number of embeddings of H into G. This is achieved by taking the sum, over each subset $V_c(H)$ of $V(H)$ (including the empty subset) with size at most c, of the number of embeddings of H into G in which exactly those vertices in $V_c(H)$ are

assigned to the set of vertices of $V(G)$ with degree exceeding Δ. We then divide this value by $\#Aut(H)$ to obtain the number of times that H appears as a subgraph of G.

We will use $\mathcal{V}_c(H)$ to denote the set of all subsets $V_c(H)$ of $V(H)$ with size at most c. Let $V_c(H)$ be a set in $\mathcal{V}_c(H)$. We will use $\#Emb(V_c(H), H, G)$ to denote the number of embeddings of H into G in which exactly those vertices in $V_c(H)$ are assigned to the set of vertices of $V(G)$ with degree exceeding Δ. We have that

$$\#Emb(H, G) = \sum_{V_c(H) \in \mathcal{V}_c(H)} \#Emb(V_c(H), H, G). \tag{3.6}$$

Let $V_c(H)$ be a subset of $V(H)$ of size at most c, and let $V_c(G)$ denote the set of at most c vertices in $V(G)$ with degree exceeding Δ. We denote the subgraph of H formed from the subset $V(H) \setminus V_c(H)$ of $V(H)$ and all edges in $E(H)$ joining pairs of vertices in $V(H) \setminus V_c(H)$ by $H \setminus V_c(H)$. Similarly, we denote the subgraph of G containing the vertices in $V(G) \setminus V_c(G)$ and all corresponding edges from $E(G)$ by $G \setminus V_c(G)$. We count the number of embeddings of H into G such that exactly the vertices in $V_c(H)$ are assigned to vertices in $V_c(G)$ as follows. For each assignment of $V_c(H)$ to $V_c(G)$ such that pairs of neighbours in $V_c(H)$ are assigned to pairs of neighbours in $V_c(G)$, we count the number of embeddings of $H \setminus V_c(H)$ into the bounded degree subgraph $G \setminus V_c(G)$ of G. In particular, we must count the number of embeddings of $H \setminus V_c(H)$ into $G \setminus V_c(G)$ with lists, where the lists are determined by the assignment of $V_c(H)$ to $V_c(G)$. The value of $\#Emb(V_c(H), H, G)$ is then equal to the sum, over each possible assignment of $V_c(H)$ to $V_c(G)$, of the number of embeddings with lists of $H \setminus V_c(H)$ into $G \setminus V_c(G)$.

Let $f : V_c(H) \to V_c(G)$ be an assignment of $V_c(H)$ to $V_c(G)$. Since $|V_c(H)| \leq c$ and $|V_c(G)| \leq c$, we can check that all pairs of neighbours in $V_c(H)$ are assigned to pairs of neighbours in $V_c(G)$ in time $\mathcal{O}(c^2)$. Moreover, there are at most c^c such assignments. Given f, we must now count the number of ways to embed the remaining part $H \setminus V_c(H)$ of H into the remaining part $G \setminus V_c(G)$ of G. We first edit the lists $V_{v,G\setminus V_c(G)}$ for each $v \in V(H) \setminus V_c(H)$ so that pairs of neighbours in $V_c(H)$ and $V(H) \setminus V_c(H)$ are matched only to pairs of neighbours from $V_c(G)$ and $V(G) \setminus V_c(G)$ respectively. For each $v \in V(H) \setminus V_c(H)$, let U_v denote the set of neighbours of v in $V_c(H)$. For each $u \in U_v$, let $U_{f(u)}$ denote the neighbours of $f(u)$ in $V(G) \setminus V_c(G)$. We set $V_{v,G\setminus V_c(G)} = \bigcap_{u \in U_v} U_{f(u)}$. Let $V_{H\setminus V_c(H),G\setminus V_c(G)}$ denote the set containing the subsets $V_{v,G\setminus V_c(G)}$ for each $v \in V(H) \setminus V_c(H)$. Note that the graph $H \setminus V_c(H)$ may not be connected. Thus, in order to take advantage of the degree bound Δ of the vertices in $V(G) \setminus V_c(G)$, we count embeddings of $H \setminus V_c(H)$ into $G \setminus V_c(G)$ by splitting $H \setminus V_c(H)$ into its connected components $\mathcal{C} = \{C_1, \ldots, C_\ell\}$, and counting non-overlapping embeddings of these components into $G \setminus V_c(G)$ with lists $V_{\mathcal{C},G\setminus V_c(G)}$. Observe that since H contains at most m vertices, we have that $\sum_{i \in [\ell]} V(C_i) \leq m$ and $\ell \leq m$. It

follows from Lemma 3.10 that we count non-overlapping embeddings of $\mathcal{C}$ into $G \setminus V_c(G)$ with lists $V_{\mathcal{C},G\setminus V_c(G)}$ in time $\mathcal{O}(n^2 m^{2m}\Delta^m)$. The number of such embeddings is at most n^m. It follows that we can compute the value of $\#Emb(V_c(H), H, G)$ in time

$$\begin{aligned}&\mathcal{O}(c^c \times (c^2 + n^2 m^{2m}\Delta^m + m\log n))\\ &= \mathcal{O}(c^c n^2 m^{2m}\Delta^m).\end{aligned} \tag{3.7}$$

The number of ways to select a subset of size at most c from a set of size m is at most m^c. It follows that $|\mathcal{V}_c(H)| \leq m^c$. It then follows from (3.6) and (3.7) that we can compute the value of $\#Emb(H, G)$ in time

$$\begin{aligned}&\mathcal{O}(m^c \times (c^c n^2 m^{2m}\Delta^m + m\log n))\\ &= \mathcal{O}(c^c n^2 m^{2m+c}\Delta^m).\end{aligned}$$

The brute-force approach to counting automorphisms of H takes time $\mathcal{O}(m^m)$. The total number of embeddings of H into G is at most n^m, while the number of automorphisms of H is at most m^m. It follows that we can compute the value of $\#Sub(H, G)$ in time

$$\begin{aligned}&\mathcal{O}(c^c n^2 m^{2m+c}\Delta^m + m^m + m\log n \cdot m\log m)\\ &= \mathcal{O}(c^c n^2 m^{2m+c}\Delta^m).\end{aligned}$$

□

3.7 Remarks and Open Problems

In this chapter, we described an FPT algorithm for subgraph counting in host graphs with almost-bounded degree parameterised by the order of the pattern graph. We note that the same runtime bound applies when the host graph G can be partitioned into a graph with order at most c and a graph with maximum degree Δ. Note that in this setting the subgraph with bounded degree Δ may contain vertices with degree exceeding Δ in G. Many applications of subgraph counting [50, 51, 52] involve comparing large networks by analysing the frequency with which all k-vertex connected pattern graphs appear in each network for small values of k. Since our algorithm does not place any restrictions on the structure of the pattern graph, we expect that our algorithm could be usefully applied in such settings.

A variant of the subgraph counting problem asks for the number of induced copies of a pattern graph in a host graph. It seems highly likely that our algorithm can be extended with some small modifications to count induced subgraphs. Particular care must be taken to check that the method of counting non-overlapping embeddings of the components in our

algorithm carries over into the induced setting.

In Section 3.5, we presented examples of real-world networks with relatively few high-degree vertices. We note that in each of these networks, the degree drop-off seen when removing the high-degree vertices is insufficient for our algorithm to be practical. However, the structure observed in these networks suggests that there may exist real-world networks with the necessary structure for practical subgraph counting using our algorithm.

Chapter 4

Counting Stable Matchings

4.1 Motivation

The problem of counting stable matchings first appeared in a monograph by Donald Knuth on stable marriage in 1976 [53]. Irving and Leather [12] have since shown that counting solutions to even the most basic stable matching problems is computationally hard in general. In light of this, it is natural to ask whether efficient algorithms exist for counting solutions to parameterised variants of stable matching problems. In many practical applications of stable matching, it is reasonable to assume that agents who are deemed as "similar" in some important way may be regarded as equally desirable by their available partners. For instance, consider the problem of matching university graduates to available graduate jobs. Here the type of the job is defined by its field (software engineering, banking, teaching), and the type of the graduate is defined by the subject which they studied. Firms hiring for software engineering roles typically most prefer graduates in computer science, but also have a strong preference for graduates in subjects such as mathematics or physics. Meanwhile, graduates in mathematics are likely to prefer graduate jobs (accountancy, software engineering, finance) that require their logical reasoning skills.

In [11], Meeks and Rastegari introduce the notion of agent "types", where the type of an agent determines his/her preferences, and agents of the same type are considered equally desirable by their available partners. It is shown that a number of important stable matching problems which are computationally hard in general are efficiently solvable when we parameterise by the number of agent types. Here we extend the work of Meeks and Rastegari in the computational counting setting. We will see that, as in the decision setting, parameterising by the number of agent types creates tractability for several key stable matching problems.

4.2 Definitions and Notation

The *stable marriage problem* (SM) consists of a set of men and a set of women, each of whom has a strict preference ordering over the set of agents of the opposite gender. We typically use n to denote the number of agents in an instance of stable marriage. Given an instance I of SM, a *matching* M admitted by I is an assignment of the agents into man-woman pairs. The size of a matching M, denoted by $|M|$, is equal to the number of man-woman pairs in M. A *complete matching* is a matching containing all available agents. We write $M(w) = m$ to mean that woman w is matched to man m in the matching M. We may also write this as $(m, w) \in M$. If woman w has no partner in M, we write $M(w) = \varnothing$. We say that woman w strictly prefers man m_1 to m_2, written $m_1 \succ_w m_2$, if m_1 appears earlier in the preference list of w than m_2. A *blocking pair* in M is a pair of agents who would strictly prefer to be matched together (according to their preference lists) than with the respective partners assigned to them in the matching. A *weakly stable matching* is a matching containing no blocking pairs. Unless otherwise specified, we shall use the term "stable matching" to mean a weakly stable matching. Given an instance I of SM, the stable marriage problem asks whether there exists a stable matching of the agents in I.

The *stable roommates problem* (SR) is a generalisation of SM consisting of a single set of n agents, each of whom ranks all other agents in strict order of preference. In this setting, a stable matching is a partition of the set of agents into pairs such that no two agents would (strictly) prefer to be matched together than with their assigned partners.

Hospitals/residents (HR) is a many-one generalisation of stable marriage in which agents are labelled as either "residents" (junior doctors) or hospitals. In this setting, many residents may be assigned to the same hospital subject to the capacity constraints of the hospital. We denote the capacity of hospital h by $q(h)$, and we assume without loss of generality that $q(h) > 0$ for all h. In this setting, agents on both sides may declare a subset of their available partners as unacceptable matches. Let I be an instance of HR. We say that a matching M of the agents in I is stable if there is no hospital/resident pair (h, r) who find each other acceptable such that r is unmatched or prefers h to $M(r)$, and either h has spare capacity or else prefers r to the least desirable resident assigned to h under M.

A relaxation of agents' preference lists allows agents to express *indifference* between available partners. We say that agent a is indifferent between agents b and c, written $b \simeq_a c$, if a finds b and c equally desirable as partners. We write $b \succeq_a c$ if a either strictly prefers b to c or is indifferent between them. If a is indifferent between b and c then we may say that b and c are *tied* in the preference list of a. If agents b and c are tied in the preference list of a, we may write this as $\{b, c\}$ in the preference list (this notation extends in the obvious way to include ties involving more than two agents). Note that preference lists which contain ties are not considered strict. We refer to the variant of SM which allows agents to express

indifference between their available partners as *stable matching with ties* (SMT). Equivalent generalisations of stable roommates and hospitals/residents are denoted by SRT and HRT respectively.

In the presence of ties, we may define two additional notions of stability. Let I be an instance of SMT. We say that a matching M admitted by I is *strongly stable* if there is no pair of agents x and y such that x strictly prefers y to their assigned partner under M, and y is indifferent between x and their partner in the matching. A matching of the agents in I is *super-stable* if there is no pair of agents each of whom each either strictly prefers the other to the partner he/she has been assigned or is indifferent between the two candidates. Observe that any super-stable matching is also strongly stable, and any strongly stable matching is weakly stable.

A second relaxation of the standard stable matching model allows agents to declare some of their available partners as unacceptable in any matching i.e. they would prefer to be unmatched than to be matched with such a partner. If an agent declares one or more other agents as unacceptable, then we say that the agent's preference list is *incomplete*. Recall that the standard hospitals/residents problem already allows for incomplete preference lists. The generalisation of stable marriage which allows incomplete preference lists is referred to as *stable marriage with incomplete preference lists* (SMI). The equivalent generalisation in the stable roommates setting is known as *stable roommates with incomplete preference lists* (SRI). We refer to the relaxation of stable marriage which allows for both ties and incomplete preference lists by SMTI. Equivalent abbreviations for the hospitals/residents and stable roommates problems in this setting are HRT and SRTI respectively.

In some stable matching problems, there may exist problem instances that admit stable matchings of different sizes. An example is the stable marriage problem with ties and incomplete preference lists. In such settings, we may wish to find the maximum cardinality stable matching. We denote the problem of finding a stable matching of maximum cardinality admitted by an instance of SMTI by MAX SMTI. Equivalent abbreviations are used when we seek a maximum cardinality stable matching in other settings.

In typed variations of stable matching problems, we say that agents are of the same *type* if their preference lists are identical and all available partners are indifferent between them. We write $\text{type}(x) = i$ to mean that agent x has type i. In this setting, preference lists are declared by agent types rather than by individual agents. We write $j \succ_i \ell$, or equivalently $\ell \prec_i j$, if agents of type i strictly prefer agents of type j to agents of type ℓ. We write $j \simeq_i \ell$ to denote that agents of type i are indifferent between agents of type j and agents of type ℓ, and we write $j \succeq_i \ell$ if agents of type i either strictly prefer agents of type j to those of type ℓ or are indifferent between the two types. If types i and j are tied in the preference list of type ℓ, then we may write this as $\{i, j\}$ in the preference list of type ℓ.

A typed instance of stable marriage contains a set of n men and women, and a set of k agent types $1, \ldots, k$ such that each agent belongs to exactly one type. In this setting, each agent type has a preference list over the set of types of agents of the opposite gender. Notice that if we assign each individual agent a different type then we arrive at the standard stable marriage problem. Let I be a typed instance of stable marriage with k agent types. A matching M admitted by I is weakly stable if there is no man/woman pair (m, w) with types (i, j) such that $(m, w) \notin M$ and $j \succ_i \text{type}(M(m))$ and $i \succ_j \text{type}(M(w))$. Analogous definitions are given for strong and super-stability in this setting. Typed instances of stable roommates and hospitals/residents are defined similarly. We prefix stable matching problems with "TYPED" to indicate that problem instances are expected to be typed. For example, we denote the problem of finding a maximum size stable matching in a typed instance of SMTI by TYPED MAX SMTI.

Any instance I of a stable matching problem can be represented using a graph $G = (V(G), E(G))$ - sometimes called the *acceptability graph* of I - where the vertices in $V(G)$ represent the set of agents in I, and the edges in $E(G)$ represent acceptability between pairs of agents. A matching of the agents in I corresponds to a matching in G. A complete matching of the agents in I corresponds to a perfect matching in G. We will require the following definitions on structural properties of graphs. Let $G = (V(G), E(G))$ be a graph. Two vertices $u, v \in V(G)$ are said to have the same *type* if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. We say that G has *neighbourhood diversity* w [54] if w is the smallest number such that there exists a partition of $V(G)$ into w sets such that all the vertices in a set have the same type. The neighbourhood diversity of a graph can be computed in polynomial time [54]. Each graph $G = (V(G), E(G))$ on n vertices has an associated $n \times n$ (0,1)-matrix called an *adjacency matrix* $A = (a_{i,j})$, where $a_{i,j} = 1$ if $v_i v_j \in E(G)$, and $a_{i,j} = 0$ otherwise. The *rank* of a matrix is defined as the maximum number of linearly independent rows in the matrix.

4.3 Literature Review

In this section, we survey the literature on stable matching problems. A comprehensive study of the structural and algorithmic aspects of stable matching is provided by Gusfield and Irving in [55]. A more recent monograph on the subject is given by Manlove in [56].

In Sections 4.3.1 to 4.3.4, we survey the literature on finding (weakly) stable matchings in instances of stable matching problems with and without ties and incomplete preference lists. In Section 4.3.5, we cover results on finding super-stable and strongly stable matchings. In Section 4.3.6, we survey approximation results for hard variants of stable matching. Sections 4.3.7 and 4.3.8 cover results on parameterised stable matching problems. In particular, Section 4.3.8 contains results on typed stable matching problems. In Section 4.3.9, we sur-

vey the small number of results on stable matching problems in which preference lists are restricted in other ways. Finally, in Section 4.3.10, we examine what is known about counting stable matchings.

4.3.1 Stable Matching

Matching problems first appeared in the literature in a seminal paper of David Gale and Lloyd Shapley in 1962 [8]. In their paper, the authors describe an $\mathcal{O}(n^2)$ algorithm - now known as the *Gale-Shapley algorithm* - for finding a stable matching admitted by an instance of SM containing at most n agents. Note that an instance of SM always admits at least one stable matching [8]. In addition, since preference lists are complete, every stable matching is complete. In their paper [8], Gale and Shapley also introduce the stable roommates problem, and describe an example instance which does not admit a stable matching. A paper due to Irving [9] describes an $\mathcal{O}(n^2)$ algorithm for finding a stable matching admitted by an instance of SR, or else reporting that no such matching exists. An $\mathcal{O}(n \log^3 n)$ parallel algorithm for finding a stable matching (if one exists) in an instance of SR is given by Feder et al. in [57].

4.3.2 Stable Matching with Ties

Every instance of SMT admits at least one stable matching [58]. Given an instance of SMT, a stable matching can be obtained in polynomial time by breaking ties arbitrarily and then applying the Gale-Shapley algorithm [58]. In [59], Ronn showed that the problem of deciding whether an instance or SRT admits a stable matching is NP-complete.

4.3.3 Stable Matching with Incomplete Preference Lists

In their seminal paper, Gale and Shapley also show that every instance of HR (and hence SMI) admits at least one stable matching [8]. In their book, Gusfield and Irving [55] show that a polynomial-time variation of the Gale-Shapley algorithm finds a stable matching admitted by an instance of SMI. A well-known "cloning technique" [55] can be used to transform an instance of HR into an instance of SMI in polynomial time. It follows that the extended Gale-Shapley algorithm due to Gusfield and Irving can also be used to find a stable matching admitted by an instance of HR.

The Rural Hospitals Theorem [60, 6, 61] states that for any instance of HR, the same set of residents are assigned in all stable matchings, each hospital is assigned the same number of residents in all stable matchings, and any hospital that has spare capacity in one stable matching is assigned exactly the same set of residents in all stable matchings. It follows

that the set of stable matchings admitted by an instance of HR (or SMI) all have the same cardinality.

We saw that an instance of SRI need not admit a stable matching [8]. However, if a stable matching exists, then all stable matchings have the same cardinality and match the same set of agents [55]. In [55], Gusfield and Irving generalised Irving's polynomial-time algorithm for finding a stable matching (or reporting that none exists) in an instance of SR [9] to the setting with incomplete lists.

4.3.4 Stable Matching with Ties and Incomplete Preference Lists

An instance of SMTI always admits a stable matching, and one can be found in polynomial time using an extension of the Gale-Shapley algorithm [62]. The cloning technique described previously can be used to transform an instance of HRT into an instance of SMTI. It follows that we can find a stable matching admitted by an instance of HRT in polynomial time. It is shown in [59] that the problem of deciding whether an instance of SRTI admits a stable matching is NP-hard.

An instance of SMTI (and hence both HRT and SRTI) may admit stable matchings of different sizes [63]. As such, it is of interest to find a maximum cardinality stable matching in this setting. The problem of finding a maximum size stable matching admitted by an instance of SMTI is NP-hard even in the severely restricted case where ties are present in the men's preference lists only, the ties are all at the end of lists, there is at most one tie per list, and each tie is of length 2 [10]. It follows that MAX SRTI and MAX HRT are also NP-hard in general.

4.3.5 Super-Stable and Strongly Stable Matchings

There exist instances of SMT such that no super-stable or strongly stable matching exists [58]. Given any instance of HRT such that a super-stable matching exists, the Rural Hospitals Theorem applies to the set of all super-stable matchings [64]. In [65], it was shown that all strongly stable matchings admitted by an instance of HRT are of equal cardinality. It follows from [66] that for instances of SRTI (and hence SMTI) which admit a super-stable matching, all super-stable matchings match exactly the same set of agents. In [67], it is shown that the same is true under strong stability.

The problem of finding a strongly stable matching admitted by an instance of SRTI (or reporting that none exists) is solvable in time $\mathcal{O}(nm)$ [68] where n and m are the number of vertices and edges respectively in the acceptability graph constructed from the problem instance. It was shown in [66] that the problem of finding a super-stable matching (if one

exists) in an instance of SRTI is solvable in time $\mathcal{O}(m)$. It follows that the problems of finding a (maximum-cardinality) strongly stable or super-stable matching admitted by an instance of SMTI are solvable in time $\mathcal{O}(nm)$ and $\mathcal{O}(m)$ respectively. An $\mathcal{O}(m)$ algorithm for finding a super-stable matching or reporting that none exists for an instance of HRT was given by Irving et al. in [64]. The same authors describe an $\mathcal{O}(nm)$ algorithm in [69] for finding a strongly stable matching admitted by an instance of HRT.

4.3.6 Approximation Algorithms for Stable Matching

Given that many of the key stable matching problems are NP-hard, it is natural to ask about the existence of efficient approximation algorithms for these problems. Given an instance of MAX HRT, two stable matchings can differ in size by a factor of at most two [10]. As a consequence, a variant [10] of the Gale-Shapley algorithm can be used to approximate MAX HRT within a factor of two. In [70], it is shown that MAX SMTI cannot be approximated within a factor of less than 21/19 unless P = NP, even if ties occur only in the men's preference lists [70]. In [71], McDermid proved the existence of a 3/2-approximation algorithm for MAX SMTI. A 3/2-approximation algorithm for MAX HRT is described in [72]. In [66], Irving and Manlove prove that MAX SRTI is also approximable within a factor of two.

4.3.7 Parameterised Stable Matching Problems

An alternative approach to tackling NP-hard stable matching problems is to search for parameterisations of these problems which yield efficient parameterised algorithms. In [73], Adil et al. prove that MAX SMTI is in FPT with respect to solution size, and that MAX SRTI belongs to XP parameterised by the treewidth [74] of the acceptability graph. Marx and Schlotter [75] have established that MAX SMTI is in FPT parameterised by the total length of the ties in agents' preference lists. On the other hand, they showed that MAX SMTI is W[1]-hard parameterised by the number of ties, even if the ties are only on one side.

In [76], Bredereck et al. study the parameterised complexity of MAX SRTI with respect to several structural parameters of the acceptability graph. They show that MAX SRTI is W[1]-hard when the acceptability graph has bounded treedepth [77], bounded tree-cut width [78], or disjoint paths modulator number [76]. On the other hand, it is shown that MAX SRTI is in FPT parameterised by the vertex cover number or feedback edge set number [76] of the acceptability graph. It is also shown that SRTI belongs to FPT parameterised by the tree-cut width of the acceptability graph, but is W[1]-hard parameterised by the treedepth or disjoint paths modulator number of the acceptability graph. In [79] Gupta et al. showed that MAX SMTI is W[1]-hard parameterised by the treewidth of the acceptability graph.

4.3.8 Typed Stable Matching Problems

Meeks and Rastegari [11] consider the parameterised complexity of stable matching problems when the set of agents can partitioned into a small number of types. In their paper, Meeks and Rastegari show that TYPED MAX HRT and TYPED MAX SRTI (and thus TYPED MAX SMTI) belong to FPT parameterised by the number of agent types. The authors also consider several generalisations of the notion of agent types. The first generalisation allows agents of the same type to order their preference lists differently, as long as the set of available partners of the same type occur consecutively. Under this generalisation, each of TYPED MAX SRTI, TYPED MAX SMTI and TYPED MAX HRT remain in FPT parameterised by the number of types. Moreover, if we enforce that preferences over types are strict, then each of these problems become polynomial-time solvable.

A second generalisation of agent types examined in the paper allows individual agents to rank a small number of "exceptional" candidates in their preference list without regard to the candidate's type. In this setting, TYPED MAX SMTI belongs to FPT when each individual agent may place at most one exceptional candidate at the top of their preference list. However, the problem becomes NP-hard if agents are allowed to place two or more exceptional candidates anywhere in their preference lists when the number of types in the instance is bounded by a constant. It follows that the problem is not in XP unless $\mathsf{P} = \mathsf{NP}$. In addition, both TYPED MAX SRTI and TYPED MAX HRT are also NP-hard in this setting.

4.3.9 Restrictions on Preference Lists

Models of stable matching in which groups of agents have very similar preferences were first considered by Scott [67], who introduced a variant of the stable marriage problem called *stable marriage with master lists*. In this setting, all agents with the same gender share the same "master list" of preferences except for unacceptable partners, which may be declared by individual agents. This variant of stable marriage has also been studied by Irving et al. in [80]. It is shown that MAX SMTI and many other variants of stable marriage remain NP-hard in this setting even under severe restrictions [67, 80]. On the other hand, several variants of stable marriage that are solvable in polynomial time in the original setting can be solved by faster or simpler algorithms in the setting with master lists.

4.3.10 Counting Stable Matchings

In [12], Irving and Leather show that the problem of counting the number of (weakly) stable matchings admitted by an instance of stable marriage is #P-complete with respect to parsimonious reductions. It follows that there can be no polynomial-time algorithm which

counts the number of solutions to an instance of SM in general unless P = NP. Since counting matchings exactly is hard, it is natural to ask whether we can approximate the number of stable matchings efficiently. However, it has been shown that the problem of counting solutions to an instance of SM belongs to a class of problems for which we do not expect to find efficient approximation algorithms [81, 82]. In particular, we do not expect to find an FPRAS (defined in Section 2.3.4) for approximately counting stable matchings in general. In [83], Bhatnagar et al. introduce a model of stable matching whereby agents' preferences are formed using the k-attribute model, in which agents assign weights to a list of k attributes. Candidates are then ranked according to their strength in the most important (highly weighted) attributes. The authors of [81, 84] show that even approximate counting is hard in this setting, subject to appropriate complexity-theoretic assumptions.

4.4 Contributions

In Section 4.5, we describe an XP algorithm for counting the number of weakly stable matchings admitted by an instance of TYPED SMTI parameterised by the number of agent types needed to describe the instance. Our algorithm broadly involves reducing our problem to the problem of counting perfect matchings in balanced bipartite graphs with bounded neighbourhood diversity. That the latter problem belongs to XP follows from a result due to Barvinok [85] on the complexity of computing the "permanent" of a square matrix with bounded rank.

In Section 4.6, we extend the result from Section 4.5 to the stable roommates setting. The different structure of the acceptability graph in this setting means that we cannot employ exactly the same approach as in the stable marriage setting. However, with some careful modifications to this method, we reduce our problem to that of counting perfect matchings in a general graph. Our result then follows from the existence of an XP algorithm for computing the "Hafnian" of a square matrix parameterised by the rank of the matrix.

The focus of Section 4.7 is on the complexity of finding and counting strongly stable and super-stable matchings in the typed setting. Under super-stability, we show that the set of super-stable matchings for each of TYPED SMTI, TYPED SRTI and TYPED HRT have strong structural properties when we restrict the number of allowable agent types needed to describe an instance. In each setting, we will see that these structural properties allow us to find and count super-stable matchings efficiently. Under strong stability, we show that the algorithm described in Section 4.6 can be extended to count the number of strongly stable matchings admitted by an instance of TYPED SRTI.

4.5 #TYPED SMTI is in XP

In this section, we describe an algorithm for counting the number of weakly stable matchings admitted by an instance of TYPED SMTI. This problem is defined as follows.

#TYPED SMTI	
Input:	An instance I of TYPED SMTI containing at most n agents with at most k agent types.
Parameter:	k.
Question:	How many stable matchings does I admit?

Recall that the problem of counting stable matchings admitted by an instance of stable marriage is #P-complete [12] even when preference lists are complete and do not allow ties. In what follows, we show that the problem of counting stable matchings admitted by an instance of TYPED SMTI belongs to XP parameterised by the number of agent types needed to describe the instance. We begin by giving a high-level overview of our proof.

4.5.1 Proof Overview

The main idea of our proof is to reduce our problem to that of counting perfect matchings in a balanced bipartite graph parameterised by the neighbourhood diversity of the graph. We first show that the number of matchings admitted by an instance of TYPED SMTI is equal to the number of matchings meeting certain restrictions on the pairs of types present in the matching. We will see that the number of matchings meeting these restrictions can be obtained via an inclusion-exclusion argument from the number of perfect matchings in (a bounded number of) balanced bipartite graphs constructed from the problem instance. We show that the neighbourhood diversity of each such graph is bounded by a function of the number of agent types. In addition, the rank of the adjacency matrix corresponding to each graph has value at most the neighbourhood diversity of the graph. There exists a well-known relationship between the number of perfect matchings in a balanced bipartite graph and the value of the permanent of its biadjacency matrix. Our result then follows from an XP algorithm due to Barvinok [85] for computing the permanent of a matrix parameterised by its rank.

Our proof is divided into the following sections. Section 4.5.2 provides some preliminary definitions and basic lemmas needed to describe our result. Obtaining the reduced row echelon form of the input matrix is the first step of Barvinok's algorithm. Section 4.5.3 describes a variation of the Gaussian Elimination method for transforming a matrix into reduced row echelon form. We provide a specific upper bound on the runtime of this method in order to

obtain a precise bound on the runtime of Barvinok's algorithm in Section 4.5.4. Finally, in Section 4.5.5 we describe how to reduce our problem to that of counting perfect matchings in a bounded number of balanced bipartite graphs each with bounded neighbourhood diversity.

4.5.2 Preliminaries

This section contains some definitions and basic lemmas needed to describe our algorithm.

Definition 4.1. Let $G = ((U, V), E(G))$ be a bipartite graph with parts $U = \{u_1, \ldots, u_m\}$ and $V = \{v_1, \ldots, v_n\}$. The *biadjacency matrix* associated with G is an $m \times n$ matrix $B = (b_{i,j})$ such that $b_{i,j} = 1$ if $u_i v_j \in E(G)$, and $b_{i,j} = 0$ otherwise.

Lemma 4.2. *Let $G = ((U, V), E(G))$ be a bipartite graph with biadjacency matrix $B = (b_{i,j})$. The rank of B is at most the neighbourhood diversity of G.*

Proof. Let w denote the neighbourhood diversity of G. Since G is bipartite, vertices of the same type in fact have exactly the same neighbourhood in G. Thus, the rows in B corresponding to vertices of the same type are identical. It follows that there can be a maximum of w linearly independent rows in B. □

Note that it is possible that rows in B corresponding to vertices of different types in $V(G)$ may be linearly dependent. In this case, the rank of B is less than the neighbourhood diversity of G. We use S_n to denote the *symmetric group* containing all permutations of the numbers in the $[n]$. Let $s \in S_n$ be a permutation of $1, \ldots, n$. We use $s(i)$ to denote the number at position i in s for each $i \in [n]$.

Definition 4.3 ([86])**.** Let $A = (a_{i,j})$ be a real $n \times n$ matrix. The *permanent* of A is defined as

$$\text{per}(A) = \sum_{s \in S_n} \prod_{i=1}^{n} a_{i,s(i)}.$$

4.5.3 Gaussian Elimination with Complete Pivoting

Barvinok's algorithm takes as input an $n \times n$ matrix $A = (a_{i,j})$ and an upper bound r on the rank of A, and returns the value of the permanent of A. As an intermediary step of the algorithm, we must obtain the reduced row echelon form of the matrix. In this section, we describe a variant of the Gaussian elimination method, known as Gaussian elimination with complete pivoting, used for transforming a matrix into reduced row echelon form. Note that this variant of the Gaussian elimination method is chosen as it is known to prevent large growth of intermediary values in the reduction process [87]. We provide a detailed analysis

of the runtime of this method, which will be used later to give an explicit upper bound on the runtime of Barvinok's algorithm.

Let $A = (a_{i,j})$ be a matrix. We say that row i of A is a *zero-row* if all entries in a_i are zeros. If a_i contains at least one non-zero entry, then we call the leftmost non-zero entry in a_i the *leading coefficient* of row i. If all entries in A are zeros then we call A the *zero matrix*. We say that A is in *row echelon form* if all pairs of rows a_i and a_j in A with $i < j$ are such that

- if a_i is a zero-row, then a_j is also a zero-row, and
- if neither a_i nor a_j is a zero-row, then the column c_i of the leading coefficient of a_i and the column c_j of the leading coefficient of a_j are such that $c_j > c_i$.

A *square matrix* (containing the same number of rows as columns) satisfying these conditions is said to be an *upper triangular matrix*, as all non-zero entries lie on or above the leading diagonal. We say that a matrix A is in *reduced row echelon form* if A is in row echelon form and also

- the leading coefficient of any row which is not a zero-row is equal to one, and
- the leading coefficient of any row which is not a zero-row is the only non-zero entry in its column.

The following operations on the rows/columns of a matrix are known collectively as *elementary row operations*:

- multiplying a row/column by a non-zero constant;
- adding a constant multiple of one row/column to another;
- swapping two rows/columns;

Two matrices are said to be *equivalent* if one can be transformed into the other via a sequence of elementary row operations.

Gaussian elimination is a method for transforming any matrix into its equivalent reduced row echelon form using a sequence of elementary row operations. The first stage of Gaussian elimination - called *forward elimination* - reduces the matrix into row echelon form. The second stage - known as *backward substitution* - transforms the reduced matrix into reduced row echelon form. Note that every matrix has a unique reduced row echelon form.

Here we shall use a variation of Gaussian elimination known as *Gaussian elimination with complete pivoting* [87]. Let $A = (a_{i,j})$ be a matrix. The forward elimination of Gaussian elimination with complete pivoting involves repeatedly selecting an element p, called the *pivot*, from the lower right submatrix of A whose entries are not yet in reduced row echelon form, and rearranging rows and columns of A so that p is the top left element of the submatrix. Multiples of the row containing p are then subtracted from every row beneath so that each of these rows have a zero in the column containing p. This process is repeated until A is

in row echelon form. The first step of the backward substitution stage involves dividing each row in A containing at least one non-zero entry by the value of its leading coefficient. This ensures that all leading coefficients are equal to 1. Finally, for each row of A with a leading coefficient p, we subtract a multiple of this row from each row above to ensure that p is the only non-zero entry in its column.

In what follows, we describe the Gaussian elimination method with complete pivoting. In our setting, we apply Barvinok's algorithm only to binary square matrices. Hence, we shall assume that the input to our Gaussian elimination algorithm is a binary square matrix $A = (a_{i,j})$. We also assume without loss of generality that A is not the zero matrix. In what follows, we will see that the runtime of our algorithm depends upon the value of the rank of A. In particular, we will see that the number of "rounds" of both forward and backward substitution needed are each at most the rank of A. Note that the algorithm also returns the new positions of the columns from the original matrix in the matrix returned by the algorithm - this information will be required by the Barvinok algorithm.

Algorithm 1: Gaussian elimination with complete pivoting

input : a non-zero binary $n \times n$ matrix $A = (a_{i,j})$
output: the reduced row echelon form of A and the new positions $\mathbf{p} = [p_0, \ldots, p_{n-1}]$ of the columns in A in its reduced row echelon form

1 //Transform A into row echelon form
2 Let $\mathbf{p} = [p_0, \ldots, p_{n-1}]$ contain the current positions in A of the columns from the input matrix;
3 **for** $0 \leq k < n$ **do**
4 Find the element x with largest absolute value in the lower right $(n-k) \times (n-k)$ submatrix of A;
5 **if** $x \neq 0$ **then**
6 Swap rows and columns of A so that x is at position (k, k) and update $\mathbf{p}$;
7 **for** $k < i < n$ **do**
8 Subtract $\frac{a_{i,k}}{a_{k,k}}$ multiples of row k from row i so that $a_{i,k} = 0$;
9 **end**
10 **else**
11 //k is the rank of A;
12 Set $r = k$;
13 **break**;
14 **end**
15 If r has not been assigned a value then set $r = n$;
16 //Transform A into reduced row echelon form
17 **for** $0 \leq k < r$ **do**
18 **if** $a_{k,k} \neq 0$ **then**
19 Divide all entries in row k by $a_{k,k}$;
20 **end**
21 **end**
22 **for** $1 \leq k < r$ **do**
23 **foreach** $0 \leq i < k$ **do**
24 Subtract $a_{i,k}$ multiples of row k from row i so that $a_{i,k} = 0$;
25 **end**
26 **end**
27 **return** $A, \mathbf{p}$;

Observe that since Algorithm 1 performs only elementary row operations on A, the matrix returned by the algorithm is indeed equivalent to A. In what follows, we prove that Algorithm 1 transforms the input matrix A into reduced row echelon form in time depending polynomially on the number of rows in A and also on the rank of A.

Lemma 4.4. *Let $A = (a_{i,j})$ be the input to Algorithm 1. The value of r set at either Line 12 or Line 15 is equal to the rank of A.*

Proof. Suppose first that we set $r = k$ at Line 12 for some $0 \leq k < n$. It follows that all elements in the lower right $(n-k) \times (n-k)$ submatrix of A are zeros. In addition, the previous k iterations of the loop at Line 3 have ensured that

- the first k entries in rows k to $n-1$ are zeros, and
- for each $0 \leq i < k$, row i has a leading coefficient in column i.

It follows that the first k rows of A are linearly independent and that rows k to $n-1$ are zero rows. Thus, the rank of A is equal to k.

Now suppose that we set $r = n$ at Line 15. Since the loop at Line 3 did not break, we know that, for all $0 \leq i < n$, the leading coefficient of row i is in column i. It follows that every row in A is linearly independent. The result follows. □

The following lemma proves that the first stage (Lines 3 to 15) of the algorithm transforms A into row echelon form. The next lemma then proves that the second stage (Lines 17 to 27) of the algorithm then transforms the reduced matrix into reduced row echelon form.

Lemma 4.5. *Let $A = (a_{i,j})$ be the input to Algorithm 1, and let r denote the rank of A. Let ℓ_i denote the column of the leading coefficient in row i of A. Once Line 15 of Algorithm 1 is reached, the following hold:*

- *$\ell_i = i$ for all $0 \leq i < r$, and*
- *rows r to $n-1$ of A are zero-rows.*

Proof. Let ℓ_i denote the column of the leading coefficient in row i. It follows from Lemma 4.4 that exactly r iterations of the loop at Line 3 complete without breaking. In what follows, we prove via induction on k, for each $0 \leq k < r$, that after iteration k of the loop at Line 3, we have that

- $\ell_i = i$ for all $0 \leq i \leq k$, and
- $a_{i,j} = 0$ for all pairs $0 \leq i < n$ and $0 \leq j \leq k$ with $j < i$.

If $r = n$ then at this point we are done. Otherwise, we show that after iteration r of the loop at Line 3, we have that

- $\ell_i = i$ for all $0 \leq i < r$, and
- rows r to $n-1$ are zero-rows.

The result follows.

First consider the base case $k = 0$. Since A is not the zero matrix, we have that $x \neq 0$ at Line 5. After Line 6 of the first iteration, we have that $a_{0,0} \neq 0$ and hence $\ell_0 = 0$. In addition, the loop at Line 7 ensures that $a_{i,0} = 0$ for all $0 < i < n$ and does not change the values of the elements in the first row of A.

Now suppose that after iteration k, for some $0 \leq k < r-1$, we have that

(1) $\ell_i = i$ for $0 \leq i \leq k$, and

(2) $a_{i,j} = 0$ for all pairs $0 \leq i < n$ and $0 \leq j \leq k$ such that $j < i$.

We now consider iteration $(k+1)$. Since $k < r-1$, and we know that r iterations complete without breaking, we must have that $x \neq 0$ at Line 5 in this iteration. Observe that none of rows zero to k are altered in iteration $(k+1)$. Hence, it follows from (2) that $a_{k+1,j} = 0$ for each $0 \leq j \leq k$. Thus, after Line 6 of iteration $(k+1)$, we have that $\ell_{k+1} = (k+1)$. In addition, after subtracting row $(k+1)$ from all rows beneath, it follows from (1) that we have $a_{i,j} = 0$ for all pairs $0 \leq i < n$ and $0 \leq j \leq k+1$ such that $i > j$.

It remains to show that, if $r < n$, then after the final (rth) iteration of the loop at Line 3, we have that

- $\ell_i = i$ for all $0 \leq i < r$, and
- rows r to $n-1$ of A are zero-rows.

By Lemma 4.4, since $r < n$, we must have that $x = 0$ in the final iteration, so that the loop breaks immediately. The first condition then follows from the inductive argument above. We also have that $a_{i,j} = 0$ for all pairs $0 \leq i < n$ and $0 \leq j < r$ with $j < i$. Since $\ell_i = i$ for all $0 \leq i < r$, it follows that the first r rows of A are linearly independent. Suppose for a contradiction that a_q is not a zero-row for some $r \leq q < n$. Since $a_{i,j} = 0$ for all pairs $0 \leq i < n$ and $0 \leq j < r$ with $j < i$, we must have that $\ell_q \geq r$. It follows that row a_q is linearly independent from all of rows $a_0, \ldots, a_{r-1}$, a contradiction of the fact that A has rank r. □

It follows from Lemma 4.5 that once Line 15 of Algorithm 1 is reached, the input matrix A has been transformed into row echelon form. The following lemma proves that Algorithm 1 transforms A into its (unique) reduced row echelon form.

Lemma 4.6. *Let $A = (a_{i,j})$ be the input to Algorithm 1, and let r denote the rank of A. Once Line 27 of Algorithm 1 is reached, the following hold:*

(1) $a_{i,i} = 1$ for all $0 \leq i < r$, and

(2) $a_{i,j} = 0$ for all pairs $0 \leq i < r$ and $0 \leq j < r$ such that $i \neq j$, and

(3) rows r to $n-1$ of A are zero-rows.

Proof. Let ℓ_i denote the column of the leading coefficient in row i of A. We consider each of (1), (2) and (3) in turn. By Lemma 4.5, once Line 15 of Algorithm 1 is reached, the following hold:

(i) $\ell_i = i$ for each $0 \leq i < r$, and and

(ii) rows r to $n-1$ of A are zero-rows.

It follows from (i) that $a_{i,i} \neq 0$ for each $0 \leq i < r$ at Line 15. Hence, after the final iteration of the loop at Line 17, we have that $a_{i,i} = 1$ for each $0 \leq i < r$. That is, condition (1) holds for A at Line 21. Since the loop at Line 22 only subtracts multiples of row i from row j where $i > j$, it follows that condition (1) still holds for A at Line 27.

For each $0 \leq i < r$ and each $0 \leq j < i$, the loop at Line 23 subtracts sufficiently many multiples of row i from row j to ensure that $a_{j,i} = 0$. It follows from (i) and (ii) that row i is now the only row in A containing a non-zero entry in column i. Since row i is not subtracted from row j again in the algorithm, it follows that condition (2) holds for A at Line 27.

Finally, since rows r to $n-1$ of A are zero-rows at Line 15, and the loop at Line 22 does not alter any of rows r to $n-1$, it follows that condition (3) holds for A at Line 27. □

The following corollary follows directly from Lemma 4.6.

Corollary 4.7. *Given an $n \times n$ non-zero binary matrix $A = (a_{i,j})$ as input, Algorithm 1 transforms A into reduced row echelon form.*

Let $A = (a_{i,j})$ be the input to Algorithm 1. The following lemmas bound the number of bits needed to store the elements of A at each stage of the algorithm. We show that all non-zero intermediate values can be represented as a fraction whose numerator and denominator are sufficiently small. It follows that we can store each such value using a sufficiently small number of bits by storing the value of the numerator and the denominator separately. This will allow us to bound the time required to compute arithmetic operations on such values. Note that if we instead computed the division of the numerator and denominator then, if the denominator is not equal to a power of two, we may require a very large number of bits to store the value. Since the number of perfect matchings in a graph is always an integer, we do not have to be concerned with returning a non-integer value as the output of our overall algorithm.

Lemma 4.8. *Let $A = (a_{i,j})$ be the input to Algorithm 1, and let r denote the rank of A. Between Lines 3 and 15 of the algorithm, the elements of A are such that, for all $0 \leq i, j < n$, we have that either $a_{i,j} = 0$ or else $a_{i,j} = b_{i,j}/c_{i,j}$ for some $b_{i,j}, c_{i,j} \in \mathbb{Z}$ such that $0 < |b_{i,j}| \leq 2^{4^{2(r-1)}}$ and $0 < |c_{i,j}| \leq 2^{4^{2(r-1)}}$.*

Proof. Since the input matrix is binary, the elements of A satisfy the stated bounds prior to the first iteration of the loop at Line 3. Moreover, it follows from Lemma 4.4 that the values of the elements change in only the first r iterations of the loop at Line 3. In what follows, we prove by induction on k for $1 \leq k \leq r$ that, throughout iteration k of the loop at Line 3, for all pairs $0 \leq i, j < n$ such that $a_{i,j} \neq 0$, we have that $a_{i,j} = b_{i,j}/c_{i,j}$ for some $b_{i,j}, c_{i,j} \in \mathbb{Z}$ such that $0 < |b_{i,j}| \leq 2^{4^{2(k-1)}}$ and $0 < |c_{i,j}| \leq 2^{4^{2(k-1)}}$. The result then follows immediately.

First, consider the base case $k = 1$. Since the input matrix contains at least one non-zero value, we have that $x \neq 0$ at Line 5 in the first iteration. After the swapping of rows and columns, for each $0 < i < n$ and each $0 \leq j < n$, we set

$$a_{i,j} = a_{i,j} - \frac{a_{i,k}}{a_{k,k}} a_{k,j}.$$

Since the input matrix is binary, it follows that the new value of $a_{i,j}$ is such that $a_{i,j} \in \{-1, 0, 1\}$, so we are done.

Now suppose that, throughout iteration k for some $1 \leq k \leq r-1$, for all pairs $0 \leq i, j < n$ such that $a_{i,j} \neq 0$, we have that $a_{i,j} = b_{i,j}/c_{i,j}$ for some $b_{i,j}, c_{i,j} \in \mathbb{Z}$ such that $0 < |b_{i,j}| \leq 2^{4^{2(k-1)}}$ and $0 < |c_{i,j}| \leq 2^{4^{2(k-1)}}$. We now consider iteration $(k+1)$. After the swapping of rows and columns at Line 6, the element $a_{k+1,k+1}$ has the largest absolute value among all elements in the lower right $(n-k-1) \times (n-k-1)$ submatrix of A. Then, for each $k+1 < i < n$ and each $0 \leq j < n$, we set

$$a_{i,j} = a_{i,j} - \frac{a_{i,k+1}}{a_{k+1,k+1}} a_{k+1,j}. \tag{4.1}$$

Note that since $x \neq 0$ in this iteration, it follows that $a_{k+1,k+1} \neq 0$, and hence the fraction in (4.1) is defined. If any of the other values in (4.1) is equal to zero, then the upper bounds on the numerator on the denominator of the new value of $a_{i,j}$ are lower than they are otherwise, so we may assume without loss of generality that this is not the case. Then, by our inductive assumption, we may rewrite (4.1) in fractional form as

$$\begin{aligned} \frac{b_{i,j}}{c_{i,j}} &= \frac{b_{i,j}}{c_{i,j}} - \frac{b_{i,k+1}}{c_{i,k+1}} \frac{c_{k+1,k+1}}{b_{k+1,k+1}} \frac{b_{k+1,j}}{c_{k+1,j}} \\ &= \frac{b_{i,j} c_{i,k+1} b_{k+1,k+1} c_{k+1,j} - b_{i,k+1} c_{k+1,k+1} b_{k+1,j} c_{i,j}}{c_{i,j} c_{i,k+1} b_{k+1,k+1} c_{k+1,j}}. \end{aligned}$$

It follows from our inductive assumption that the new value of $a_{i,j}$ is such that either $a_{i,j} = 0$, or else $a_{i,j} = b_{i,j}/c_{i,j}$ for some $b_{i,j}, c_{i,j} \in \mathbb{Z}$ such that $0 < |b_{i,j}| \leq 2^{4 \times 4^{2(k-1)}+1} < 2^{4^{2k}}$ and $0 < |c_{i,j}| \leq 2^{4 \times 4^{2(k-1)}} < 2^{4^{2k}}$. □

Lemma 4.9. *Let $A = (a_{i,j})$ be the input to Algorithm 1, and let r denote the rank of A. Between Lines 17 and 27 of the algorithm, the elements of A are such that for all $0 \leq i, j < n$ we have that either $a_{i,j} = 0$ or else $a_{i,j} = b_{i,j}/c_{i,j}$ for some $b_{i,j}, c_{i,j} \in \mathbb{Z}$ such that $0 < |b_{i,j}| \leq 2^{4^{3(r-1)}}$ and $0 < |c_{i,j}| \leq 2^{4^{3(r-1)}}$.*

Proof. By Lemma 4.5, once Line 15 of Algorithm 1 is reached, rows r to $n-1$ of A are zero-rows. Observe that Lines 17 to 27 change only the values of the elements in the first r rows of A. Hence, we only need to show that the elements in the first r rows of A satisfy the stated bounds. By Lemma 4.8, once Line 15 of Algorithm 1 is reached, for all $0 \leq i < r$

and $0 \leq j < n$ such that $a_{i,j} \neq 0$ we have that $a_{i,j} = b_{i,j}/c_{i,j}$ for some $b_{i,j}, c_{i,j} \in \mathbb{Z}$ such that $0 < |b_{i,j}| \leq 2^{4^{2(r-1)}}$ and $0 < |c_{i,j}| \leq 2^{4^{2(r-1)}}$.

First consider the loop at Line 17. For each $0 \leq k < r$ and each $0 \leq j < n$, we set $a_{k,j} = a_{k,j}/a_{k,k}$. It follows from Lemma 4.8 that, after the loop at Line 17, either $a_{k,j} \neq 0$ or else $a_{k,j} = b_{k,j}/c_{k,j}$ for some $b_{k,j}, c_{k,j} \in \mathbb{Z}$ such that $0 < |b_{k,j}| \leq 2^{4^{2r-1}}$ and $0 < |c_{k,j}| \leq 2^{4^{2r-1}}$.

We now consider the loop at Line 22. Since the loop at Line 22 only subtracts multiples of row k from row j with $k > j$, it follows that between Lines 22 and 27 we continue to have that $a_{i,i} = 1$ for all $0 \leq i < r$. In what follows, we prove by induction on k, for $1 \leq k < r$, that throughout the kth iteration of the loop at Line 22, for all pairs $0 \leq i < r$ and $0 \leq j < n$ such that $a_{i,j} \neq 0$, we have that $a_{i,j} = b_{i,j}/c_{i,j}$ for some $b_{i,j}, c_{i,j} \in \mathbb{Z}$ such that $|b_{i,j}| \leq 2^{4^{(k-1)+2r-1}}$ and $0 < c \leq 2^{4^{(k-1)+2r-1}}$. Since $k < r$, the result follows.

The base case $k = 1$ follows from the bound provided above on the values of the elements after the loop at Line 17. Now suppose that throughout the kth iteration of the loop at Line 22, for some $1 \leq k < r - 1$, for all pairs $0 \leq i < r$ and $0 \leq j < n$ such that $a_{i,j} \neq 0$, we have that $a_{i,j} = b_{i,j}/c_{i,j}$ for some $b_{i,j}, c_{i,j} \in \mathbb{Z}$ such that $0 < |b_{i,j}| \leq 2^{4^{(k-1)+2r-1}}$ and $0 < |c_{i,j}| \leq 2^{4^{(k-1)+2r-1}}$. We now consider the $(k+1)$th iteration. For each $0 \leq i < k+1$ and each $0 \leq j < n$, at Line 24 we set

$$a_{i,j} = a_{i,j} - a_{i,k}a_{k+1,j}. \tag{4.2}$$

If any of the values on the right-hand side of (4.2) are nonzero, then the upper bounds on the numerator and denominator of the new value of $a_{i,j}$ are lower than they would be otherwise, so we may assume without loss of generality that this is not the case. Then, we may rewrite (4.2) in fractional form as

$$\frac{b_{i,j}}{c_{i,j}} = \frac{b_{i,j}c_{i,k+1}c_{k+1,j} - b_{i,k+1}b_{k+1,j}c_{i,j}}{c_{i,j}c_{i,k+1}c_{k+1,j}}.$$

It follows from our inductive assumptions that the new value of $a_{i,j}$ is such that either $a_{i,j} = 0$ or else $a_{i,j} = b_{i,j}/c_{i,j}$ for some $b_{i,j}, c_{i,j} \in \mathbb{Z}$ such that

$$0 < |b_{i,j}| \leq 2 \times (2^{4^{(k-1)+2r-1}})^3 < 2^{4^{k+2r-1}}$$

and

$$0 < |c_{i,j}| \leq (2^{4^{(k-1)+2r-1}})^3 < 2^{4^{k+2r-1}}.$$

The result follows. □

The following corollary follows from Lemmas 4.8 and 4.9.

Corollary 4.10. *Let $A = (a_{i,j})$ be the input to Algorithm 1, and let r denote the rank of A. The matrix $B = (b_{i,j})$ returned by the algorithm is such that, for all $0 \leq i, j < n$, we have that either $b_{i,j} = 0$ or else $b_{i,j} = b/c$ for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3(r-1)}}$ and $0 < |c| \leq 2^{4^{3(r-1)}}$.*

We are now ready to describe an upper bound on the runtime of Algorithm 1. The first lemma bounds the time required to transform the matrix into row echelon form. The second lemma then bounds the runtime of the overall algorithm.

Lemma 4.11. *Let $A = (a_{i,j})$ be an $n \times n$ binary matrix with rank r. Given A as input, Lines 3 to 15 of Algorithm 1 take time $\mathcal{O}(rn^2 4^{4r})$.*

Proof. It follows from Lemma 4.4 that there are r iterations of the loop at Line 3 which do not break. For each $0 \leq k \leq r$, at Line 4 we find the element in the lower right $(n-k) \times (n-k)$ submatrix of A with the largest absolute value. By Lemma 4.8, between Lines 3 to 15 of Algorithm 1, the elements of A are such that, for all pairs $0 \leq i, j < n$ such that $a_{i,j} \neq 0$, we have that $a_{i,j} = b_{i,j}/c_{i,j}$ for some $b_{i,j}, c_{i,j} \in \mathbb{Z}$ such that $0 < |b_{i,j}| \leq 2^{4^{2(r-1)}}$ and $0 < |c_{i,j}| \leq 2^{4^{2(r-1)}}$. We can compare the value of two such elements by multiplying the numerator of each by the denominator of the other and comparing the values of the products. This can be achieved in time

$$\begin{aligned} &\mathcal{O}(2 \times (4^{2(r-1)} \times 4^{2(r-1)})) \\ &= \mathcal{O}(4^{4(r-1)}) \end{aligned} \tag{4.3}$$

Since finding the largest element in an $n \times n$ matrix involves at most n^2 comparisons, it follows from (4.3) that Line 4 takes time

$$\mathcal{O}(n^2 4^{4(r-1)}) \tag{4.4}$$

for each $0 \leq k < r$.

For each $0 \leq k < r$, at Line 8, we set

$$a_{i,j} = a_{i,j} - \frac{a_{i,k}}{a_{k,k}} a_{k,j} \tag{4.5}$$

for each $k < i < n$ and each $0 \leq j < n$. Since $a_{k,k} \neq 0$, the fraction in (4.5) is valid. If any other term on the right-hand side is zero, then the equation simplifies and the runtime of computing the new value is reduced, so we may suppose without loss of generality that this is not the case. It then follows from Lemma 4.8 that we can rewrite (4.5) as

$$a_{i,j} = \frac{b_{i,j} c_{i,k+1} b_{k+1,k+1} c_{k+1,j} - b_{i,k+1} c_{k+1,k+1} b_{k+1,j} c_{i,j}}{c_{i,j} c_{i,k+1} b_{k+1,k+1} c_{k+1,j}} \tag{4.6}$$

where, for all $0 \leq i', j' < n$, we have that $b_{i',j'}, c_{i',j'} \in \mathbb{Z}$ where $0 < |b_{i',j}| \leq 2^{4^{2(r-1)}}$ and $0 < |c_{i',j'}| \leq 2^{4^{2(r-1)}}$. It follows that each of the products in (4.6) is an integer a such that

$$0 < |a| \leq (2^{4^{2(r-1)}})^4 = 2^{4^{2r-1}}.$$

It follows that computing all three products in (4.6) takes time

$$\begin{aligned} &\mathcal{O}(3 \times 3 \times (4^{2(r-1)} \times 4^{2r-1})) \\ &= \mathcal{O}(4^{4r}). \end{aligned}$$

In addition, the subtraction in (4.6) can be computed in time $\mathcal{O}(4^{2r})$. It follows that computing the new value of $a_{i,j}$ (and storing the numerator and denominator separately) takes time $\mathcal{O}(4^{4r})$. Hence, the loop at Line 7 takes time

$$\mathcal{O}(n^2 4^{4r}). \tag{4.7}$$

Finally, it follows from (4.4) and (4.7) that Lines 3 to 15 of Algorithm 1 take time

$$\begin{aligned} &\mathcal{O}(r \times (n^2 4^{2(r-1)} + n^2 4^{4r})) \\ &= \mathcal{O}(r n^2 4^{4r}). \end{aligned}$$

□

The following lemma bounds the runtime of Algorithm 1.

Lemma 4.12. *Let $A = (a_{i,j})$ be an $n \times n$ binary matrix with rank r. Given A as input, Algorithm 1 returns the reduced row echelon form of A in time $\mathcal{O}(r^2 n^2 4^{6r})$.*

Proof. By Corollary 4.7, given $A = (a_{i,j})$ as input, Algorithm 1 returns the reduced row echelon form of A. By Lemma 4.11, Lines 3 to 15 of Algorithm 1 take time $\mathcal{O}(r n^2 4^{4r})$. In what follows, we show that Lines 17 to 27 take time $\mathcal{O}(r^2 n 4^{6r})$. The result follows.

By Lemma 4.9, between Lines 17 and 27 of Algorithm 1, each element $a_{i,j}$ is such that either $a_{i,j} = 0$ or else $a_{i,j} = b_{i,j}/c_{i,j}$ for some $b_{i,j}, c_{i,j} \in \mathbb{Z}$ such that $0 < |b_{i,j}| \leq 2^{4^{3(r-1)}}$ and $0 < |c_{i,j}| \leq 2^{4^{3(r-1)}}$. It follows that between Lines 17 and 27 of Algorithm 1, the numerator and denominator of each element of A can be represented using $\mathcal{O}(4^{3(r-1)})$ bits each.

We first consider the runtime of the loop at Line 17. For each $0 \leq k < r$ and each $0 \leq j < n$, calculating the new value of $a_{k,j}$ requires multiplying the numerator of (the old value of) $a_{k,j}$ by the denominator of $a_{k,k}$, and multiplying the numerator of $a_{k,k}$ by the denominator of (the

old value of) $a_{k,j}$. It follows that calculating the new value of $a_{k,j}$ takes time

$$\begin{aligned}&\mathcal{O}(2 \times (4^{3(r-1)})^2)\\ &= \mathcal{O}(4^{6(r-1)})\end{aligned}$$

for each $0 \leq k < r$ and each $0 \leq j < n$. Hence, the loop at Line 17 takes time

$$\mathcal{O}(4^{6(r-1)}rn). \tag{4.8}$$

We now consider the runtime of the loop at Line 22. For each $1 \leq k < r$, each $0 \leq i < k$ and each $0 \leq j < n$, we set

$$a_{i,j} = a_{i,j} - a_{i,k}a_{k,j}. \tag{4.9}$$

Note that if either $a_{i,k} = 0$ or $a_{k,j} = 0$, then we do not change the value of $a_{i,j}$, so we may suppose without loss of generality that this is not the case. It follows from Lemma 4.9 that we can rewrite (4.9) in fractional form as

$$\frac{b_{i,j}}{c_{i,j}} = \frac{b_{i,j}c_{i,k}c_{k,j} - b_{i,k}b_{k,j}c_{i,j}}{c_{i,j}c_{i,k}c_{k,j}} \tag{4.10}$$

where, for each $0 \leq i', j' < n$, we have that $b_{i',j'}, c_{i',j'} \in \mathbb{Z}$ where $0 < |b_{i',j'}| \leq 2^{4^{3(r-1)}}$ and $0 < |c_{i',j'}| \leq 2^{4^{3(r-1)}}$. It follows that the value of each product in (4.10) is an integer a such that $0 < |a| \leq (2^{4^{3(r-1)}})^3 < 2^{4^{3r}}$. Hence, calculating all three products in (4.10) takes time

$$\begin{aligned}&\mathcal{O}(3 \times 3 \times (4^{3(r-1)} \times 4^{3r}))\\ &= \mathcal{O}(4^{6r}).\end{aligned}$$

Computing the subtraction in (4.10) takes time $\mathcal{O}(4^{3r})$. Hence, computing the new value of $a_{i,j}$ takes time $\mathcal{O}(4^{6r})$. It follows that the loop at Line 22 takes time

$$\mathcal{O}(r^2n4^{6r}). \tag{4.11}$$

It follows from (4.8) and (4.11) that Lines 17 to 27 of Algorithm 1 take time

$$\begin{aligned}&\mathcal{O}(rn4^{6(r-1)} + r^2n4^{6r})\\ &= \mathcal{O}(r^2n4^{6r}).\end{aligned}$$

□

4.5.4 Computing the Permanent

In this section, we describe Barvinok's XP algorithm [85] for computing the permanent of a square matrix parameterised by the rank of the matrix. The original runtime bound for the algorithm provided in [85] contains an unknown constant in the exponent of n. Here, we bound the runtime precisely.

Let $A = (a_{i,j})$ be an $n \times n$ matrix with rank r. A *matrix factorisation* is a factorisation of a matrix into a product of matrices. Since A has rank r, there exists a matrix factorisation of A into a product CB, where $C = (c_{i,j})$ is an $n \times r$ matrix, and $B = (b_{i,j})$ is an $r \times n$ matrix. We may then define two polynomials L and R, whose coefficients are entries from B and C respectively. Barvinok's algorithm computes the value of per(A) from the values of the coefficients in the expansions of each of L and R. We note that in [85], the input matrix is factorised into two $n \times n$ matrices P and Q. However, the algorithm uses terms from only the top left $r \times n$ submatrix of P, and the top left $n \times r$ submatrix of Q. Hence, we may instead factorise A into an $r \times n$ and an $n \times r$ matrix.

Let $A = (a_{i,j})$ be an $n \times n$ matrix with rank r. Since we will apply Barvinok's algorithm only to binary matrices, we shall assume that A is binary. We also assume without loss of generality that A is not the zero matrix. We note that no specific factorisation of A is provided in [85]. Here, we use the following factorisation described by Piziak and Odell in [88]. Let $D = (d_{i,j})$ be the reduced row echelon form of A, and let $\mathbf{p} = [p_0, \ldots, p_{n-1}]$ denote the order in which the columns of A appear in D. Let $B = (b_{i,j})$ be the $r \times n$ matrix obtained from D by removing all zero-rows. Let $C = (c_{i,j})$ be the $n \times r$ matrix containing those columns of A which correspond (via $\mathbf{p}$) to columns containing leading coefficients in D. Note that the columns in C should appear in the same relative order in which they appear in A. We have that $CB = A$.

Let α_r^n denote the set containing all sets $\alpha = \{\alpha_1, \ldots, \alpha_r\}$ of r non-negative integers whose sum is equal to n. The following algorithm describes Barvinok's method for computing the permanent of an $n \times n$ binary matrix A with rank r using the factorisation of A provided by Piziak and Odell. Note that we obtain the reduced row echelon form of A using Algorithm 1 from Section 4.5.3.

Algorithm 2: Computing the Permanent

input : a binary square matrix $A = (a_{i,j})$ with rank r

output: the value p of the permanent of A

1 Let $(D = (d_{i,j}), \mathbf{p})$ be the output of Algorithm 1 with input A;
2 Let $B = (b_{i,j})$ be the matrix obtained from D by removing all zero-rows;
3 Let $C = (c_{i,j})$ be the matrix obtained from A by removing all columns that do not contain a leading one in D;
4 Set $L(x_1, \ldots, x_r) = \prod_{j=0}^{n-1} \sum_{i=0}^{r-1} b_{i,j} x_{i+1}$;
5 Set $R(x_1, \ldots, x_r) = \prod_{i=0}^{n-1} \sum_{j=0}^{r-1} c_{i,j} x_{j+1}$;
6 Set $per(A) = 0$;
7 **foreach** $\alpha \in \alpha_r^n$ **do**
8 Let λ_α be the coefficient of $x_1^{\alpha_1} \ldots x_r^{\alpha_r}$ in L;
9 Let ρ_α be the coefficient of $x_1^{\alpha_1} \ldots x_r^{\alpha_r}$ in R;
10 Increment $per(A)$ by $\alpha_1! \cdots \alpha_r! \cdot \lambda_\alpha \cdot \rho_\alpha$;
11 **end**
12 **return** $per(A)$;

In the remainder of this section, we describe bounds on the runtime and the size of the output produced by Algorithm 2. The following lemma describes an upper bound on the time needed to compute the expansion of L in Algorithm 2, as well as upper and lower bounds on the values of the coefficients in the expansion.

Lemma 4.13. *Let $A = (a_{i,j})$ be a binary $n \times n$ matrix, and let r denote the rank of A. Given A as input, expanding L at Line 4 of Algorithm 2 takes time*

$$\mathcal{O}((n + r - 1)^r n^4 r^3 4^{6(r-1)})$$

and produces a polynomial

$$L = \sum_{\alpha = \{\alpha_1, \ldots, \alpha_r\} \in \alpha_r^n} b_\alpha x_1^{\alpha_1} \ldots x_r^{\alpha_r}$$

such that, for all $\alpha \in \alpha_r^n$, we have that either $b_\alpha = 0$, or else $b_\alpha = b/c$ for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3(r-1)} rn(n-1) r^{(n-1)}}$ and $0 < |c| \leq 2^{4^{3(r-1)} rn(n-1)}$.

Proof. By Corollary 4.10, the elements of the matrix $B = (b_{i,j})$ in Algorithm 2 are such that,

for all $0 \leq i, j < n$, we have that either $b_{i,j} = 0$ or $b_{i,j} = b/c$ for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3(r-1)}}$ and $0 < |c| \leq 2^{4^{3(r-1)}}$. We can write the expansion of L in Algorithm 2 as

$$L = (b_{0,0}x_1 + b_{1,0}x_2 + \ldots + b_{r-1,0}x_r) \times \cdots \times (b_{0,n-1}x_1 + b_{1,n-1}x_2 + \ldots b_{r-1,n-1}x_r). \tag{4.12}$$

In what follows, we prove by induction on k, for $1 \leq k \leq n$, that multiplying out the first k brackets of (4.12) takes time

$$\mathcal{O}\left(\sum_{1 \leq i \leq k} \left((i+r-1)^r i^3 r^3 4^{6(r-1)}\right)\right)$$

and produces a polynomial

$$\sum_{\alpha = \{\alpha_1, \ldots, \alpha_r\} \in \alpha_r^k} b_\alpha x_1^{\alpha_1} \ldots x_r^{\alpha_r}$$

such that, for all $\alpha \in \alpha_r^k$, we have that either $b_\alpha = 0$, or else $b_\alpha = b/c$ for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3(r-1)} rk(k-1)} r^{k-1}$ and $0 < |c| \leq 2^{4^{3(r-1)} rk(k-1)}$. The result follows.

First consider the base case $k = 1$. In this case, our expansion simply includes the first bracket of (4.12). It follows from Corollary 4.10 that the values of the coefficients in the first bracket of (4.12) are such that, for all $1 \leq i \leq r$, we have that either $b_{i,0} = 0$, or else $b_{i,0} = b/c$ for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3(r-1)}}$ and $0 < |c| \leq 2^{4^{3(r-1)}}$.

Now suppose that multiplying out the first $(k-1)$ brackets of (4.12) takes time

$$\mathcal{O}\left(\sum_{1 \leq i \leq k-1} \left((i+r-1)^r r^3 i^3 4^{6(r-1)}\right)\right) \tag{4.13}$$

and produces a polynomial

$$\sum_{\alpha = \{\alpha_1, \ldots, \alpha_r\} \in \alpha_r^{k-1}} b_\alpha x_1^{\alpha_1} \ldots x_r^{\alpha_r} \tag{4.14}$$

such that for each $\alpha \in \alpha_r^{k-1}$ we have that either $b_\alpha = 0$, or else $b_\alpha = b/c$ for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3(r-1)} r(k-1)(k-2)} r^{(k-2)}$ and $0 < |c| \leq 2^{4^{3(r-1)} r(k-1)(k-2)}$. We now consider the expansion of the first k brackets of (4.12). Expanding the first k brackets of (4.12) is equivalent to multiplying (4.14) by

$$(b_{0,k-1}x_1 + \ldots + b_{r-1,k-1}x_r). \tag{4.15}$$

The first step involves multiplying together pairs of terms from each of (4.14) and (4.15).

By the stars and bars argument, the number of ways to select r non-negative integers whose sum is equal to $(k-1)$ is at most $\binom{k+r-2}{r-1}$. It follows that the number of elements in the set α_r^{k-1} - and hence the number of distinct terms in the expansion of (4.14) - is at most $(k+r-2)^r$. There are at most r terms in (4.15). It follows from Corollary 4.10 that the numerator and the denominator of each coefficient in (4.15) can each be represented using $\mathcal{O}(4^{3(r-1)})$ bits. By our inductive assumption, the numerator and the denominator of each coefficient in (4.14) can be represented using $\mathcal{O}(4^{3(r-1)}r(k-1)(k-2)+(k-2)\log r)$ bits and $\mathcal{O}(4^{3(r-1)}r(k-1)(k-2))$ bits respectively. It follows that multiplying together a single pair of terms from each of (4.14) and (4.15) takes time

$$\begin{aligned}&\mathcal{O}((4^{3(r-1)}r(k-1)(k-2)+(k-2)\log r)\times 4^{3(r-1)}+4^{3(r-1)}r(k-1)(k-2)\times 4^{3(r-1)})\\&=\mathcal{O}(4^{6(r-1)}r(k-1)(k-2)).\end{aligned}$$

Hence, multiplying all pairs of terms from (4.14) and (4.15) takes time

$$\begin{aligned}&\mathcal{O}(r\times(k+r-2)^r\times(4^{6(r-1)}r(k-1)(k-2)))\\&=\mathcal{O}((k+r-2)^r r^2(k-1)(k-2)4^{6(r-1)}).\end{aligned}\tag{4.16}$$

In addition, each coefficient a of the multiplied terms is such that either $a=0$ or else $a=b/c$ for some $b,c\in\mathbb{Z}$ such that

$$0<|b|\leq 2^{4^{3(r-1)}(r(k-1)(k-2)+1)}r^{k-2}\tag{4.17}$$

and

$$0<|c|\leq 2^{4^{3(r-1)}(r(k-1)(k-2)+1)}.\tag{4.18}$$

The second step of the expansion involves collecting like terms. The number of ways to select r non-negative integers whose sum is equal to k - and hence the number of distinct terms in the expansion of the first k brackets of (4.12) - is at most $(k+r-1)^r$. For each of these terms, there are up to r like terms in the multiplication of (4.14) and (4.15). To sum the coefficients of each of the like terms, we must first determine a common denominator between them. The expansion of the first k brackets of (4.12) involves using at most rk distinct elements from the matrix B. It follows that to produce a common denominator among the coefficients of the (at most) r like terms, we need to multiply (the numerator and denominator of) the coefficient of each of the (at most r) like terms by the denominator of at most rk distinct elements from B. It follows from Corollary 4.10, (4.17) and (4.18) that, once we have established a common denominator among the coefficients of the like terms,

each new coefficient a is such that either $a = 0$ or else $a = b/c$ for some $b, c \in \mathbb{Z}$ where

$$\begin{aligned} 0 < |b| &\leq 2^{4^{3(r-1)}(r(k-1)(k-2)+1)} r^{(k-2)} \times (2^{4^{3(r-1)}})^{rk} \\ &\leq 2^{4^{3(r-1)} rk(k-1)} r^{(k-2)} \end{aligned} \tag{4.19}$$

and

$$\begin{aligned} 0 < |c| &\leq 2^{4^{3(r-1)}(r(k-1)(k-2)+1)} \times (2^{4^{3(r-1)}})^{rk} \\ &\leq 2^{4^{3(r-1)} rk(k-1)}. \end{aligned} \tag{4.20}$$

Hence, for a particular (distinct) term in the multiplication of (4.14) and (4.15), creating a common denominator among the coefficients of the like terms takes time

$$\begin{aligned} &\mathcal{O}(r \times rk \times 4^{3(r-1)} \times ((4^{3(r-1)} rk(k-1) + (k-2)\log r) + 4^{3(r-1)} rk(k-1))) \\ &= \mathcal{O}(r^3 k^2 (k-1) 4^{6(r-1)}). \end{aligned} \tag{4.21}$$

Finally, we must sum each of the (numerators of) the at most r like terms. It follows from (4.19) that this takes time

$$\begin{aligned} &\mathcal{O}(r \times (4^{3(r-1)} rk(k-1) + (k-2)\log r)) \\ &= \mathcal{O}(r^2 k(k-1) 4^{3(r-1)}) \end{aligned} \tag{4.22}$$

and, in each case, produces a new numerator b' such that $b' = 0$ or else $b' \in \mathbb{Z}$ and

$$\begin{aligned} 0 < |b'| &\leq r \times 2^{4^{3(r-1)} rk(k-1)} r^{(k-2)} \\ &= 2^{4^{3(r-1)} rk(k-1)} r^{(k-1)}. \end{aligned} \tag{4.23}$$

Since there are at most $(r + k - 1)^r$ distinct terms in the expansion of the first k brackets of (4.12), it follows from (4.21) and (4.22) that collecting like all terms requires time

$$\begin{aligned} &\mathcal{O}((r+k-1)^r \times (r^3 k^2 (k-1) 4^{6(r-1)}) + (r^2 k(k-1) 4^{3(r-1)})) \\ &= \mathcal{O}((r+k-1)^r 4^{6(r-1)} r^3 k^2 (k-1)). \end{aligned} \tag{4.24}$$

It follows from (4.16) and (4.24) that multiplying (4.14) and (4.15) takes time

$$\begin{aligned} &\mathcal{O}((k+r-2)^r r^2 (k-1)(k-2) 4^{6(r-1)} + (r+k-1)^r 4^{6(r-1)} r^3 k^2 (k-1)) \\ &= \mathcal{O}((r+k-1)^r 4^{6(r-1)} r^3 k^2 (k-1)). \end{aligned} \tag{4.25}$$

Finally, it follows from (4.13) and (4.25) that multiplying out the first k brackets of (4.12)

takes time

$$\mathcal{O}\left(\sum_{1\leq i\leq k}\left((i+r-1)^r r^3 i^3 4^{6(r-1)}\right)\right).$$

In addition, it follows from (4.20) and (4.23) that that the value of the coefficient of any term in the expansion of the first k brackets of (4.12) is either equal to zero or can be written in the form b/c for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3(r-1)}rk(k-1)}r^{(k-1)}$ and $0 < |c| \leq 2^{4^{3(r-1)}rk(k-1)}$. □

We now consider the time needed to expand the polynomial R in Algorithm 2. Since we have assumed that the input matrix A is binary, it follows that C is also binary. Hence, we can use Lemma 4.13 to obtain a rudimentary upper bound on the time needed to expand R and also on the values of the coefficients in the expanded polynomial.

Lemma 4.14. *Let $A = (a_{i,j})$ be a binary $n \times n$ matrix, and let r denote the rank of A. Given A as input, expanding R at Line 5 of Algorithm 2 takes time*

$$\mathcal{O}((n+r-1)^r n^4 r^3 4^{6(r-1)})$$

and produces a polynomial

$$R = \sum_{\alpha=\{\alpha_1,\ldots,\alpha_r\}\in\alpha_r^n} c_\alpha x_1^{\alpha_1}\ldots x_r^{\alpha_r}$$

such that, for all $\alpha \in \alpha_r^n$, we have that either $c_\alpha = 0$, or else $c_\alpha = a/b$ for some $a, b \in \mathbb{Z}$ such that $0 < |a| \leq 2^{4^{3(r-1)}rn(n-1)}r^{2(n-1)}$ and $0 < |b| \leq 2^{4^{3(r-1)}rn(n-1)}$.

In the following lemma, we bound the time needed to compute the value added to p at Line 10 of Algorithm 2 at each iteration.

Lemma 4.15. *Let $A = (a_{i,j})$ be a binary $n \times n$ matrix with rank r. For each $\alpha \in \alpha_r^n$, the value of $\alpha_1!\ldots\alpha_r! \cdot \lambda_\alpha \cdot \rho_\alpha$ can be computed in time $\mathcal{O}(4^{6r}r^2n^2(n-1)^2)$. In addition, the value is either equal to zero or else can be written in the form b/c for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3r}rn(n-1)}r^{2(n-1)}n^n$ and $0 < |c| \leq 2^{4^{3r}rn(n-1)}$.*

Proof. Observe that if, for any $\alpha \in \alpha_r^n$, we have that $\lambda_\alpha = 0$ or $\rho_\alpha = 0$, then we do nothing in that iteration. Thus, we may suppose without loss of generality that this is not the case. It follows from Lemmas 4.13 and 4.14 that, for each $\alpha \in \alpha_r^n$, we have that

- $\lambda_\alpha = b/c$ for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3(r-1)}rn(n-1)}r^{(n-1)}$ and $0 < |c| \leq 2^{4^{3(r-1)}rn(n-1)}$, and

- $\rho_\alpha = b/c$ for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3(r-1)}rn(n-1)}r^{(n-1)}$ and $0 < |c| \leq 2^{4^{3(r-1)}rn(n-1)}$.

It follows that, for each α, computing the value of the product $\lambda_\alpha \cdot \rho_\alpha$ takes time

$$\begin{aligned}&\mathcal{O}(2 \times ((4^{3(r-1)}rn(n-1) + 2(n-1)\log r)^2)\\ &= \mathcal{O}(4^{6(r-1)}r^2n^2(n-1)^2)\end{aligned} \tag{4.26}$$

and produces a value a such that either $a = 0$ or else $a = b/c$ for some $b, c \in \mathbb{Z}$ such that

$$\begin{aligned}0 < |b| &\leq 2^{2\times 4^{3(r-1)}rn(n-1)}r^{2(n-1)}\\ &< 2^{4^{3r}rn(n-1)}r^{2(n-1)}\end{aligned} \tag{4.27}$$

and

$$\begin{aligned}0 < |c| &\leq 2^{2\times 4^{3(r-1)}rn(n-1)}\\ &< 2^{4^{3r}rn(n-1)}.\end{aligned} \tag{4.28}$$

For each α, we have that $0 \leq \alpha_i \leq n$ for all $1 \leq i \leq r$. It follows from [89] that we can compute the value of $\alpha_i!$ in time $\mathcal{O}(n^2)$ for each $1 \leq i \leq r$. In addition, for each α, we have that

$$\alpha_1 + \ldots + \alpha_r \leq n$$

and hence

$$\alpha_1! \ldots \alpha_r! \leq n^n. \tag{4.29}$$

It follows that, for each $\alpha \in \alpha_r^n$, we can compute the value of $\alpha_1! \ldots \alpha_r!$ in time

$$\mathcal{O}(r \times (n^2 + (n\log n)^2)) = \mathcal{O}(rn^2\log^2 n). \tag{4.30}$$

It follows from (4.27) and (4.29) that computing the product of $\alpha_1! \ldots \alpha_r!$ and $\lambda_\alpha \cdot \rho_\alpha$ takes time

$$\begin{aligned}&\mathcal{O}((4^{3r}rn(n-1) + 2(n-1)\log r) \times n\log n)\\ &= \mathcal{O}(4^{3r}rn^2(n-1)\log n).\end{aligned} \tag{4.31}$$

It follows (4.26), (4.30) and (4.31) that, for each $\alpha \in \alpha_r^n$, the value of $\alpha_1! \ldots \alpha_r! \cdot \lambda_\alpha \cdot \rho_\alpha$ can

be computed in time

$$\begin{aligned}&\mathcal{O}(4^{6(r-1)}r^2n^2(n-1)^2 + rn^2\log^2 n + 4^{3r}rn^2(n-1)\log n)\\&= \mathcal{O}(4^{6r}r^2n^2(n-1)^2).\end{aligned}$$

Finally, it follows from (4.27), (4.28) and (4.30) that, for each $\alpha \in \alpha_r^n$, either $\alpha_1!\ldots\alpha_r!\cdot\lambda_\alpha\cdot\rho_\alpha = 0$, or else its value can be expressed as a fraction b/c for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3r}rn(n-1)}r^{2(n-1)}n^n$ and $0 < |c| \leq 2^{4^{3r}rn(n-1)}$. □

In the next lemma, we use the bounds from Lemma 4.15 to bound the runtime of the loop at Line 7 of Algorithm 2.

Lemma 4.16. *Let $A = (a_{i,j})$ be a binary $n \times n$ matrix with rank at most r. Given A as input, the loop at Line 7 of Algorithm 2 takes time $\mathcal{O}(4^{6r}r^2n^2(n-1)^2(n+r-1)^{3r})$.*

Proof. There are at most $\binom{n+r-1}{r-1}$ sets in the set α_r^n. It follows that there are at most $(n + r - 1)^r$ iterations of the loop at Line 7. Let K denote the number of iterations of the loop at Line 7, where $K \leq (n + r - 1)^r$. In order to bound the runtime of the loop at Line 7, we must first bound the value of the summation. By Lemma 4.15, for each $\alpha \in \alpha_r^n$, the value of $\alpha_1!\ldots\alpha_r!\cdot\lambda_\alpha\cdot\rho_\alpha$ is either is equal to zero or else can be written as a fraction b/c for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3r}rn(n-1)}r^{2(n-1)}n^n$ and $0 < |c| \leq 2^{4^{3r}rn(n-1)}$. In what follows, we prove by induction on k, for $1 \leq k \leq K$ that, after the kth iteration of the loop at Line 7, the value of $per(A)$ is such that either $per(A) = 0$ or else $per(A) = b_k/c_k$ for some $b_k, c_k \in \mathbb{Z}$ such that $0 < |b_k| \leq 2^{4^{3r}rn(n-1)k(k-1)+k}r^{2(n-1)}n^n$ and $0 < |c_k| \leq 2^{4^{3r}rn(n-1)k}$. It then follows that the final value of $per(A)$ - if nonzero - can be written as a fraction b/c for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3r}rn(n-1)(n+r-1)^{2r}+(n+r-1)^r}r^{2(n-1)}n^n$ and $0 < |c_k| \leq 2^{4^{3r}rn(n-1)(n+r-1)^r}$.

The base case $k = 1$ follows immediately from the bounds provided in Lemma 4.15. Now suppose that after the kth iteration, for some $1 \leq k < K$, we have that either $per(A) = 0$ or else $per(A) = b_k/c_k$ for some $b_k, c_k \in \mathbb{Z}$ such that $0 < |b_k| \leq 2^{4^{3r}rn(n-1)k(k-1)+k}r^{2(n-1)}n^n$ and $0 < |c_k| \leq 2^{4^{3r}rn(n-1)k}$. It follows from Lemma 4.15 that, during the $(k+1)$th iteration, we increment the value of $per(A)$ by some value a such that either $a = 0$ or else $a = b/c$ for some $b, c \in \mathbb{Z}$ such that $0 < |b| \leq 2^{4^{3r}rn(n-1)}r^{2(n-1)}n^n$ and $0 < |c| \leq 2^{4^{3r}rn(n-1)}$. After the summation the new value of $per(A)$ is such that either $per(A) = 0$ or else $per(A) = b_{k+1}/c_{k+1}$ where $b_{k+1} = (b_kc + bc_k)$ and $c_{k+1} = cc_k$, and hence $b_{k+1}, c_{k+1} \in \mathbb{Z}$. It follows that

$$\begin{aligned}0 < |b_{k+1}| \leq 2 \times b_kc_k &\leq 2 \times 2^{4^{3r}rn(n-1)k(k-1)+k}r^{2(n-1)}n^n \times 2^{4^{3r}rn(n-1)k}\\&\leq 2^{4^{3r}rn(n-1)(k+1)k+(k+1)}r^{2(n-1)}n^n\end{aligned}$$

and

$$\begin{aligned} 0 < |c_{k+1}| &\leq 2^{4^{3r}rn(n-1)k} \times 2^{4^{3r}rn(n-1)} \\ &= 2^{4^{3r}rn(n-1)(k+1)}. \end{aligned}$$

Making use of the above bounds, we now evaluate the runtime. For each $\alpha \in \alpha_r^n$, it follows from Lemma 4.15 that the value of $\alpha_1! \dots \alpha_r! \cdot \lambda_\alpha \cdot \rho_\alpha$ can be computed in time

$$\mathcal{O}(4^{6r}r^2n^2(n-1)^2). \tag{4.32}$$

It remains to determine the time needed to add $\alpha_1! \dots \alpha_r! \cdot \lambda_\alpha \cdot \rho_\alpha$ to $per(A)$ at each iteration. We may assume without loss of generality that $\alpha_1! \dots \alpha_r! \cdot \lambda_\alpha \cdot \rho_\alpha$ is nonzero (otherwise, we do nothing in that iteration). It follows from Lemma 4.15 that the value of $\alpha_1! \dots \alpha_r! \cdot \lambda_\alpha \cdot \rho_\alpha$ at any iteration can be written as a fraction b_1/c_1 where $b_1, c_1 \in \mathbb{Z}$ and the value of each of b_1 and c_1 can be stored using $\mathcal{O}(4^{3r}rn(n-1))$ bits. It follows from the above arguments that the value of $per(A)$ (if nonzero) at any iteration can be written as a fraction b_2/c_2 where $b_2, c_2 \in \mathbb{Z}$ and the value of each of b_2 and c_2 can be stored using $\mathcal{O}(4^{3r}rn(n-1)(n+r-1)^{2r})$ bits. The new value of $per(A)$ can then be written as

$$per(A) = \frac{b_1c_2 + b_2c_1}{c_1c_2}.$$

Hence, for each $\alpha \in \alpha_r^n$, computing the new value of $per(A)$ at Line 10 (and storing the numerator and denominator of the result separately) takes time

$$\begin{aligned} &\mathcal{O}(3 \times 4^{3r}rn(n-1) \times 4^{3r}rn(n-1)(n+r-1)^{2r}) \\ &= \mathcal{O}(4^{6r}r^2n^2(n-1)^2(n+r-1)^{2r}). \end{aligned} \tag{4.33}$$

Finally, it follows from (4.32) and (4.33) that the loop at Line 7 takes time

$$\begin{aligned} &\mathcal{O}((n+r-1)^r \times (4^{6r}r^2n^2(n-1)^2 + 4^{6r}r^2n^2(n-1)^2(n+r-1)^{2r})) \\ &= \mathcal{O}(4^{6r}r^2n^2(n-1)^2(n+r-1)^{3r}). \end{aligned}$$

□

In the final lemma of this section, we establish an upper bound on the runtime of Barvinok's algorithm.

Lemma 4.17. *Let $A = (a_{i,j})$ be a binary non-zero $n \times n$ matrix with rank r. Given A as input, Algorithm 2 returns the permanent of A in time $\mathcal{O}(4^{6r}r^2n^2(n-1)^2(n+r-1)^{3r})$.*

Proof. By Lemma 4.12, Line 1 of Algorithm 2 takes time

$$\mathcal{O}(r^2n^24^{6r}). \tag{4.34}$$

It follows from Lemmas 4.13 and 4.14 that Lines 4 and 5 each take time

$$\mathcal{O}((n+r-1)^rn^4r^34^{6(r-1)}). \tag{4.35}$$

It follows from Lemma 4.16 that the loop at Line 7 takes time

$$\mathcal{O}(4^{6r}r^2n^2(n-1)^2(n+r-1)^{3r}). \tag{4.36}$$

Finally, it follows from (4.34), (4.35) and (4.36) that Algorithm 2 returns the value of the permanent of A in time

$$\begin{aligned}&\mathcal{O}(r^2n^24^{6r} + 2 \times (n+r-1)^rn^4r^34^{6(r-1)} + 4^{6r}r^2n^2(n-1)^2(n+r-1)^{3r})\\ &= \mathcal{O}(4^{6r}r^2n^2(n-1)^2(n+r-1)^{3r}).\end{aligned}$$

□

4.5.5 Reducing Typed Stable Matching to Counting Perfect Matchings

In this section, we prove that #TYPED SMTI is in XP parameterised by the number of agent types. To achieve this, we use observations made in the preceding sections to show that the problem of counting perfect matchings in a balanced bipartite graph belongs to XP parameterised by the neighbourhood diversity of the graph. We then show that the number of solutions to an instance of TYPED SMTI can be written in terms of the number of perfect matchings in each of a bounded number of balanced bipartite graphs whose neighbourhood diversity is bounded by the number of agent types.

Observe that the biadjacency matrix associated with a balanced bipartite graph is a square matrix. The following result describes a well-known relationship between the number of perfect matchings in a balanced bipartite graph and the permanent of its biadjacency matrix.

Theorem 4.18 ([90])**.** *Let $G = ((U, V), E)$ be a balanced bipartite graph with biadjacency matrix B. Let P denote the number of perfect matchings in G. We have that*

$$\text{per}(B) = P.$$

By Lemma 4.2, the rank of the biadjacency matrix associated with a bipartite graph is at

most the neighbourhood diversity of the graph. In Lemma 4.17, we described a precise upper bound on the runtime of Barvinok's algorithm for computing the permanent of a binary square matrix with bounded rank. The following corollary follows from Theorem 4.18, as well as Lemmas 4.2 and 4.17.

Corollary 4.19. *Let $G = (V(G), E(G))$ be a balanced bipartite graph with at most n vertices and neighbourhood diversity at most r. Let P denote the number of perfect matchings in G. We can determine the value of P in time*

$$\mathcal{O}(4^{6r}r^2n^2(n-1)^2(n+r-1)^{3r}).$$

The following lemma due to Meeks and Rastegari [11] shows that the stability of a matching admitted by an instance of TYPED SMTI depends only upon the pairs of types present in the matching. Let I be an instance of TYPED SMTI with at most k agent types, and let M be a matching admitted by I. Let $\text{worst}_M(i)$ denote the least desirable type of agent (from the perspective of type i) matched to any agent of type i in M. If a woman w with type i is unmatched in M, then for notational convenience we will say that w is matched to a "dummy man" of type $(k+1)$. Similarly, we say that an unmatched man is matched to a "dummy woman" of type $(k+2)$. Note that type $(k+1)$ (respectively type $(k+2)$) is considered less desirable to each type of woman (respectively man) than any other type.

Lemma 4.20 ([11])**.** *Let I be an instance of TYPED SMTI with at most k agent types. A matching M in I is stable if and only if there is no pair of types $(i,j) \in [k]^2$, with $i \neq j$, such that $j \succ_i \text{worst}_M(i)$ and $i \succ_j \text{worst}_M(j)$.*

We call a set T of pairs (i,j) of agent types in I a *type set*. The size of a type set T, denoted by $|T|$, is equal to the number of distinct pairs of types contained in T. We use $\text{worst}_T(i)$ to denote the least desirable type (from the perspective of type i) such that the pair $(\text{worst}_T(i), i)$ is contained in the type set T. We call a type set T a *stable type set* if it does not contain any pair of types (i,j) such that $j \succ_i \text{worst}_T(i)$ and $i \succ_j \text{worst}_T(j)$. We will use $\mathcal{T}$ to denote the set of stable type sets in an instance of TYPED SMTI. The following corollary follows from Lemma 4.20.

Corollary 4.21. *Let I be an instance of TYPED SMTI with at most k agent types. The stability of a type set in I can be determined in time $\mathcal{O}(k^2)$.*

The following lemma bounds the number of possible type sets in an instance of TYPED SMTI.

Lemma 4.22. *Let I be an instance of TYPED SMTI with at most k agent types. There are at most $2^{k(k+1)}$ type sets in I.*

Proof. Since there are at most $(k+1)$ types of women (including the type used to represent an unmatched man), there are at most 2^{k+1} ways to select a subset of types of women from I. Thus, for each type i of the men in I, there are at most 2^{k+1} ways to choose which types of women are matched to type i men. Since there are at most k types of men (not including the type used to represent an unmatched woman), there can be at most $2^{k(k+1)}$ type sets. □

The following corollary follows directly from Corollary 4.21 and Lemma 4.22.

Corollary 4.23. *Let I be an instance of TYPED SMTI with at most k agent types. The set $\mathcal{T}$ of stable type sets can be obtained in time $\mathcal{O}(2^{k(k+1)}k^2)$.*

We say that a matching M *satisfies* a type set T if, for each pair of agents $(x, y) \in M$, we have that (type(x),type(y)) $\in T$. Note that there may be some pairs $(t_1, t_2) \in T$ such that there are no pairs $(x, y) \in M$ with type$(x) = t_1$ and type$(y) = t_2$. We make the following observation.

Observation 4.24. Let I be an instance of TYPED SMTI, and let T be a stable type set in I. Let M be a matching which satisfies T. For any $i \in [k]$, we have that

$$\text{worst}_M(i) \succeq_i \text{worst}_T(i).$$

We use $\mathcal{M}^T$ to denote the set of all matchings which satisfy a type set T. The number of matchings that satisfy T is denoted by $|\mathcal{M}^T|$. We say that a matching M satisfies a type set T *precisely* if M satisfies T and, for each pair of types $(t_1, t_2) \in T$, there is at least one pair of agents $(x, y) \in M$ such that type$(x) = t_1$ and type$(y) = t_2$. We use $\mathcal{M}_P^T$ to denote the set of matchings precisely satisfying a type set T. The number of matchings precisely satisfying T is denoted by $|\mathcal{M}_P^T|$. Notice that $\mathcal{M}_P^T$ is a subset of $\mathcal{M}^T$. The following corollary follows from Observation 4.24.

Corollary 4.25. *All matchings precisely satisfying a stable type set are stable.*

The following observation follows directly from definitions.

Observation 4.26. Each stable matching precisely satisfies exactly one stable type set.

Corollary 4.25 and Observation 4.26 can be combined to give the following.

Corollary 4.27. *Let I be an instance of TYPED SMTI. The number of solutions to I is equal to $\sum_{T \in \mathcal{T}} |\mathcal{M}_P^T|$.*

We will see that the problem of calculating the value of $|\mathcal{M}^T|$ for each $T \in \mathcal{T}$ can be reduced to the problem of counting perfect matchings in a balanced bipartite graph whose

neighbourhood diversity is bounded by a function of the number of agent types. However, since a matching may satisfy more than one stable type set, the sum

$$\sum_{T \in \mathcal{T}} |\mathcal{M}^T|$$

over-counts the number of solutions. In the next lemmas, we describe how to efficiently obtain the number of stable matchings admitted by I given the value of $|\mathcal{M}^T|$ for each $T \in \mathcal{T}$.

We say that a type set T is *contained* in another type set T' if the pairs of types in T form a proper subset of the pairs of types in T'. We write this as $T \subset T'$. The following lemma describes the relationship between $\mathcal{M}_P^T$ and $\mathcal{M}^T$ for any $T \in \mathcal{T}$.

Lemma 4.28. *Let I be an instance of TYPED SMTI, and let T be a type set in $\mathcal{T}$. We have that*

$$\mathcal{M}_P^T = \mathcal{M}^T \setminus \bigcup_{T' \subset T} \mathcal{M}_P^{T'}.$$

Proof. We proceed by showing that any matching M in I belongs to the set $\mathcal{M}_P^T$ if and only if it belongs to the set $\mathcal{M}^T \setminus \bigcup_{T' \subset T} \mathcal{M}_P^{T'}$. In the first direction, suppose that M precisely satisfies T. By definition M satisfies T, so suppose for a contradiction that $M \in \mathcal{M}_P^{T'}$ for some $T' \subset T$. By definition, there exists at least one pair of types $(t_1, t_2) \in T$ which are not contained in T'. Since M precisely satisfies T', it follows that M does not contain any pair of matched agents with types t_1 and t_2, a contradiction.

In the other direction, suppose that

$$M \in \mathcal{M}^T \setminus \bigcup_{T' \subset T} \mathcal{M}_P^{T'}.$$

Suppose for a contradiction that M does not precisely satisfy T. By Observation 4.26, there exists a stable type set T' which is precisely satisfied by M. Since M satisfies T, it follows that $T' \subset T$. Hence, M is contained in the set $\bigcup_{T' \subset T} \mathcal{M}_P^{T'}$, a contradiction. □

The following corollary follows from Lemma 4.28 and Observation 4.26.

Corollary 4.29. *Let I be an instance of TYPED SMTI, and let T be a type set in $\mathcal{T}$. We have that*

$$|\mathcal{M}_P^T| = |\mathcal{M}^T| - \sum_{T' \subset T} |\mathcal{M}_P^{T'}|.$$

The following lemma uses the relationship described in Corollary 4.29 to bound the time needed to extract the number of matchings precisely satisfying a stable type set from the number of matchings satisfying the type set.

Lemma 4.30. *Let I be an instance of TYPED SMTI with at most n agents and k agent types. Given $\mathcal{T}$ and the value of $|\mathcal{M}^T|$ for each $T \in \mathcal{T}$, we can count the number of solutions to I in time $\mathcal{O}(2^{k^2(k+1)^2} n \log n)$.*

Proof. By Corollary 4.27, the number of solutions to I is equal to $\sum_{T \in \mathcal{T}} |\mathcal{M}_P^T|$. We first consider the time needed to extract the value of $|\mathcal{M}_P^T|$ from the value of $|\mathcal{M}^T|$ for each $T \in \mathcal{T}$. By Corollary 4.29, we have that

$$|\mathcal{M}_P^T| = |\mathcal{M}^T| - \sum_{T' \subset T} |\mathcal{M}_P^{T'}|. \tag{4.37}$$

For any $T \in \mathcal{T}$, we have that $|T'| < |T|$ for each $T' \subset T$. It follows that if we compute the value of $|\mathcal{M}_P^T|$ for each $T \in \mathcal{T}$ in ascending order of size, then we know the value of $|\mathcal{M}_P^{T'}|$ for each $T' \subset T$ when we compute $|\mathcal{M}_P^T|$.

Let T be a stable type set in I. Since the number of solutions in I is at most n^n, it follows that the value of each of $|\mathcal{M}_P^T|$ and $|\mathcal{M}^T|$ can be represented using at most $n \log n$ bits. Since each stable matching satisfies exactly one stable type set, it follows that the value of the sum in (4.37) is at most n^n. By Lemma 4.22, there are at most $2^{k(k+1)}$ type sets T' such that $T' \subset T$. It follows that computing the value of $|\mathcal{M}_P^T|$ from the value of $|\mathcal{M}^T|$ and each $|\mathcal{M}_P^{T'}|$ takes time $\mathcal{O}(2^{k(k+1)} n \log n)$. Since there are at most $2^{k(k+1)}$ stable type sets in $\mathcal{T}$, and at most n^n solutions in I, it follows from Corollary 4.27 that we can compute the number of solutions to I in time $\mathcal{O}(2^{k^2(k+1)^2} n \log n)$. □

We now describe how to reduce the problem of computing the values $|\mathcal{M}^T|$ for each $T \in \mathcal{T}$ to the problem of counting perfect matchings in a balanced bipartite graph. Let I be an instance of TYPED SMTI with at most n agents and at most k agent types, and let T be a stable type set in I. We use n_m and n_w to denote the number of men and women respectively in I. Observe that a matching M in I has size in the range $0 \leq |M| < \min(n_w, n_m)$. We will use c to denote the number of unmatched agents in the smaller set. Observe that $0 \leq c < \min(n_w, n_m)$. We construct a balanced bipartite graph $G_c^T = ((U, V), E)$ from I, T and c as follows. For each man m in I, we add a vertex u_m to U, and for each woman w in I, we add a vertex v_w to V. We add an additional $c + (\max(n_w, n_m) - n_m)$ vertices to U and $c + (\max(n_w, n_m) - n_w)$ vertices to V corresponding to dummy agents. Notice that $|U| = |V|$, so G_c^T is indeed balanced. Two vertices u_m and v_w are adjacent in G_c^T if and only if the corresponding man-woman pair are such that type(m)type$(w) \in T$. A vertex is adjacent to the set of dummy agents (of the opposite gender) if the corresponding agent

has type t and either $(t, k+1) \in T$ if the agent is a woman, or $(t, k+2) \in T$ if the agent is a man. Note that there are no edges in G_c^T with both endpoints corresponding to dummy agents since pairings of such agents are meaningless. We will refer to the graph G_c^T as a *type set graph*, and we will use $|\mathcal{M}_{G_c^T}|$ to denote the number of perfect matchings in G_c^T.

Notice that the number of possible type set graphs for an instance of TYPED SMTI with at most n agents and at most k agent types is bounded by the number of stable type sets in $\mathcal{T}$ (there are at most $2^{k(k+1)}$, and the number of possible values of c (which is at most n). In what follows, we show that the number of matchings admitted by I can be obtained directly from the number of perfect matchings in each possible type set graph. We first make the following observation of the neighbourhood diversity of a type set graph.

Lemma 4.31. *Let I be an instance of TYPED SMTI with at most k agent types, and let T be a type set in $\mathcal{T}$. For each $0 \leq c < \min(n_w, n_m)$, the type set graph G_c^T has neighbourhood diversity at most $(k+2)$.*

Proof. Let V_i denote the set of vertices corresponding to type i agents (including the set of male and female dummy agents) for each $i \in [k+2]$. Observe that the neighbourhood of each vertex in V_i is the same for every such vertex. The result follows. □

Let I be an instance of TYPED SMTI and let T be a stable type set in I. The following lemma describes the relationship between the number of solutions admitted by I of a certain size which satisfy T, and the number of perfect matchings in the corresponding type set graph.

Lemma 4.32. *Let I be an instance of TYPED SMTI with n_w women and n_m men, and let T be a stable type set in $\mathcal{T}$. There are*

$$\frac{|\mathcal{M}_{G_c^T}|}{c!(|n_w - n_m| + c)!}$$

matchings of size $\min(n_w, n_m) - c$ which satisfy T.

Proof. In what follows, we show that the set of perfect matchings in G_c^T correspond to $c!(|n_w - n_m| + c)!$ matchings of size $\min(n_w, n_m) - c$ satisfying T. Let M_G be a perfect matching in G_c^T, and let M be the matching admitted by I formed by adding the pair (m, w) to M if $u_m v_w$ is an edge in M_G. By construction, M is a stable matching which satisfies T. Since G_c^T contains $(c + (\max(n_w, n_m) - n_m)$ vertices in U and $(c + (\max(n_w, n_m) - n_w)$ vertices in V corresponding to dummy agents (and no pairs of dummy agents are matched

in M_G), it follows that M has size

$$\begin{aligned}
&\frac{n_w + n_m - (c + (\max(n_w, n_m) - n_m)) - (c + (\max(n_w, n_m) - n_w))}{2} \\
&= \frac{2n_w + 2n_m - 2c - 2\max(n_w, n_m)}{2} \\
&= n_w + n_m - c - \max(n_w, n_m) \\
&= \min(n_w, n_m) - c.
\end{aligned}$$

If each of the dummy agents were distinct, then each perfect matching in G_c^T would correspond to a unique matching admitted by I. However, we are not concerned about which of the dummy agents an agent is matched with. Since there are $(c + (\max(n_w, n_m) - n_m)$ male dummy agents, and $(c + (\max(n_w, n_m) - n_w)$ female dummy agents, a unique matching admitted by I will correspond to

$$\begin{aligned}
&(c + (\max(n_w, n_m) - n_w))!(c + (\max(n_w, n_m) - n_m))! \\
&= c!(|n_w - n_m| + c)!
\end{aligned}$$

distinct perfect matchings in G_c^T. □

The following corollary follows directly from Lemma 4.32.

Corollary 4.33. *Let I be an instance of TYPED SMTI with n_w women and n_m men, and let T be a stable type set in $\mathcal{T}$. We have that*

$$|\mathcal{M}^T| = \sum_{0 \leq c < \min(n_w, n_m)} \frac{|\mathcal{M}_{G_c^T}|}{c!(|n_w - n_m| + c)!}.$$

The following lemma uses the relationship described in Corollary 4.33 to bound the time needed to count the number of matchings satisfying a particular stable type set.

Lemma 4.34. *Let I be an instance of TYPED SMTI with at most n agents and at most k agent types, and let T be a stable type set in $\mathcal{T}$. The value of $|\mathcal{M}^T|$ can be calculated in time $\mathcal{O}((n + k + 1)^{3(k+2)} n^3 (n-1)^2 k^2 4^{6k}))$.*

Proof. By Corollary 4.33, we have that

$$|\mathcal{M}^T| = \sum_{0 \leq c < \min(n_w, n_m)} \frac{|\mathcal{M}_{G_c^T}|}{c!(|n_w - n_m| + c)!}. \tag{4.38}$$

We first bound the time needed to calculate the value of each individual term in the sum. Since $c \leq n - 1$, we can compute the value of $c!$ in time $\mathcal{O}(n^2)$ [89]. Similarly, since

$|n_w - n_m| \leq n$, we can compute the value of $(|n_w - n_m| + c)!$ in time $\mathcal{O}(n^2)$. We can represent the value of each of $c!$ and $(|n_w - n_m| + c)!$ using $\mathcal{O}(n \log n)$ bits. It follows that computing the value of $c!(|n_w - n_m| + c)!$ takes time

$$\begin{aligned} &\mathcal{O}(2n^2 + n^2 \log^2 n) \\ &= \mathcal{O}(n^2 \log^2 n) \end{aligned} \tag{4.39}$$

and produces an integer value in the range

$$1 \leq c!(|n_w - n_m| + c)! \leq n^n (2n)^{2n}.$$

Thus, the value of $c!(|n_w - n_m| + c)!$ can be represented using $\mathcal{O}(n \log n)$ bits.

We construct the graph G_c^T from T and c by first adding a vertex to G_c^T for each agent in I, as well as at most $2n$ vertices corresponding to dummy agents. Hence, constructing the vertex set of G_c^T takes time $\mathcal{O}(n)$. For each man-woman pair (m, w) in I, we add the edge $u_m v_w$ to G_c^T if and only if (type(m)type$(w)) \in T$. Since T contains at most $(k+1)^2$ pairs of types (including dummy types), it follows that for each such pair (m, w), we can decide whether to add the edge $u_m v_w$ to G_c^T in time $\mathcal{O}(k^2)$. There can be at most $2n$ pairs of agents. Hence, adding the edges to G_c^T takes time $\mathcal{O}(nk^2)$. Thus, constructing G_c^T takes time

$$\begin{aligned} &\mathcal{O}(n + nk^2) \\ &= \mathcal{O}(nk^2). \end{aligned}$$

By Corollary 4.19, we can count the number of perfect matchings in a balanced bipartite graph with at most n vertices and neighbourhood diversity at most r in time

$$\mathcal{O}((n + r - 1)^{3r} n^2 (n-1)^2 r^2 4^{6r}).$$

By Lemma 4.31, for each $0 \leq c < \min(n_w, n_m)$, the graph G_c^T has neighbourhood diversity at most $(k+2)$. Hence, we can count the number of perfect matchings in G_c^T in time

$$\mathcal{O}((n + k + 1)^{3(k+2)} n^2 (n-1)^2 k^2 4^{6k}). \tag{4.40}$$

Since there are at most n^n perfect matchings admitted by any n vertex graph, we have that $0 \leq |\mathcal{M}_{G_c^T}| \leq n^n$ for any $0 \leq c < \min(n_w, n_m)$. Hence, the value of $|\mathcal{M}_{G_c^T}|$ can be represented using $\mathcal{O}(n \log n)$ bits. It then follows from (4.39) to (4.40) that, for each $1 \leq$

$c < \min(n_w, n_m)$, the value of

$$\frac{|\mathcal{M}_{G_c^T}|}{c!(|n_w - n_m| + c)!}$$

can be determined in time

$$\begin{aligned}&\mathcal{O}(n^2 \log^2 n + nk^2 + (n+k+1)^{3(k+2)} n^2 (n-1)^2 k^2 4^{6k} + n^2 \log^2 n) \\ &= \mathcal{O}((n+k+1)^{3(k+2)} n^2 (n-1)^2 k^2 4^{6k}). \end{aligned} \tag{4.41}$$

It remains to consider the time needed to compute the sum in (4.38). By definition, the value of $|\mathcal{M}^{\mathcal{T}}|$ is a positive integer with value at most n^n. By definition, all values in the summation are positive integers and must therefore also have value at most n^n. Since c takes up to n distinct values, it follows from (4.41) that calculating the value of $|\mathcal{M}^T|$ takes time

$$\begin{aligned}&\mathcal{O}(n \times ((n+k+1)^{3(k+2)} n^2 (n-1)^2 k^2 4^{6k} + n \log n)) \\ &\quad = \mathcal{O}((n+k+1)^{3(k+2)} n^3 (n-1)^2 k^2 4^{6k}).\end{aligned}$$

□

We are now ready to prove our main result.

Theorem 4.35. *Let I be an instance of TYPED SMTI with at most n agents and at most k agent types. We can count the number of stable matchings admitted by I in time $\mathcal{O}(2^{k^2(k+1)^2}(n+k+1)^{3(k+2)} n^3 (n-1)^2 k^2 4^{6k})$.*

Proof. By Corollary 4.23, we can obtain the set $\mathcal{T}$ of stable type sets in time

$$\mathcal{O}(2^{k(k+1)} k^2). \tag{4.42}$$

By Lemma 4.30, given the value of $|\mathcal{M}^T|$ for each $T \in \mathcal{T}$, we can count the number of solutions to I in time

$$\mathcal{O}(2^{k^2(k+1)^2} n \log n). \tag{4.43}$$

By Lemma 4.22, the set $\mathcal{T}$ of stable type sets has size at most $2^{k(k+1)}$. Thus, it follows from Lemma 4.34 that we can calculate the value of $|\mathcal{M}^T|$ for every $T \in \mathcal{T}$ in time

$$\mathcal{O}(2^{k(k+1)} (n+k+1)^{3(k+2)} n^3 (n-1)^2 k^2 4^{6k}). \tag{4.44}$$

Finally, it follows from (4.42), (4.43) and (4.44) that solving #TYPED SMTI takes time

$$\begin{aligned}&\mathcal{O}(2^{k(k+1)}k^2 + 2^{k(k+1)}(n+k+1)^{3(k+2)}n^3(n-1)^2k^2 4^{6k} + 2^{k^2(k+1)^2} n \log n)\\ &= \mathcal{O}(2^{k^2(k+1)^2}(n+k+1)^{3(k+2)}n^3(n-1)^2k^2 4^{6k}).\end{aligned}$$

□

4.6 #TYPED SRTI is in XP

In this section, we consider the problem of counting solutions to an instance of TYPED SRTI. We define this problem as follows.

#TYPED SRTI	
Input:	An instance I of TYPED SRTI containing at most n agents with at most k agent types.
Parameter:	k.
Question:	How many stable matchings does I admit?

In what follows, we prove that #TYPED SRTI belongs to XP parameterised by the number of types needed to describe an instance. The proof of this result follows a broadly similar approach to that of Theorem 4.35.

4.6.1 Proof Overview

As in the SMTI setting, our result is achieved via a reduction to the problem of counting perfect matchings in graphs with bounded neighbourhood diversity. However, due to the different definition of stability in this setting, we cannot determine the stability of a matching by looking only at the pairs of types of agents in the matching - we also need to know how many pairs of types are realised by more than one pair of agents. Moreover, since agents may be matched with a partner of the same type, it follows that the acceptability graph associated with an instance does not necessarily have bounded neighbourhood diversity.

In this setting, a "stable type function" describes the number of pairs of agents that may be realised by each pair of types in a stable matching. The number of stable type functions in an instance of TYPED SRTI is bounded by a function of the number of agent types. We will see that we can count solutions to an instance of TYPED SRTI by counting the number of matchings which "satisfy" a stable type function. For this, we first count the number of ways to match together pairs of agents of the same type. We then count the number of ways to select a single pair of agents (from the remaining set) for each pair of types which can be realised by at most one pair of agents. It remains to count the number of matchings of the remaining set of agents by counting perfect matchings in the reduced acceptability graph. We will see that this graph has bounded neighbourhood diversity.

For general graphs, counting the number of perfect matchings is computationally equivalent to computing the "hafnian" of the associated adjacency matrix. Our result then follows from a result due to Björklund et al. [91], who demonstrated that the problem of computing the hafnian of a square matrix belongs to XP parameterised by the rank of the matrix. Note that

we do not analyse the time needed to compute the hafnian at the same level of detail as we did for computing the permanent in Section 4.5. As a consequence, in this setting our final runtime bound contains an unknown constant.

In Section 4.6.2, we show that the number of solutions to an instance of TYPED SRTI can be written in terms of the number of matchings satisfying each stable type function. In Section 4.6.3, we show how to count the number of matchings satisfying a stable type function. Finally, in Section 4.6.4 we use these results to describe a reduction from #TYPED SRTI to the problem of counting perfect matchings in a graph parameterised by the neighbourhood diversity of the graph.

4.6.2 Stable Type Functions

In this section, we show that the stability of a matching admitted by an instance of TYPED SRTI can be described by a function from the set of pairs of types to the number of pairs of matched agents with these types. Akin to the role of stable type sets from the stable marriage setting, type functions allow us to describe the number of solutions in terms of the number of matchings satisfying each possible stable type function.

Let I be an instance of TYPED SRTI with at most k agent types, and let M be a matching admitted by I. If an agent x with type i is unmatched under M, then for notational convenience we may say that x is matched to a dummy agent of type $(k+1)$. Note that type $(k+1)$ is considered less desirable to each type of agent than any other type in their preference list. Let $\text{worst}_M(i)$ and $\text{second\_worst}_M(i)$ denote the types of the least desirable and second least desirable agents matched to any agent of type i in M. If there is only one agent of type i, then we set $\text{second\_worst}_M(i) = \varnothing$. Note that if there are two (or more) pairs (x, y) and (x', y') in M such that $\text{type}(x) = \text{type}(x') = i$ and $\text{type}(y) = \text{type}(y') = \text{worst}_M(i)$ then we have that $\text{worst}_M(i) = \text{second\_worst}_M(i)$. The following result due to Meeks and Rastegari [11] defines the stability of a matching admitted by I in terms of the values of $\text{worst}_M(i)$ and $\text{second\_worst}_M(i)$ for each $i \in [k]$.

Lemma 4.36 ([11])**.** *Let I be an instance of TYPED SRTI with at most k agent types. A matching M in I is stable if and only if*

- *there is no pair of types $(i, j) \in [k]^2$ with $i \neq j$ such that $j \succ_i \text{worst}_M(i)$ and $i \succ_j \text{worst}_M(j)$, and*
- *there is no type $i \in [k]$ such that $\text{second\_worst}_M(i) \neq \varnothing$ and $i \succ_i \text{second\_worst}_M(i)$.*

It follows from Lemma 4.36 that to determine whether a matching is stable, we need to know which pairs of types of agents are present in the matching, and also which pairs of types are realised by more than one pair of agents in the matching. In this setting, we define a *type*

function as a function $f : [k+1]^2 \to \{0, 1, 2\}$ where, for each $(t_1, t_2) \in [k+1]^2$, we have that $f(t_1, t_2) = f(t_2, t_1)$ and

- $f(t_1, t_2) = 0$ if there are to be no pairs of matched agents with types t_1 and t_2 respectively, and
- $f(t_1, t_2) = 1$ if there may be at most one pair of matched agents with types t_1 and t_2 respectively, and
- $f(t_1, t_2) = 2$ if there may be any number of pairs of matched agents with types t_1 and t_2 respectively.

Note that $f(k+1, k+1) = 0$ for any type function f since the type $(k+1)$ is used to represent dummy agents. For each type $i \in [k]$, we define $\text{worst}_f(i)$ as least desirable type of agent from the perspective of type i such that $f(i, \text{worst}_f(i)) \neq 0$. If $f(i, \text{worst}_f(i)) = 1$, then let $\text{second\_worst}_f(i)$ be the second least desirable type (from the perspective of type i) such that $f(i, \text{second\_worst}_f(i)) \neq 0$. Otherwise, set $\text{second\_worst}_f(i) = \text{worst}_f(i)$. We call f a *stable type function* if

- there are no pairs $(i, j) \in [k]^2$ with $i \neq j$ such that $j \succ_i \text{worst}_f(i)$ and $i \succ_j \text{worst}_f(j)$, and
- there is no type $i \in [k]$ such that $\text{second\_worst}_f(i) \neq \varnothing$ and $i \succ_i \text{second\_worst}_f(i)$.

The following observation follows from the fact that we can determine the stability of a type function by comparing each pair (i, j) of types to the values of $\text{worst}_f(i)$ and $\text{second\_worst}_f(i)$, and $\text{worst}_f(j)$ and $\text{second\_worst}_f(j)$.

Observation 4.37. Let I be an instance of TYPED SRTI with at most k agent types. The stability of a type function can be determined in time $\mathcal{O}(k^2)$.

In the following lemma, we bound the number of possible type functions in terms of the number of agent types needed to describe an instance.

Lemma 4.38. *Let I be an instance of TYPED SRTI with at most k agent types. There are at most $3^{k(k+1)}$ type functions over the set of types in I.*

Proof. Let f be a type function in I. By definition, we have that $f(k+1, k+1) = 0$. For each $i \in [k]$ and each $j \in [k+1]$, there are at most 3 possible values of $f(i, j)$. It follows that there can be at most $3^{k(k+1)}$ type functions. □

We use $\mathcal{F}$ to denote the set of stable type functions in an instance of TYPED SRTI. The following corollary bounds the time needed to obtain the set $\mathcal{F}$ in an instance of TYPED SRTI with at most k types, and follows directly from Observation 4.37 and Lemma 4.38.

Corollary 4.39. *Let I be an instance of TYPED SRTI with at most k agent types. The set $\mathcal{F}$ of stable type functions can be obtained in time $\mathcal{O}(3^{k(k+1)}k^2)$.*

Let I be an instance of TYPED SRTI. We say that a matching M in I *satisfies* a type function f if, for each $(t_1, t_2) \in [k]^2$, we have that

- if $f(t_1, t_2) = 0$, then there are no pairs of agents with types t_1 and t_2 respectively who are matched together under M, and
- if $f(t_1, t_2) = 1$, then there is at most one pair of agents with types t_1 and t_2 respectively who are matched together under M.

In addition, for each $t \in [k]$, we have that

- if $f(t, k+1) = 0$, then all agents of type t are matched under M, and
- if $f(t, k+1) = 1$, then there is at most one unmatched agent of type t under M.

We have the following lemma.

Lemma 4.40. *Let I be an instance of TYPED SRTI, and let f be a type function in $\mathcal{F}$. Let M be a matching satisfying f. For any $i \in [k]$, we have that*

$$\textit{worst}_M(i) \succeq_i \textit{worst}_f(i).$$

In addition, if $\textit{second\_worst}_M(i) = \varnothing$, then $\textit{second\_worst}_f(i) = \varnothing$. If $\textit{second\_worst}_M(i) \neq \varnothing$, then

$$\textit{second\_worst}_M(i) \succeq_i \textit{second\_worst}_f(i).$$

Proof. We first consider the values of $\text{worst}_M(i)$ and $\text{worst}_f(i)$ for each $i \in [k]$. Suppose for a contradiction that $\text{worst}_M(i) \prec_i \text{worst}_f(i)$ for some $i \in [k]$. By definition, the value of $\text{worst}_M(i)$ is equal to the least desirable type (from the perspective of type i) such that a pair of agents of types $(i, \text{worst}_M(i))$ are matched together under M. Since we have assumed $\text{worst}_M(i) \prec_i \text{worst}_f(i)$, it follows from the definition of f that $f(i, \text{worst}_M(i)) = 0$, a contradiction of the assumption that M satisfies f.

Now suppose that $\text{second\_worst}_M(i) \prec_i \text{second\_worst}_f(i)$, so we must have that $\text{second\_worst}_f(i) \neq \text{worst}_f(i)$ and $\text{second\_worst}_M(i)\text{worst}_M(i) = \text{worst}_f(i)$. Since $\text{second\_worst}_f(i) \neq \text{worst}_f(i)$, it follows from the definition of f that $f(i, \text{worst}_f(i)) = 1$. However, since $\text{second\_worst}_M(i) = \text{worst}_M(i)$, there must be at least two pairs of matched agents in M with types $(i, \text{worst}_M(i))$, a contradiction of the assumption that M satisfies f. □

We shall use $\mathcal{M}^f$ to denote the set of matchings admitted by I which satisfy the stable type function f. The number of matchings satisfying f is denoted by $|\mathcal{M}^f|$. We say that a

matching M in I *precisely satisfies* a stable type function f if, for each $(t_1, t_2) \in [k]^2$, we have that

- if $f(t_1, t_2) = 0$ then there are no pairs of agents with types t_1 and t_2 respectively who are matched together under M, and
- if $f(t_1, t_2) = 1$ then there is exactly one pair of agents with types t_1 and t_2 respectively who are matched together under M, and
- if $f(t_1, t_2) = 2$ then there is more than one pair of agents with types t_1 and t_2 respectively who are matched together under M.

In addition, for each $t \in [k]$, we have that

- if $f(t, k+1) = 0$ then there are no unmatched agents of type t under M, and
- if $f(t, k+1) = 1$ then there is exactly one unmatched agent of type t under M, and
- if $f(t, k+2) = 2$ then there is more than one unmatched agent of type t under M.

We use $\mathcal{M}_P^f$ to denote the set of matchings precisely satisfying a stable type function f. The number of matchings precisely satisfying f is denoted by $|\mathcal{M}_P^f|$. The following Corollary follows from Lemma 4.40.

Corollary 4.41. *All matchings precisely satisfying a stable type function are stable.*

The following observation follows directly from definitions.

Observation 4.42. Each stable matching precisely satisfies a unique stable type function.

We can combine Corollary 4.41 and Observation 4.42 to obtain the following relationship between the number of solutions to instance of TYPED SRTI and the number of matchings precisely satisfying each stable type function.

Corollary 4.43. *Let I be an instance of TYPED SRTI. The number of solutions to I is equal to $\sum_{f \in \mathcal{F}} |\mathcal{M}_P^f|$.*

Let I be an instance of TYPED SRTI. We say that a type function f_1 in I is *contained* in another type function f_2 if, for each pair $(t_1, t_2) \in [k+1]^2$ we have that $f_1(t_1, t_2) \leq f_2(t_1, t_2)$, and there exists at least one pair $(t_1, t_2) \in [k+1]^2$ such that $f_1(t_1, t_2) < f_2(t_1 t_2)$. We write this as $f_1 \subset f_2$. We say that a type function f_1 is *smaller* than a type function f_2 if

$$\sum_{(t_1,t_2) \in [k+1]^2} f_1(t_1, t_2) < \sum_{(t_1,t_2) \in [k+1]^2} f_2(t_1 t_2).$$

Note that each type function f_1 contained in a type function f_2 is smaller than f_2. The following results describe how to obtain the value of $|\mathcal{M}_P^f|$ from the value of $|\mathcal{M}^f|$ for each $f \in \mathcal{F}$. We will then show that the problem of calculating the value of $|\mathcal{M}^f|$ belongs to XP parameterised by the number of agent types.

Lemma 4.44. *Let I be an instance of TYPED SRTI and let f be a type function in $\mathcal{F}$. We have that*

$$\mathcal{M}_P^f = \mathcal{M}^f \setminus \bigcup_{f' \subset f} \mathcal{M}_P^{f'}.$$

Proof. We proceed by showing that any matching M in I belongs to the set $\mathcal{M}_P^f$ if and only if it belongs to the set $\mathcal{M}^f \setminus \bigcup_{f' \subset f} \mathcal{M}_P^{f'}$. In the first direction, suppose that M precisely satisfies f. By definition M satisfies f, so suppose for a contradiction that $M \in \mathcal{M}_P^{f'}$ for some $f' \subset f$. Since $f' \subset f$, there exists at least one pair of types $(t_1, t_2) \in [k]^2$ such that $f'(t_1, t_2) < f(t_1, t_2)$. It follows that M does not precisely satisfy f, a contradiction.

In the other direction, suppose that $M \in \mathcal{M}^f \setminus \bigcup_{f' \subset f} \mathcal{M}_P^{f'}$. Suppose for a contradiction that M does not precisely satisfy f. By Observation 4.42, there exists a type function f' which is precisely satisfied by M. Since M satisfies f, it follows from the definition of containment that $f' \subset f$. Hence, M is contained in the set $\bigcup_{f' \subset f} \mathcal{M}_P^{f'}$, a contradiction. □

The following corollary follows from Lemma 4.44 and our earlier observation (Observation 4.42) that every stable matching precisely satisfies a unique stable type function.

Corollary 4.45. *Let I be an instance of TYPED SRTI, and let f be a stable type function in $\mathcal{F}$. We have that*

$$|\mathcal{M}_P^f| = |\mathcal{M}^f| - \sum_{f' \subset f} |\mathcal{M}_P^{f'}|.$$

The following lemma uses the relationship described in Corollary 4.45 to bound the time needed to extract the number of matchings precisely satisfying a particular stable type function from the number of matchings satisfying each stable type function.

Lemma 4.46. *Let I be an instance of TYPED SRTI with at most n agents and at most k agent types. Given $\mathcal{F}$ and the value of $|\mathcal{M}^f|$ for each $f \in \mathcal{F}$, we can count the number of solutions to I in time $\mathcal{O}(3^{k^2(k+1)^2} n \log n)$.*

Proof. By Corollary 4.43, the number of solutions to I is equal to the sum

$$\sum_{f \in \mathcal{F}} |\mathcal{M}_P^f|. \tag{4.45}$$

We first consider the time needed to obtain the value of $|\mathcal{M}_P^f|$ for each $f \in \mathcal{F}$. By Corollary 4.45, we have that

$$|\mathcal{M}_P^f| = |\mathcal{M}^f| - \sum_{f' \subset f} |\mathcal{M}_P^{f'}|. \tag{4.46}$$

For any $f \in \mathcal{F}$, each type function f' contained in f is smaller than f. It follows that if we compute the value of $|\mathcal{M}_P^f|$ for each $f \in \mathcal{F}$ in ascending order of size, then we know the value of $|\mathcal{M}_P^{f'}|$ for each $f' \subset f$ when we compute $|\mathcal{M}_P^f|$.

The number of solutions in I is at most n^n. It follows that the value of each of $|\mathcal{M}_P^f|$ and $|\mathcal{M}^f|$ can be represented using at most $n \log n$ bits. Since each stable matching precisely satisfies a unique stable type function, it follows that the value of the sum in (4.46) is at most n^n, and so can also be represented using at most $n \log n$ bits. By Lemma 4.38, there are at most $3^{k(k+1)}$ stable type functions f' such that $f' \subset f$. It follows that computing the sum in (4.46) takes time $\mathcal{O}(3^{k(k+1)} n \log n)$, and computing the subtraction takes time $\mathcal{O}(n \log n)$. Hence, we can compute the value of $|\mathcal{M}_P^f|$ in time

$$\begin{aligned} &\mathcal{O}(3^{k(k+1)} n \log n + n \log n) \\ &= \mathcal{O}(3^{k(k+1)} n \log n). \end{aligned} \tag{4.47}$$

Since there are at most $3^{k(k+1)}$ stable type functions in $\mathcal{F}$, and at most n^n solutions in I, it follows from (4.45) and (4.47) that we can compute the number of solutions to I in time

$$\begin{aligned} &\mathcal{O}(3^{k(k+1)}(3^{k(k+1)} n \log n + n \log n)) \\ &= \mathcal{O}(3^{k^2(k+1)^2} n \log n). \end{aligned}$$

□

4.6.3 Counting Matchings Satisfying a Stable Type Function

In this section, we describe how to count the number of matchings satisfying a stable type function. Let I be an instance of TYPED SRTI with at most n agents and at most k agent types, and let f be a stable type function in $\mathcal{F}$. To count the number of matchings satisfying f, we first count the number of ways to assign a subset of agents in I to a partner of their own type. We then count the number of ways to match together at most one pair of agents (from the remaining set of agents) of types i and j respectively for each pair $(i, j) \in [k+1]^2$ such that $i < j$ and $f(i, j) = 1$. Finally, we show that the problem of counting the number of matchings of the remaining set of agents can be reduced to the problem of counting perfect matchings in a graph with bounded neighbourhood diversity. In the next section, we will make use of these observations to show that the problem of counting the number of solutions to I belongs to XP parameterised by the number of agent types.

Let I be an instance of TYPED SRTI with at most n agents and at most k agents types, and let f be a stable type function in $\mathcal{F}$. We first describe how to count the number of ways in which a subset of agents in I could be matched with a partner of their own type in a matching

satisfying f. We will require some additional notation. Let n_i denote the number of type i agents in I for each $i \in [k]$. Let $d \in \{0, \ldots, n-1\}$ denote the number of unmatched agents. Note that a matching in which d agents are unmatched has cardinality $(n-d)/2$, so we require that $(n-d)$ is even. For each $i \in [k]$, we use m_i to denote the number of pairs of agents of type i who are matched with a partner of their own type. We require that $m_i \leq \lfloor n_i/2 \rfloor$ and $m_i \leq f(i,i)$. In addition, we must have that $\sum_{i\in[k]} m_i \leq (n-d)/2$. For each $d \in \{0, \ldots, n-1\}$ such that $(n-d)$ is even, we define the set M_d^f to contain all sets $\mathbf{m} = \{m_1, \ldots, m_k\}$ which meet these requirements. Given some $\mathbf{m} \in M_d^f$, the following lemma describes the number of ways that we can match a subset of the agents in I to a partner of their own type.

Lemma 4.47. *Let I be an instance of TYPED SRTI with at most n agents and at most k types, and let f be a stable type function in $\mathcal{F}$. Let $d \in \{0, \ldots, n-1\}$ be such that $(n-d)$ is even, and let $\mathbf{m}$ be a set in M_d^f. The number of ways to select $\sum_{i\in[k]} m_i$ pairs of same-type agents from I containing m_i pairs of type i for each $i \in [k]$ is equal to*

$$\prod_{i\in[k]} \frac{n_i!}{(n_i - 2m_i)!(m_i)!2^{m_i}}$$

Proof. For each $i \in [k]$, we must choose $2m_i$ agents of type i from a set of size n_i, and assign them into unordered pairs. The number of ways to select $2m_i$ items from a set of size n_i is equal to $n_i!/(n_i - 2m_i)!$. Since we are not interested in the order of the m_i pairs, or the order of the agents within each pair, we must divide this value by $(m_i)!2^{m_i}$. The result follows. □

In the following lemma, we describe an upper bound on the time needed to count the number of ways to select $\sum_{i\in[k]} m_i$ pairs of same-type agents from I with m_i pairs of type i for each $i \in [k]$.

Lemma 4.48. *Let I be an instance of TYPED SRTI with at most n agents and at most k types. Let f be a stable type function in $\mathcal{F}$. Let $d \in \{0, \ldots, n-1\}$ be such that $(n-d)$ is even, and let $\mathbf{m}$ be a set in M_d^f. We can count the number of ways to select $\sum_{i\in[k]} m_i$ pairs of same-type agents from I with m_i pairs of type i for each $i \in [k]$ in time $\mathcal{O}(kn^2 \log^2 n)$.*

Proof. By Lemma 4.47, the number of ways to select $\sum_{i\in[k]} m_i$ pairs of same-type agents from I with m_i pairs of type i for each $i \in [k]$ is equal to

$$\prod_{i\in[k]} \frac{n_i!}{(n_i - 2m_i)!(m_i)!2^{m_i}}. \tag{4.48}$$

Since $n_i, m_i \leq n$, it follows that each of $n_i!$, $(n_i - 2m_i)!$ and $(m_i)!$ can be computed in time $\mathcal{O}(n^2)$ [89] and can be represented using at most $n \log n$ bits. Computing the value of 2^{m_i}

takes time $\mathcal{O}(n^2)$, and the value can be represented using at most n bits. It follows that the value of the product $(n_i - 2m_i)!(m_i)!2^{m_i}$ can be represented using $\mathcal{O}(n \log n)$ bits, and can be computed in time $\mathcal{O}(n^2 \log^2 n)$. It follows that computing the value of each term in (4.48) takes time $\mathcal{O}(n^2 \log^2 n)$. Since each of the at most k terms in (4.48) is a positive integer, and the overall product has value at most n^n, it follows that we can compute the value of (4.48) in time $\mathcal{O}(kn^2 \log^2 n)$. □

We now describe how to count the number of ways to match together types of agents from the remaining set when at most one such pair of allowed. Specifically, let $\mathbf{m} = \{m_1, \ldots, m_k\}$ be a set in M_d^f. Let $I_{\mathbf{m}}$ be the instance formed from removing (any) $2m_i$ agents of type i from I for each $i \in [k]$. For each pair $(i, j) \in [k]^2$ such that $i \neq j$ and $f(i, j) = 1$, we wish to count the number of ways to match together a single pair of agents of type i and j respectively, given that we have already matched $2m_i$ agents of type i and $2m_j$ agents of type j. In addition, for each $i \in [k]$ such that $f(i, (k+1)) = 1$, we wish to count the number of ways to choose at most one agent of type i who will not be unmatched.

For each pair $(i, j) \in [k+1]^2$ with $i < j$, let $p_{i,j}$ denote the number of pairs of agents of types i and j respectively who are matched together in a matching of size $n - d$. For each $i \in [k]$, we require that $p_{i,j} \leq f(i, j)$. In addition, since we have already matched m_i agents of type i to a partner of their own type, we require that

$$\sum_{j<i} p_{j,i} + \sum_{j \leq k} p_{i,j} + 2m_i \leq n_i.$$

We also require that

$$\sum_{(i,j) \in [k]^2} p_{i,j} + \sum_{i \in [k]} 2m_i \leq n - d$$

and

$$\sum_{i \in [k]} p_{i,k+1} \leq d.$$

Let $\mathbf{p_m} = \{p_{1,2}, p_{1,3}, \ldots, p_{k,k+1}\}$. We use $P_{\mathbf{m}}$ to denote the set containing all sets $\mathbf{p_m}$ which meet the above requirements. Given some $\mathbf{p_m} \in P_{\mathbf{m}}$, the following lemma describes the number of ways to select $p_{i,j}$ pairs of agents of types i and j respectively from $I_{\mathbf{m}}$ for each $(i, j) \in [k+1]^2$ such that $f(i, j) = 1$.

Lemma 4.49. *Let I be an instance of TYPED SRTI with at most n agents and at most k types. Let f be a stable type function in $\mathcal{F}$, and let $d \in \{0, \ldots, n-1\}$ be a value such that $(n-d)$ is even. Let $\mathbf{m} = \{m_1, \ldots, m_k\}$ be a set in M_d^f, and let $\mathbf{p_m} = \{p_{1,2}, \ldots, p_{k-1,k}\}$ be a set in $P_{\mathbf{m}}$. The number of ways to select $p_{i,j}$ pairs of agents of types i and j respectively from $I_{\mathbf{m}}$ for each $(i,j) \in [k+1]^2$ such that $f(i,j) = 1$ is equal to*

$$\frac{\prod_{i \in [k]} \frac{(n_i - 2m_i)!}{(n_i - 2m_i - \sum_{j<i} p_{j,i} - \sum_{j \leq k+1} p_{i,j})!}}{2^{\sum_{1 \leq i < j \leq k} p_{i,j}} (\sum_{1 \leq i < j \leq k+1} p_{i,j})!}.$$

Proof. The total number of agents of type i that we wish to select is equal to

$$\sum_{j<i} p_{j,i} + \sum_{j \leq k+1} p_{i,j}.$$

Since there are a total of $(n_i - 2m_i)$ agents of type i to choose from, it follows that the number of ways to select the type i agents is equal to

$$\frac{(n_i - 2m_i)!}{(n_i - 2m_i - \sum_{j<i} p_{j,i} - \sum_{j \leq k+1} p_{i,j})!}.$$

It follows that the total number of ways to select the agents is equal to

$$\prod_{i \in [k]} \frac{(n_i - 2m_i)!}{(n_i - 2m_i - \sum_{j<i} p_{j,i} - \sum_{j \leq k+1} p_{i,j})!}. \tag{4.49}$$

The total number of pairs of (non-dummy) agents selected is equal to $\sum_{1 \leq i < j \leq k} p_{i,j}$. The total number of pairs of agents (including each agent matched to a dummy agent) selected is equal to $\sum_{1 \leq i < j \leq k+1} p_{i,j}$. Since we are not interested in the order of the agents within each pair of agents, or the ordering of the pairs of agents (including each agent matched to a dummy agent), it follows that we must divide (4.49) by $2^{\sum_{1 \leq i < j \leq k} p_{i,j}} (\sum_{1 \leq i < j \leq k+1} p_{i,j})!$. The result follows. □

Let $\mathbf{p_m}$ be a set in $P_{\mathbf{m}}$. In the following lemma, we describe an upper bound on the time needed to compute the number of ways to select $p_{i,j}$ pairs of agents of types i and j respectively from $I_{\mathbf{m}}$ for each $(i,j) \in [k+1]^2$ such that $f(i,j) = 1$.

Lemma 4.50. *Let I be an instance of TYPED SRTI with at most n agents and at most k types. Let f be a stable type function in $\mathcal{F}$, and let $d \in \{0, \ldots, n-1\}$ be a value such that $(n-d)$ is even. Let $\mathbf{m} = \{m_1, \ldots, m_k\}$ be a set in M_d^f, and let $\mathbf{p_m} = \{p_{1,2}, \ldots, p_{k-1,k}\}$ be a set in $P_{\mathbf{m}}$. The number of ways to select $p_{i,j}$ pairs of agents of types i and j respectively from $I_{\mathbf{m}}$ for each $(i,j) \in [k+1]^2$ with $f(i,j) = 1$ can be computed in time $\mathcal{O}(k^3 n^2 \log^2 n)$.*

Proof. By Lemma 4.49, the number of ways to select $p_{i,j}$ pairs of agents of types i and j

respectively from $I_{\mathbf{m}}$ for each $(i,j) \in [k+1]^2$ such that $f(i,j) = 1$ is equal to

$$\frac{\prod_{i\in[k]} \frac{(n_i-2m_i)!}{(n_i-2m_i-\sum_{j<i} p_{j,i}-\sum_{j\leq k+1} p_{i,j})!}}{2^{\sum_{1\leq i<j\leq k} p_{i,j}} (\sum_{1\leq i<j\leq k+1} p_{i,j})!}. \tag{4.50}$$

We first consider the time needed to compute the value of

$$\frac{(n_i-2m_i)!}{(n_i-2m_i-\sum_{j<i} p_{j,i}-\sum_{j\leq k+1} p_{i,j})!} \tag{4.51}$$

for each $i \in [k]$. By definition, we have that $0 \leq p_{i,j} \leq 1$ for any $(i,j) \in [k+1]^2$. It follows that the sums $\sum_{j<i} p_{j,i}$ and $\sum_{j\leq k+1} p_{i,j}$ have value at most k each, and so can be computed in time $\mathcal{O}(k^2)$. Computing the value of $2m_i$ takes time $\mathcal{O}(\log n)$. Since each of the values in $(n_i - 2m_i - \sum_{j<i} p_{j,i} - \sum_{j\leq k+1} p_{i,j})$ is at most n, it follows that computing every subtraction takes time $\mathcal{O}(\log n)$. Since $(n_i - 2m_i - \sum_{j<i} p_{j,i} - \sum_{j\leq k+1} p_{i,j}) \leq n$, it follows that computing the factorial $(n_i - 2m_i - \sum_{j<i} p_{j,i} - \sum_{j\leq k+1} p_{i,j})!$ takes time $\mathcal{O}(n^2)$. Hence, computing the value of $(n_i - 2m_i - \sum_{j<i} p_{j,i} - \sum_{j\leq k+1} p_{i,j})!$ takes time

$$\begin{aligned}&\mathcal{O}(k^2 + 2\log n + n^2)\\ &= \mathcal{O}(k^2 n^2).\end{aligned}$$

Note that the value of $(n_i - 2m_i - \sum_{j<i} p_{j,i} - \sum_{j\leq k+1} p_{i,j})!$ is at most n^n. The value of $(n_i - 2m_i)!$ is also at most n^n, and can be computed in time $\mathcal{O}(n^2)$. Hence, computing the value of (4.51) takes time

$$\begin{aligned}&\mathcal{O}(k^2 n^2 + n^2 + n^2 \log^2 n)\\ &\mathcal{O}(k^2 n^2 \log^2 n)\end{aligned}$$

for each $i \in [k]$. This value is at most n^n for each $i \in [k]$. Thus, the value of

$$\prod_{i\in[k]} \frac{(n_i-2m_i)!}{(n_i-2m_i-\sum_{j<i} p_{j,i}-\sum_{j\leq k+1} p_{i,j})!}$$

is at most n^{nk}, and can be computed in time

$$\begin{aligned}&\mathcal{O}(k \times (k^2 n^2 \log^2 n + nk \log n \times n \log n))\\ &\mathcal{O}(k^3 n^2 \log^2 n).\end{aligned} \tag{4.52}$$

We now consider the time needed to compute the value of

$$2^{\sum_{1\leq i<j\leq k} p_{i,j}} \left(\sum_{1\leq i<j\leq k+1} p_{i,j} \right)!. \tag{4.53}$$

Since there are at most k^2 values in the sum $\sum_{1\leq i<j\leq k} p_{i,j}$, and each value is at most 1, it follows that we can compute the sum in time $\mathcal{O}(k^3)$. The value of $2^{\sum_{1\leq i<j\leq k} p_{i,j}}$ is at most 2^{k^2} so, (once the value of $\sum_{1\leq i<j\leq k} p_{i,j}$ is known) computing it takes time $\mathcal{O}(k^4)$. Hence, computing the value of $2^{\sum_{1\leq i<j\leq k} p_{i,j}}$ takes time

$$\begin{aligned} &\mathcal{O}(k^3 + k^4) \\ &= \mathcal{O}(k^4). \end{aligned} \tag{4.54}$$

The value of $\sum_{1\leq i<j\leq k+1} p_{i,j}$ is at most $k(k+1)$, so computing it takes time $\mathcal{O}(k^3)$. Computing the value of $(k(k+1))!$ takes time $\mathcal{O}(k^4)$. It follows that computing the value of $(\sum_{1\leq i<j\leq k+1} p_{i,j})!$ takes time

$$\begin{aligned} &\mathcal{O}(k^3 + k^4) \\ &= \mathcal{O}(k^4). \end{aligned} \tag{4.55}$$

The value of $(\sum_{1\leq i<j\leq k+1} p_{i,j})!$ is at most $(k(k+1))^{k(k+1)}$. It follows that computing the product in (4.53) takes time

$$\begin{aligned} &\mathcal{O}(k^2 \log(k^2) \times k^2)) \\ &= \mathcal{O}(k^5). \end{aligned} \tag{4.56}$$

It follows from (4.54), (4.55) and (4.56) that computing the value of (4.53) takes time

$$\begin{aligned} &\mathcal{O}(2 \times k^4 + k^5) \\ &= \mathcal{O}(k^5). \end{aligned} \tag{4.57}$$

Moreover, the value of (4.53) is at most $2^n k(k+1)$. Finally, it follows from (4.52) and (4.57) that computing the value of (4.50) takes time

$$\begin{aligned} &\mathcal{O}(k^3 n^2 \log^2 n + k^5 + nk \log n \times (n + \log(k(k+1)))) \\ &= \mathcal{O}(k^3 n^2 \log^2 n). \end{aligned}$$

□

Let I be an instance of TYPED SRTI with at most k agent types, and let f be a stable type

function in $\mathcal{F}$. Using the above observations, we now describe how to count the number of matchings admitted by I which satisfy f. Since agents of the same type have identical preferences and are considered equally desirable by their set of available partners, we can count the number of such matchings by separately computing

- the number of ways to select a subset of agents and assign them into same-type pairs, and
- the number of ways to choose at most a single pair of agents of types i and j respectively for each $1 \leq i < j \leq k+1$ with $f(i,j) = 1$ from among the remaining set of agents, and
- the number of matchings of the remaining number of agents of each type into pairs of different types which satisfy f.

In Lemma 4.47, we described how to compute the number of ways to select a subset of agents from I and assign them into same-type pairs. In Lemma 4.49, we described how to count the number of ways to match together a single pair of agents of types i and j respectively for each $1 \leq i < j \leq k+1$ with $f(i,j) = 1$ from among the remaining set of agents.

In what follows, we describe how to count the number of stable matchings of the remaining set of agents of each type into pairs of different types. As in the stable marriage setting, this is achieved by counting the number of perfect matchings in a graph constructed from a problem instance and a stable type function. We will see that the graph formed from the (remaining) set of agents has bounded neighbourhood diversity. As a consequence, we can efficiently count the set of perfect matchings in the graph. Since all agents of the same type have identical preference lists and are seen as equally desirable by all agents, our "reduced" instance can be formed by removing any subset of the agents which meet the constraints given by $\mathbf{m}$ and $\mathbf{p_m}$. As such, we are able to break down the problem of counting stable matchings admitted by an instance of TYPED SRTI into three efficiently solvable problems.

Let I be an instance of TYPED SRTI with at most n agents and at most k types. Let f be a stable type function in $\mathcal{F}$, and let $d \in \{0, \ldots, n-1\}$ be a value such that $(n-d)$ is even. Let $\mathbf{m} = \{m_1, \ldots, m_k\}$ be a set in M_d^f, and let $\mathbf{p_m} = \{p_{1,2}, p_{1,3}, \ldots, p_{k,k+1}\}$ be a set in $P_{\mathbf{m}}$. We construct the (unique) *type function* graph $G^f_{\mathbf{m},\mathbf{p_m}}$ associated with $\mathbf{m}, \mathbf{p_m}, f$ and d as follows. For each type $i \in [k]$, we add $(n_i - 2m_i - \sum_{j<i} p_{j,i} - \sum_{j \leq k+1} p_{i,j})$ vertices to $G^f_{\mathbf{m},\mathbf{p_m}}$ corresponding to agents of type i. We also add an additional $d - \sum_{i \in [k]} p_{i,k+1}$ vertices to $G^f_{\mathbf{m},\mathbf{p_m}}$ corresponding to dummy agents. As in the stable marriage setting, the presence of dummy agents allows agents to be unmatched. Note that since we already have $\sum_{i \in [k]} p_{i,k+1}$ unmatched agents, we must add exactly $d - \sum_{i \in [k]} p_{i,k+1}$ vertices to $G^f_{\mathbf{m},\mathbf{p_m}}$ corresponding to dummy agents to ensure that exactly $(n-d)$ agents are unmatched in total. Two vertices (v_a, v_b) are adjacent in $G^f_{\mathbf{m},\mathbf{p_m}}$ if the corresponding pair of agents (a, b) in I are such that $f(\text{type}(a), \text{type}(b)) = 2$ and $\text{type}(a) \neq \text{type}(b)$. A vertex is adjacent to the set of dummy

agents if the corresponding agent has type i and $f(i, k+1) = 2$.

The following observation on the neighbhourhood diversity of a type function graph follows from the fact that agents of the same type have identical preferences, and there are no edges in the type function graph between any pairs of agents of the same type. Note that the bound on the neighbourhood diversity is lower than that in the stable marriage setting since here we do not have dummy agents of different genders.

Observation 4.51. Let I be an instance of TYPED SRTI with at most n agents and at most k types. Let d be a value from the set $\{0, \ldots, n-1\}$ such that $(n-d)$ is even. For each stable type function $f \in \mathcal{F}$, each $\mathbf{m} \in M_d^f$ and each $\mathbf{p_m} \in P_{\mathbf{m}}$, the type function graph $G^f_{\mathbf{m},\mathbf{p_m}}$ has neighbourhood diversity at most $(k+1)$.

The following observation follows from Observation 4.51 and describes a bound the rank of the adjacency matrix associated with each type function graph.

Observation 4.52. Let I be an instance of TYPED SRTI with at most n agents and at most k types. Let f be a stable type function in $\mathcal{F}$, and let $\mathbf{m}$ be a set in M_d^f for some $d \in \{0, \ldots, n-1\}$ such that $(n-d)$ is even. Let $\mathbf{p_m}$ be a set in $P_{\mathbf{m}}$, and let $G^f_{\mathbf{m},\mathbf{p_m}}$ be the type function graph associated with $f, \mathbf{m}$ and $\mathbf{p_m}$. The rank of the adjacency matrix $A = (a_{i,j})$ associated with $G^f_{\mathbf{m},\mathbf{p_m}}$ is at most $(k+1)$.

We will use $|\mathcal{M}_{G^f_{\mathbf{m},\mathbf{p_m}}}|$ to denote the number of perfect matchings in the type function graph $G^f_{\mathbf{m},\mathbf{p_m}}$. The following lemma describes the relationship between the number of perfect matchings in $G^f_{\mathbf{m},\mathbf{p_m}}$, and the number of stable matchings admitted by I with cardinality $(n-d)/2$ which satisfy f and contain m_i pairs of type i, and $p_{i,j}$ pairs with types i and j respectively for each $(i,j) \in [k+1]^2$ such that $i < j$ and $f(i,j) = 1$.

Lemma 4.53. *Let I be an instance of TYPED SRTI, and let f be a stable type function in $\mathcal{F}$. Let d be a value in the set $\{0, \ldots, n-1\}$ such that $(n-d)$ is even, and let $\mathbf{m}$ be a set in M_d^f. Let $\mathbf{p_m}$ be a set in $P_{\mathbf{m}}$, and let $G^f_{\mathbf{m},\mathbf{p_m}}$ be the type function graph constructed from $f, \mathbf{m}$ and $\mathbf{p_m}$. The number of matchings admitted by I with cardinality $(n-d)/2$ which satisfy f and contain m_i pairs of type i for each $i \in [k]$, and $p_{i,j}$ pairs with types i and j respectively for each $(i,j) \in [k+1]^2$ such that $i < j$ and $f(i,j) = 1$ is equal to*

$$\frac{|\mathcal{M}_{G^f_{\mathbf{m},\mathbf{p_m}}}|}{(d - \sum_{i\in[k]} p_{i,k+1})!} \prod_{i\in[k]} \frac{n_i!}{(n_i - 2m_i)!(m_i)!2^{m_i}} \frac{\prod_{i\in[k]} \frac{(n_i-2m_i)!}{(n_i - 2m_i - \sum_{j<i} p_{j,i} - \sum_{j\le k+1} p_{i,j})!}}{2^{\sum_{1\le i<j\le k} p_{i,j}} (\sum_{1\le i<j\le k+1} p_{i,j})!}.$$

Proof. By construction, any perfect matching in $G^f_{\mathbf{m},\mathbf{p_m}}$ corresponds to a matching of the corresponding set of agents in which $d - \sum_{i\in[k]} p_{i,k+1}$ agents are unmatched and no agent is matched to a partner of the same type. Since dummy agents are indistinguishable, the number

of perfect matchings in $G^f_{\mathbf{m},\mathbf{p_m}}$ overcounts the number of matchings of the corresponding set of agents by a factor of $(d - \sum_{i\in[k]} p_{i,k+1})!$. By Lemma 4.47, the number of ways to select $2m_i$ agents of type i agents from I and assign them into same-type pairs for each $i \in [k]$ is equal to

$$\prod_{i\in[k]} \frac{n_i!}{(n_i - 2m_i)!(m_i)!2^{m_i}}.$$

By Lemma 4.49, the number of ways to select $p_{i,j}$ pairs of agents of types i and j respectively from the reduced instance $I_{\mathbf{m}}$ for each $(i, j) \in [k + 1]^2$ such that $i < j$ and $f(i, j) = 1$ is equal to

$$\frac{\prod_{i\in[k]} \frac{(n_i-2m_i)!}{(n_i-2m_i-\sum_{j<i} p_{j,i}-\sum_{j\leq k+1} p_{i,j})!}}{2^{\sum_{1\leq i<j\leq k} p_{i,j}} (\sum_{1\leq i<j\leq k+1} p_{i,j})!}.$$

It follows from the definition of the sets M^f_d and $P_{\mathbf{m}}$, and from the construction of the type function graph $G^f_{\mathbf{m},\mathbf{p_m}}$, that for any perfect matching in $G^f_{\mathbf{m},\mathbf{p_m}}$, the corresponding matching of the set of agents satisfies f . The result follows. □

The following corollary follows from Lemma 4.53, and describes how to obtain the number of matchings satisfying a type function f from the number of perfect matchings in each type function graph constructed from f.

Corollary 4.54. *Let I be an instance of TYPED SRTI with at most n agents and at most k types, and let f be a stable type function in $\mathcal{F}$. We have that*

$$|\mathcal{M}^f| = \sum_{\substack{\{d:d\in\{0,\ldots,n-1\}\\ \text{and } (n-d) \bmod 2\equiv 0\}}} \sum_{\mathbf{m}\in M^f_d} \sum_{\mathbf{p_m}\in P_{\mathbf{m}}} \left(\frac{|\mathcal{M}_{G^f_{\mathbf{m},\mathbf{p_m}}}|}{(d - \sum_{i\in[k]} p_{i,k+1})!} \times \prod_{i\in[k]} \frac{n_i!}{(n_i - 2m_i)!(m_i)!2^{m_i}} \times \frac{\prod_{i\in[k]} \frac{(n_i-2m_i)!}{(n_i-2m_i-\sum_{j<i} p_{j,i}-\sum_{j\leq k+1} p_{i,j})!}}{2^{\sum_{1\leq i<j\leq k} p_{i,j}} (\sum_{1\leq i<j\leq k+1} p_{i,j})!} \right).$$

4.6.4 Reducing #TYPED SRTI to Counting Perfect Matchings

In this section, we describe a reduction from #TYPED SRTI to the problem of counting perfect matchings in a graph parameterised by the neighbourhood diversity of the graph. In Section 4.6.2, we saw that the number of solutions to an instance of TYPED SRTI can be written in terms of the number of matchings satisfying each possible type function. In

Section 4.6.3, we showed that the number of matchings satisfying a particular type function can be obtained from the set of perfect matchings in each graph constructed from the type function. We also showed that the neighbourhood diversity of each type function graph is bounded by a function of the number of agent types. Here, we will see that the problem of counting perfect matchings in a graph belongs to XP parameterised by the neighbourhood diversity of the graph. As a consequence, we show that the problem of counting stable matchings admitted by an instance of TYPED SRTI is also in XP parameterised by the number of agent types needed to describe the instance.

A *perfect matching permutation* [92] on a set of $2n$ elements is a function $\sigma : [2n] \to [2n]$ such that $\sigma(2i-1) < \sigma(2i)$ and $\sigma(2i-1) < \sigma(2i+1)$ for all $1 \leq i < n$. We use PMP$(2n)$ to denote the set of all perfect matching permutations of a set of size $2n$. The *hafnian* [91] of an $2n \times 2n$ symmetric matrix $A = (a_{i,j})$ is defined as

$$\mathrm{haf}(A) = \sum_{\sigma \in PMP(2n)} \prod_{i=1}^{n} A_{\sigma(2i-1)-1,\sigma(2i)-1}.$$

Theorem 4.55 ([92])**.** *Let $G = (V(G), E(G))$ be a graph with $2n$ vertices, and let $A = (a_{i,j})$ be the $2n \times 2n$ (symmetric) adjacency matrix associated with G. The number of perfect matchings in G is equal to* $\mathrm{haf}(A)$.

Theorem 4.56 ([91] (Appendix C))**.** *Let $A = (a_{i,j})$ be an $2n \times 2n$ symmetric matrix with rank at most r. We can compute* $\mathrm{haf}(A)$ *in time $\mathcal{O}((2n + r - 1)^{(r+\mathcal{O}(1))})$.*

Let I be an instance of TYPED SRTI with at most n agents and at most k agent types. Let f be a stable type function in $\mathcal{F}$, and let $d \in \{0, \ldots, n-1\}$ be such that $(n-d)$ is even. Let $\mathbf{m}$ be a set in $\mathbf{m} \in M_d^f$, and let $\mathbf{p_m}$ be a set in $P_{\mathbf{m}}$. Let $G^f_{\mathbf{m},\mathbf{p_m}}$ be the unique type function graph associated with $f, \mathbf{m}$ and $\mathbf{p_m}$. By construction, $G^f_{\mathbf{m},\mathbf{p_m}}$ contains at most $2n$ vertices and the number of vertices in $G^f_{\mathbf{m},\mathbf{p_m}}$ is even. Moreover, the $2n \times 2n$ adjacency matrix $A = (a_{i,j})$ associated with $G^f_{\mathbf{m},\mathbf{p_m}}$ is symmetric and (by Observation 4.52) has rank at most $(k+1)$. We can therefore apply Theorems 4.55 and 4.56 to the problem of counting perfect matchings in $G^f_{\mathbf{m},\mathbf{p_m}}$ as follows.

Lemma 4.57 ([92, 91])**.** *Let I be an instance of TYPED SRTI with at most n agents and at most k types. Let f be a stable type function in $\mathcal{F}$. For any $d \in \{0, \ldots, n-1\}$ such that $(n-d)$ is even, any $\mathbf{m} \in M_d^f$, and any $\mathbf{p_m} \in P_{\mathbf{m}}$, the number of perfect matchings in the type function graph $G^f_{\mathbf{m},\mathbf{p_m}}$ can be counted in time $\mathcal{O}((2n + k)^{(k+\mathcal{O}(1))})$.*

We now use Lemma 4.57 and the relationship described in Corollary 4.54 between the number of matchings satisfying a stable type function and the number of perfect matchings in the corresponding graph to bound the time needed to count the number of matchings satisfying a particular stable type function.

Lemma 4.58. *Let I be an instance of TYPED SRTI with at most n agents and at most k agent types, and let f be a stable type function in $\mathcal{F}$. The value of $|\mathcal{M}^f|$ can be calculated in time $\mathcal{O}(2^{k(k+1)}(2n+k)^{(2k+\mathcal{O}(1))})$.*

Proof. By Corollary 4.54, we have that

$$|\mathcal{M}^f| = \sum_{\substack{\{d:d\in\{0,\ldots,n-1\}\\ \text{and } (n-d) \bmod 2\equiv 0\}}} \sum_{\mathbf{m}\in M_d^f} \sum_{\mathbf{p_m}\in P_{\mathbf{m}}} \left(\frac{|\mathcal{M}_{G^f_{\mathbf{m},\mathbf{p_m}}}|}{(d-\sum_{i\in[k]} p_{i,k+1})!} \times \prod_{i\in[k]} \frac{n_i!}{(n_i-2m_i)!(m_i)!2^{m_i}} \times \frac{\prod_{i\in[k]} \frac{(n_i-2m_i)!}{(n_i-2m_i-\sum_{j<i} p_{j,i}-\sum_{j\le k+1} p_{i,j})!}}{2^{\sum_{1\le i<j\le k} p_{i,j}} (\sum_{1\le i<j\le k+1} p_{i,j})!} \right).$$

We first consider the time needed to compute the value of

$$\frac{|\mathcal{M}_{G^f_{\mathbf{m},\mathbf{p_m}}}|}{(d-\sum_{i\in[k]} p_{i,k+1})!}. \tag{4.58}$$

By Lemma 4.57, computing the value of $|\mathcal{M}_{G^f_{\mathbf{m},\mathbf{p_m}}}|$ takes time

$$\mathcal{O}((2n+k)^{(k+\mathcal{O}(1))}). \tag{4.59}$$

Since there are at most $(2n)!$ perfect matchings in a $2n$ vertex graph, the value of $|\mathcal{M}_{G^f_{\mathbf{m},\mathbf{p_m}}}|$ can be represented using $\mathcal{O}(n\log n)$ bits. To compute the value of

$$(d-\sum_{i\in[k]} p_{i,k+1})! \tag{4.60}$$

we must first compute the sum $\sum_{i\in[k]} p_{i,k+1}$. Since $p_{i,k+1} \in \{0,1\}$ for each $i \in [k]$, computing the sum takes time $\mathcal{O}(k^2)$. Since $d < n$ and $\sum_{i\in[k]} p_{i,k+1} \le k+1$, computing the subtraction in (4.60) takes time $\mathcal{O}(\log n)$. Computing the factorial in (4.60) takes time $\mathcal{O}(n^2)$. Hence, computing the value of (4.60) takes time

$$\begin{aligned} &\mathcal{O}(k^2+\log n+n^2) \\ &= \mathcal{O}(n^2). \end{aligned} \tag{4.61}$$

The value of (4.60) can be represented using $\mathcal{O}(n\log n)$ bits. Thus, it follows from (4.59)

and (4.61) that computing the value of (4.58) takes time

$$\begin{aligned}&\mathcal{O}((2n+k)^{(k+\mathcal{O}(1))}+n^2+(n\log n)^2)\\&\mathcal{O}((2n+k)^{(k+\mathcal{O}(1))}).\end{aligned}\tag{4.62}$$

It follows from the construction of $G^f_{\mathbf{m},\mathbf{p_m}}$ that the value of (4.58) is a positive integer with value at most $(2n)!$. By Lemma 4.48, the value of

$$\prod_{i\in[k]}\frac{n_i!}{(n_i-2m_i)!(m_i)!2^{m_i}}$$

is a positive integer with value at most n^n which can be calculated in time

$$\mathcal{O}(kn^2\log^2 n).\tag{4.63}$$

By Lemma 4.50, the value of

$$\frac{\prod_{i\in[k]}\frac{(n_i-2m_i)!}{(n_i-2m_i-\sum_{j<i}p_{j,i}-\sum_{j\le k+1}p_{i,j})!}}{2^{\sum_{1\le i<j\le k}p_{i,j}}(\sum_{1\le i<j\le k+1}p_{i,j})!}.$$

is a positive integer with value at most n^n which can be calculated in time

$$\mathcal{O}(k^3n^2\log^2 n).\tag{4.64}$$

Thus, it follows from (4.62), (4.63), (4.64) that, for any d, $\mathbf{m}$ and $\mathbf{p_m}$, computing the value of

$$\frac{|\mathcal{M}_{G^f_{\mathbf{m},\mathbf{p_m}}}|}{(d-\sum_{i\in[k]}p_{i,k+1})!}\prod_{i\in[k]}\frac{n_i!}{(n_i-2m_i)!(m_i)!2^{m_i}}\frac{\prod_{i\in[k]}\frac{(n_i-2m_i)!}{(n_i-2m_i-\sum_{j<i}p_{j,i}-\sum_{j\le k+1}p_{i,j})!}}{2^{\sum_{1\le i<j\le k}p_{i,j}}(\sum_{1\le i<j\le k+1}p_{i,j})!}$$

takes time

$$\begin{aligned}&\mathcal{O}((2n+k)^{(k+\mathcal{O}(1))}+kn^2\log^2 n+k^3n^2\log^2 n+n^3\log^3 n)\\&=\mathcal{O}((2n+k)^{(k+\mathcal{O}(1))}).\end{aligned}$$

There are at most $(n-1)$ possible values of d. The number of ways to select k integers whose sum is equal to n is at most $(n+k)^k$. It follows that there are at most $(n+k)^k$ elements in the set M^f_d. Given some $\mathbf{m}\in M^f_d$, the number of different elements in $P_{\mathbf{m}}$ is at most $2^{k(k+1)}$ (each $p_{i,j}$ has value 0 or 1). By definition, the value of $|\mathcal{M}^f|$ is at most n^n. Hence,

computing the value of $|\mathcal{M}^f|$ takes time

$$\begin{aligned}&\mathcal{O}(n \times (n+k)^k \times 2^{k(k+1)} \times ((2n+k)^{(k+\mathcal{O}(1))} + n \log n)) \\ &= \mathcal{O}(2^{k(k+1)}(n+k)^{(2k+\mathcal{O}(1))}).\end{aligned}$$

□

We are now ready to prove our main result.

Theorem 4.59. *Let I be an instance of TYPED SRTI with at most n agents and at most k agent types. We can count the number of stable matchings admitted by I in time $\mathcal{O}(3^{k^2(k+1)^2}(2n+k)^{(2k+\mathcal{O}(1))})$.*

Proof. By Lemma 4.46, given the value of $|\mathcal{M}^f|$ for each $f \in \mathcal{F}$, we can count the number of solutions to I in time

$$\mathcal{O}(3^{k^2(k+1)^2} n \log n).$$

Let f be a stable type function in $\mathcal{F}$. By Lemma 4.58, calculating the value of $|\mathcal{M}^f|$ takes time

$$\mathcal{O}(2^{k(k+1)}(2n+k)^{(2k+\mathcal{O}(1))}).$$

By Lemma 4.38, there are at most $3^{k(k+1)}$ stable type functions in $\mathcal{F}$. By Corollary 4.39, we can obtain the set $\mathcal{F}$ of stable type functions in time $\mathcal{O}(3^{k(k+1)}k^2)$. It follows that we can count the number of solutions to I in time

$$\begin{aligned}&\mathcal{O}(3^{k(k+1)}k^2 + 3^{k(k+1)} \times 2^{k(k+1)}(2n+k)^{(2k+\mathcal{O}(1))} + 3^{k^2(k+1)^2} n \log n) \\ &= \mathcal{O}(3^{k^2(k+1)^2}(2n+k)^{(2k+\mathcal{O}(1))}).\end{aligned}$$

□

4.7 Super-Stability and Strong Stability

In this section, we present results on the complexity of finding and counting super-stable and strongly stable matchings in instances of typed stable matching problems with ties and incomplete preference lists. In Section 4.7.1, we show that the number of super-stable matchings admitted by an instance of any of TYPED SMTI or TYPED SRTI is bounded by a function of the number of agent types needed to describe the instance. In the hospitals/residents

setting, we show that at most one hospital of each type is present in any super-stable matching, and that all residents of the same type must be assigned to the same hospital. In each setting, these observations will allow us to efficiently find and count the number of super-stable matchings. To the best of our knowledge, the problem of counting super-stable matchings has not previously appeared in the literature.

In Section 4.7.2, we extend the algorithm described in Section 4.6 to count strongly stable matchings admitted by an instance of TYPED SRTI. It follows that the problem of counting strongly stable matchings admitted by an instance of TYPED SRTI (and hence TYPED SMTI) belongs to XP parameterised by the number of agent types needed to describe the instance. We believe that ours is the first result on counting strongly stable matchings admitted by an instance of SM or SR.

4.7.1 Finding and Counting Super-Stable Matchings

In this section, we consider the problems of finding and counting super-stable matchings in typed instances of stable matching problems. The first result of this kind bounds the number of super-stable matchings admitted by an instance of TYPED SMTI in terms of the number of agent types.

Lemma 4.60. *Let I be a typed instance of SMTI. If any type contains more than one agent, then every agent of that type is unmatched in any super-stable matching.*

Proof. Let M be a super-stable matching admitted by I. Suppose for a contradiction that some type i contains two agents x and y, and suppose without loss of generality that x is matched in M. By definition, all agents of the opposite gender to x and y are indifferent between x and y. Suppose first that y is unmatched. Since type i finds type$(M(x))$ acceptable, it follows that y would strictly prefer to be matched to $M(x)$ than to be unmatched. In addition, since x and y are of the same type, agents of type type$(M(x))$ are indifferent between x and y. It follows that the pair $(y, M(x))$ forms a blocking pair. Now suppose that both x and y are matched in M, and suppose without loss of generality that type$(M(x)) \succeq_i$ type$(M(y))$. Again, since agents of type type$(M(x))$ are indifferent between x and y, the pair $(y, M(x))$ forms a blocking pair. Thus, M cannot be super-stable. □

The set of all super-stable matchings admitted by an instance of SMTI match exactly the same set of agents [66]. It follows that any super-stable matching admitted by an instance of TYPED SMTI is a maximum cardinality super-stable matching. Making use of the $\mathcal{O}(m)$ algorithm described by Irving and Manlove [66] for finding a super-stable matching (if one exists) in an instance of SRTI, the following lemma describes an $\mathcal{O}(k^2)$ algorithm for finding

a maximum size super-stable matching admitted by an instance of TYPED SMTI or reporting that no such matching exists.

Lemma 4.61. *Let I be an instance of TYPED SMTI with at most k agent types. If I admits a super-stable matching, then a maximum cardinality super-stable matching can be found in time $\mathcal{O}(k^2)$.*

Proof. By Lemma 4.60, if any type contains more than one agent, then every agent of that type is unmatched in any super-stable matching. Hence, if any type i contains more than one agent then we know that no agents of type i are present in the matching, and we may disregard all agents of type i. Since there are at most k types, this can be achieved in time $\mathcal{O}(k)$. Since there are at most k remaining agents, we can now use the algorithm due to Irving and Manlove to find a maximum cardinality super-stable matching in time $\mathcal{O}(k^2)$. □

Using similar logic to the proof of Lemma 4.61, the following result uses the observation made in Lemma 4.60 to bound the time needed to count the number of super-stable matchings admitted by an instance of TYPED SMTI.

Lemma 4.62. *Let I be an instance of TYPED SMTI with at most k agent types. We can count the number of super-stable matchings admitted by I in time $\mathcal{O}(k^{k+3})$.*

Proof. By Lemma 4.60, if any type contains more than one agent, then every agent of that type is unmatched in any super-stable matching. We can check whether any type contains more than one agent (and disregard all agents of that type) in time $\mathcal{O}(k)$. There are at most k remaining agents. It follows that there are at most k^{k+1} ways to match the set of agents (including the possibility that any agent may be unmatched). For each such matching, since there are at most k agents, we can check whether it is super-stable in time $\mathcal{O}(k^2)$. The result follows. □

Note that since all super-stable matchings admitted by an instance of SMTI have equal cardinality, Lemma 4.62 in fact counts the number of maximum cardinality super-stable matchings. The next result describes an analogous result to Lemma 4.60 in the stable roommates setting. Since a matching admitted by an instance of stable roommates may contain pairs of agents with the same type, the presence of two matched agents of the same type in a matching may no longer create a blocking pair as they may be matched to one another. However, if there are more than two matched agents of the same type, then there will always be at least two agents of the same type who are not matched together. We will see that such pairs of agents form a blocking pair. As such, we are able to bound the number of super-stable matchings admitted by an instance of TYPED SRTI in terms of the number of agent types.

Lemma 4.63. *Let I be a typed instance of SRTI. If any type contains more than two agents, then every agent of that type is unmatched in any super-stable matching admitted by I.*

Proof. Let M be a super-stable matching admitted by I, and suppose that some type i contains three agents x, y and z. Suppose without loss of generality that x is matched. Suppose first that at least one of y and z - say y - is unmatched. Since agents of type type$(M(x))$ are indifferent between x and y, and agents of type i would strictly prefer to be matched with type$(M(x))$ than to be unmatched, it follows that the pair $(y, M(x))$ forms a blocking pair. Now suppose that both y and z are matched in M. Suppose without loss of generality that x and y and not matched to each other, and that type$(M(x)) \succeq_i$ type$(M(y))$. Since $(x, y) \notin M$, and $M(x)$ is indifferent between x and y, it follows that the pair $(y, M(x))$ forms a blocking pair. Thus, M cannot be super-stable. □

As in the stable marriage setting, if an instance of SRTI admits a super-stable matching, then the set of all super-stable matchings match exactly the same set of agents [66]. It follows that if we find a super-stable matching admitted by an instance of TYPED SRTI, then we know it is of maximum size. The following bound on the complexity of finding a super-stable matching admitted by an instance of TYPED SRTI (or reporting that none exists) follows from Lemma 4.63 and the $\mathcal{O}(m)$ algorithm described in [66] for finding a super-stable matching admitted by an instance of SRTI or reporting that none exists.

Lemma 4.64. *Let I be a typed instance of SRTI with at most k agent types. If there exists a super-stable matching M admitted by I, then a maximum cardinality super-stable matching can be found in time $\mathcal{O}(k^2)$.*

The proof of the above result follows the same structure as that of Lemma 4.61 and so is not included here. In the following lemma, we use the observation made in Lemma 4.63 to bound the time needed to count the number of super-stable matchings admitted by an instance of TYPED SRTI.

Lemma 4.65. *Let I be an instance of TYPED SRTI with at most k agent types. We can count the number of super-stable matchings admitted by I in time $\mathcal{O}(k^{2k+3})$.*

Proof. By Lemma 4.63, if any type of agent contains more than two agents, then every agent of type i is unmatched in every super-stable matching. Hence, for each type i containing more than two agents, we can disregard all agents of type i from consideration. Since there are at most k types, this can be achieved in time $\mathcal{O}(k)$. Since there are at most $2k$ remaining agents, it follows that there can be at most $(2k)^{2k+1}$ ways to match them together (including the possibility that any agent may be unmatched). We can check whether a matching is super-stable in time $\mathcal{O}(k^2)$. The result follows. □

In the stable marriage and stable roommates settings, a matching contains pairs of distinct agents. In the hospitals/residents setting, a single hospital could be assigned to many residents in a matching. It follows that there could be many matched residents of a single type in a matching without necessarily creating a blocking pair. However, we are able to show that in any super-stable matching admitted by an instance of hospitals/residents, all residents of the same type must be matched to the same hospital. In addition, if there is more than one hospital of the same type, then no hospital of that type can receive any residents in a super-stable matching. These observations will allow us to find and count super-stable matchings in this setting in time depending upon the number of agent types needed to describe an instance and the natural logarithm of the number of agents in the instance.

Lemma 4.66. *Let I be an instance of TYPED HRT, and let M be a super-stable matching admitted by I. If I contains more than one hospital of the same type, then no hospital of that type receives any residents in M.*

Proof. Let h_1 and h_2 be hospitals of type i, and suppose for a contradiction that h_1 is assigned a resident r under M. Suppose first that h_2 does not receive any residents under M. Since $q(h_2) > 0$, it follows that q_2 would strictly prefer to receive r than to receive no residents. Since r is indifferent between h_1 and h_2, the pair (h_2, r) forms a blocking pair.

Now suppose that h_2 receives at least one resident under M, and let $\text{worst}_M(h_2)$ denote the worst type of resident (from the perspective of type i) received by h_2 under M. Suppose without loss of generality that $\text{type}(r) \succeq_i \text{worst}_i(h_2)$. Again, since r is indifferent between h_1 and h_2, the pair (h_2, r) forms a blocking pair. □

Lemma 4.67. *Let I be an instance of TYPED HRT with at most k agent types, and let M be a super-stable matching admitted by I. For each $i \in [k]$, the set of residents of type i are either all unassigned, or are all assigned to the same hospital in M.*

Proof. Let r_1 and r_2 be residents of type i. Suppose first that r_1 is matched to a hospital h and r_2 is unmatched. Since h is indifferent between r_1 and r_2, it follows that the pair (h, r_2) is a blocking pair. Now suppose that r_1 and r_2 are matched to distinct hospitals h_1 and h_2 under M. It follows from Lemma 4.66 that $\text{type}(h_1) \neq \text{type}(h_2)$. Suppose without loss of generality that $\text{type}(h_1) \succeq_i \text{type}(h_2)$. Since h_1 is indifferent between r_1 and r_2, the pair (h_1, r_2) forms a blocking pair. □

It follows from the above results that any super-stable matching admitted by an instance of TYPED HRT is formed by pairing together resident types and hospital types. In addition, it follows from the super-stability variant of the Rural Hospitals Theorem [64] that any super-stable matching admitted by an instance of TYPED HRT matches exactly the same set of agents. Hence, we have the following result.

Lemma 4.68. *Let I be an instance of TYPED HRT with at most n agents and at most k agent types. We can find a maximum cardinality super-stable matching admitted by I or report that no such matching exists in time $\mathcal{O}(k^{k+3} \log n)$.*

Proof. By Lemma 4.66, in any super-stable matching M of the agents in I, either all hospitals of the same type are unmatched, or there is only one hospital of that type in I. We can check whether any type of hospital contains more than one hospital in time $\mathcal{O}(k)$. If any type of hospital type does contain more than one hospital, then we may disregard all hospitals of that type. By Lemma 4.67, the set of agents of each type are either all unassigned or are all assigned to a single hospital. It follows that a possible super-stable matching is formed by assigning each type of resident to a single hospital type.

For each such matching M, we must first check whether any hospital is oversubscribed. Since all residents of the same type must be assigned to the same hospital, this can be achieved by comparing the quota of the hospital to the sum, over each type i assigned to the hospital, of the total number of residents of type i. Since there are at most n residents and at most k types, we can check whether any hospital is oversubscribed under M in time $\mathcal{O}(k^2 \log n)$.

It remains to check whether any matching is super-stable. To determine whether a matching M is super-stable, we must first determine the worst type of resident assigned to each hospital under M, and also whether each hospital has spare capacity. To determine whether a hospital h has spare capacity, it suffices to compare the capacity of h to the sum of the number of agents of each type i assigned to h. This takes time $\mathcal{O}(k^2 \log n)$ for the set of all hospitals. We can determine the worst type of resident assigned to each hospital under M in time $\mathcal{O}(k^2)$. We can now determine whether M is super-stable by comparing pairs of types of hospitals and residents. Since there are at most k types, this can be achieved in time $\mathcal{O}(k^2)$.

It follows from the above that we can determine whether a single matching M is super-stable in time $\mathcal{O}(k^2 \log n)$. Since each type of resident can be assigned to a single hospital, there are at most k^{k+1} possible pairings of types of hospitals to types of residents. It follows that we can find a super-stable matching admitted by I or report that none exist in time $\mathcal{O}(k^{k+3} \log n)$. □

Using an almost identical argument, we obtain an $\mathcal{O}(k^{k+3} \log n)$ time algorithm for counting super-stable matchings admitted by an instance of TYPED HRT.

Lemma 4.69. *Let I be an instance of TYPED HRT with at most n agents and at most k agent types. We can count the number of super-stable matchings admitted by I in time $\mathcal{O}(k^{k+3} \log n)$.*

4.7.2 Counting Strongly Stable Matchings

In this section, we consider the problem of counting strongly stable matchings admitted by an instance of TYPED SRTI. Under strong stability, indifference between two unmatched agents no longer guarantees the presence of a blocking pair in a matching. As such, we cannot obtain the same structural results given in Section 4.7.1 in this setting. Instead, we extend the algorithm described in Section 4.6 for counting weakly stable matchings to the problem of counting strongly stable matchings.

Under weak stability, we showed that a result of Meeks and Rastegari [11] allows us to determine the stability of a matching by looking at only the pairs of types present in the matching. Here, we use similar logic to prove an analogous result under strong stability. Using this observation we then show that, as under weak stability, the set of strongly stable matchings admitted by an instance of TYPED SRTI can be defined in terms of (strongly) stable type functions. We then show that some analogous results to those given in Section 4.6 allow us to compute the number of strongly stable matchings admitted by an instance of TYPED SRTI with at most n agents and at most k agent types in time $\mathcal{O}(3^{k^2(k+1)^2}(2n + k)^{(2k+\mathcal{O}(1))})$.

Let I be an instance of TYPED SRTI and let M be a matching of the set of agents in I. Recall that we use $\text{worst}_M(i)$ and $\text{second\_worst}_M(i)$ to denote the types of the least desirable and second least desirable agents (from the perspective of type i) matched to any agent of type i under M. If there is only one agent of type i, then we set $\text{second\_worst}_M(i) = \varnothing$. The following is an analogous result to Lemma 4.36 from Section 4.6.

Lemma 4.70. *Let I be an instance of TYPED SRTI with at most k agent types. A matching M admitted by I is strongly stable if and only if*

- *there is no pair of types $(i, j) \in [k]^{(2)}$, $i \neq j$, such that $j \succ_i \text{worst}_M(i)$ and $i \succeq_j \text{worst}_M(j)$, and*
- *there is no type $i \in [k]$ such that $i \succeq_i \text{second\_worst}_M(i)$.*

Proof. Suppose first that M is not strongly stable. Then there exists a pair of agents (a, b) with types (i, j) such that a and b are not matched together but type i strictly prefers type j to type$(M(a))$ and type j either strictly prefers type i to type$(M(b))$ or is indifferent between the two types. First suppose that $i \neq j$. Then we have that $j \succ_i \text{type}(M(a)) \succeq_i \text{worst}_M(i)$ and similarly $i \succeq_j \text{type}(M(b)) \succeq_j \text{worst}_M(j)$. Now suppose that $i = j$. We have that either $\text{type}(M(a)) \succeq_i \text{worst}_M(i)$ and $\text{type}(M(b)) \succeq_i \text{second\_worst}_M(i)$, or else $\text{type}(M(b)) \succeq_i \text{worst}_M(i)$ and $\text{type}(M(a)) \succeq_i \text{second\_worst}_M(i)$. In either case we have that $i \succeq_i \text{second\_worst}_M(i)$.

Now suppose that M is strongly stable, and suppose that at least one of the two conditions in the lemma statement does not hold. Suppose first that there is some pair of types (i, j)

with $i \neq j$ such that $j \succ_i \text{worst}_M(i)$ and $i \succeq_j \text{worst}_M(j)$. Then there is some agent a of type i who is matched with an agent of type $\text{worst}_M(i)$, and some agent b of type j who is matched with an agent of type $\text{worst}_M(j)$. Since type i strictly prefers type j to $\text{worst}_M(j)$, and type j either strictly prefers type i to $\text{worst}_M(j)$ or is indifferent between them, it follows that (a, b) form a blocking pair, a contradiction. Now suppose that the second condition does not hold. Then there exist agents a and b, both with type i, who are matched to agents of types $\text{worst}_M(i)$ and $\text{second\_worst}_M(i)$ respectively. Thus a strictly prefers b to $M(a)$ and b either strictly prefers a to $M(b)$ or else is indifferent between them. It follows that (a, b) is a blocking pair, and so M is not strongly stable. □

As under weak stability, it follows that to determine whether a matching is strongly stable, we only need to know which pairs of types of agents are present in the matching, and whether each pair of types is realised by more than one pair of agents. Let I be an instance of TYPED SRTI. As in Section 4.6, we define a *type function* as a function $f : [k+1]^2 \to \{0, 1, 2\}$ where $f(k+1, k+1) = 0$ and, for each $(t_1, t_2) \in [k+1]^2$, we have that $f(t_1 t_2) = f(t_2 t_1)$ and

- $f(t_1, t_2) = 0$ if there may be no pairs of agents with types t_1 and t_2 respectively, and
- $f(t_1, t_2) = 1$ if there may be at most one pair of agents with types t_1 and t_2 respectively, and
- $f(t_1, t_2) = 2$ if there may be any number of pairs of agents with types t_1 and t_2 respectively.

For each type $i \in [k]$, we define $\text{worst}_f(i)$ as least desirable type of agent from the perspective of type i such that $f(i, \text{worst}_f(i)) \neq 0$. If $f(i, \text{worst}_f(i)) = 1$, then let $\text{second\_worst}_f(i)$ be the second least desirable type (from the perspective of type i) such that $f(i, \text{second\_worst}_f(i)) \neq 0$. Otherwise, set $\text{second\_worst}_f(i) = \text{worst}_f(i)$. We call f a *strongly stable type function* if there are no pairs $(i, j) \in [k]^2$ with $i \neq j$ such that $j \succ_i \text{worst}_f(i)$ and $i \succeq_j \text{worst}_f(j)$, and there is no type $i \in [k]$ such that $\text{second\_worst}_f(i) \neq \varnothing$ and $i \succeq_i \text{second\_worst}_f(i)$. We will require the following analogous observation to Observation 4.37 from Section 4.6.

Observation 4.71. Let I be an instance of TYPED SRTI with at most k agent types. We can determine whether a type function is strongly stable in time $\mathcal{O}(k^2)$.

The following lemma is an analogous result of Lemma 4.38 from Section 4.6. The proof of each result is almost identical, so we do not include a proof here.

Lemma 4.72. *Let I be an instance of TYPED SRTI with at most k agent types. There are at most $3^{k(k+1)}$ type functions over the set of types in I.*

Let $\mathcal{F}$ denote the set of strongly stable type functions in an instance of TYPED SRTI. The following corollary follows from Observation 4.71 and Lemma 4.72.

Corollary 4.73. *Let I be an instance of TYPED SRTI with at most k agent types. The set $\mathcal{T}$ of strongly stable type functions can be obtained in time $\mathcal{O}(3^{k(k+1)}k^2)$.*

Let I be an instance of TYPED SRTI. As in Section 4.6, we use $\mathcal{M}^f$ to denote the set of matchings admitted by I which satisfy a type function f. We use $|\mathcal{M}^f|$ to denote the number of matchings which satisfy f. In Section 4.6, we showed that if we know the value of $|\mathcal{M}^f|$ for each $f \in \mathcal{F}$, then it is straightforward to compute the number of solutions to I. Here, we will require the following analogous result. We note that the proof of Lemma 4.74 is almost identical to that of Lemma 4.46 from Section 4.6.

Lemma 4.74. *Let I be an instance of TYPED SRTI with at most n agents and at most k agent types. Given $\mathcal{F}$ and the value of $|\mathcal{M}^f|$ for each $f \in \mathcal{F}$, we can count the number of solutions to I in time $\mathcal{O}(3^{k^2(k+1)^2}n \log n)$.*

In Section 4.6, we saw that the problem of computing the value of $|\mathcal{M}^f|$ for each $f \in \mathcal{F}$ can be reduced to the problem of counting perfect matchings in a general graph parameterised by the neighbourhood diversity of the graph. We apply the same reduction here. The proof of the following lemma is almost identical to that of Lemma 4.58.

Lemma 4.75. *Let I be an instance of TYPED SRTI with at most n agents and at most k agent types, and let f be a strongly stable type function in $\mathcal{F}$. The value of $|\mathcal{M}^f|$ can be calculated in time $\mathcal{O}(2^{k(k+1)}(2n+k)^{(2k+\mathcal{O}(1))})$.*

It follows from the above results that, by applying the same logic as that used in the proof of Theorem 4.59, we can obtain the following result.

Theorem 4.76. *Let I be an instance of TYPED SRTI with at most n agents and at most k agent types. We can count the number of strongly stable matchings admitted by I in time $\mathcal{O}(3^{k^2(k+1)^2}(2n+k)^{(2k+\mathcal{O}(1))})$.*

4.8 Remarks and Open Problems

In this chapter, we studied stable matching problems with ties and incomplete preference lists in the setting where the set of agents can be partitioned into a small number of types. In each case, we saw that restricting the number of allowable agent types in an instance results in improved algorithmic efficiency.

In Section 4.5, we described an XP algorithm for the problem of counting stable matchings admitted by an instance of TYPED SMTI parameterised by the number of agent types. This was achieved by reducing our problem to that of counting perfect matchings in a bipartite

graph with bounded neighbourhood diversity. Since the problem of counting perfect matchings in a bipartite graph is equivalent to computing the permanent of its adjacency matrix, our result follows from the existence of an XP algorithm due to Barvinok for computing the permanent of a matrix parameterised by its rank. By performing a thorough analysis of the complexity of Barvinok's algorithm, we were able to describe a precise upper bound on the runtime of our algorithm.

It remains an open question whether we can obtain an FPT algorithm for #TYPED SMTI parameterised by the number of agent types, or whether the problem is #W[1]-hard. In Chapter 5 (Section 5.6), we will describe an FPT approximate counting algorithm for #TYPED SMTI parameterised by the number of agent types. It follows that we can certainly approximate the number of stable matchings efficiently in instances of TYPED SMTI with a small number of agent types.

In the decision setting, a standard "cloning technique" [55] can be used to transform an instance of HRT into an instance of SMTI (each hospital h is cloned $q(h)$ times), so that an algorithm for finding a stable matching admitted by an instance of SMTI can be used to find a stable matching admitted by an instance of HRT. We believe that the same technique can be used to apply our algorithm for #TYPED SMTI to the problem of counting solutions to an instance of TYPED HRT. To avoid over-counting (since each hospital may appear more than once in the cloned instance), we require an additional step each time that we count perfect matchings in a graph formed from a stable type set. Since each hospital h appears $q(h)$ times, we believe that dividing the current value by $q(h)!$ for each hospital h should prevent over-counting.

In Section 4.6, we showed that #TYPED SRTI belongs to XP parameterised by the number of agent types. The algorithm for #TYPED SRTI follows a broadly similar approach to the algorithm for #TYPED SMTI described in Section 4.5. In this setting, since we did not perform such a thorough analysis of the complexity of computing the hafnian of a matrix (as we did for computing the permanent in the stable marriage setting), our runtime contains an unknown constant in the exponent. As in the stable marriage setting, it is of interest to determine whether there exists an FPT algorithm for #TYPED SRTI parameterised by the number of agent types.

In Section 4.7, we described new algorithmic results for finding and counting strongly stable and super-stable matchings in instances of stable matching problems with ties and incomplete preference lists. Under super-stability, we saw that structural restrictions on the set of super-stable matchings allow us to efficiently find and count super-stable matchings in instances of TYPED SMTI, TYPED HRT and TYPED SRTI. Under strong stability, we showed that the algorithm from Section 4.6 can be extended to count strongly stable matchings. It follows that the problem of counting strongly stable matchings admitted by an in-

stance of TYPED SRTI (and hence also TYPED SMTI) is in XP parameterised by the number of agent types. We suspect that the cloning technique described above can also be used to extend this algorithm to the TYPED HRT setting. It would also be interesting to investigate whether we can obtain similar structural results under strong stability as those described for super-stability.

Chapter 5

Approximately Counting Stable Matchings

5.1 Motivation

In Chapter 4, we described an algorithm for counting stable matchings in a typed instance of stable marriage with ties and incomplete preference lists. We saw that if the number of agent types needed to describe an instance is treated as a constant, then we can count the number of matchings in time depending polynomially upon the number of agents. It remains an open problem whether we can achieve tractability while allowing the number of agent types to be included as part of the input. In this chapter, we show that we can compute an arbitrarily-close approximation to the number of stable matchings admitted by an instance of stable marriage in time depending polynomially upon the number of agents and the desired error while allowing the number of agent types to be included as part of the input. As an intermediary result, we provide a generalisation of a parameterised approximation scheme due to Arvind and Raman [21] for computing the cardinality of a union of sets.

We also consider a generalisation of typed stable marriage in which individual agents may declare a constant number of their available partners as unacceptable. We conjecture that there exists an XP algorithm for approximately counting stable matchings in this setting parameterised by the number of agent types, and provide evidence to suggest that an FPT algorithm for exact counting is unlikely to exist.

5.2 Notation and Definitions

We carry over the notation and definitions for typed stable matching problems described in Section 4.2 of Chapter 4. A generalisation of typed stable matching problems defined in [11]

allows individual agents to rank a small number of their available partners without regard to their type. We say that an agent considers such candidates to be *exceptional*. In practice, this may occur if agents have additional information about a subset of their available partners. We say that an agent a has *deleted* an agent b from their preference list if type(a) considers type(b) acceptable, but b does not appear in the preference list of agent a i.e. agent a finds agent b unacceptable. We note that the notion of deletions in type stable matching problems is similar to the notion of master lists [80] in the standard stable matching setting without types.

5.3 Literature Review

We surveyed the literature on stable matching in Section 4.3 of Chapter 4. In this section, we describe results relating to the union of sets problem. The first result of this kind (due to Karp and Luby [93]) describes an FPRAS for the union of sets problem given polynomial-time algorithms for computing the size of each set, sampling from a single set, and deciding membership of a set. The second result is a parameterised version of the Karp-Luby result provided by Arvind and Raman [21]. We will also describe some examples of approximation algorithms for other computational problems which rely on either of these results.

Let m and n be positive integers, and let $\{D_1, \ldots, D_m\}$ be a collection of m sets whose elements are binary strings of length $n^{\mathcal{O}(1)}$. In [93], Karp and Luby describe an FPRAS for approximating the number of elements in the union $\bigcup_{i\in[m]} D_i$ if each of the following conditions are met for every $i \in [m]$:

1. the value of $|D_i|$ can be computed in time $f_1(n)$ for some polynomial function f_1, and
2. there exists an algorithm for sampling elements uniformly at random from D_i in time $f_2(n)$ for some polynomial function f_2, and
3. for any $s \in \bigcup_{i\in[m]} D_i$, it is possible to determine whether $s \in D_i$ in time $f_3(n)$ for some polynomial function f_3.

We now describe a selection of results in the literature which rely on the existence of the Karp-Luby algorithm. Let $F = C_1 \vee \ldots \vee C_m$ be a logical formula in disjunctive normal form (DNF) [94]. Let D_i denote the set containing the truth assignments of the variables in C_i which satisfy C_i for each $i \in [m]$. Observe that the cardinality of $\bigcup_{i=1}^{m} D_i$ is equal to the number of solutions to F. It follows that there exists an FPRAS for the problem of counting solutions to a DNF formula [93] subject to the above conditions on the sets $D_1, \ldots, D_m$. The *all-terminal network reliability problem* asks for the probability of a graph $G = (V(G), E(G))$ becoming disconnected due to edge failure, where each edge fails independently with some given probability. In [95], the Karp-Luby result is used to obtain

an FPRAS for computing an estimate for the overall failure probability. The *3-dimensional matching problem* is a generalisation of the problem of finding a matching in a bipartite graph in which there are three sets X, Y and Z of vertices, and an edge contains exactly one vertex from each set. As in the 2D setting, the size of the matching is equal to the number of edges contained in the matching. In [96], Liu et al. use the Karp-Luby result to obtain an FPTRAS for the problem of counting 3-dimensional matchings parameterised by the size of the matching.

In [21], Arvind and Raman described a parameterised version of the Karp-Luby result. Let n and k be positive integers with $0 \leq k \leq n$. Let $\{D_1, \ldots, D_m\}$ be a collection of m sets whose elements are binary strings of length $n^{\mathcal{O}(1)}$ for some $m = h(k)n^{\mathcal{O}(1)}$ where h is any computable function. There exists an FPTRAS for approximating the size of the union $\bigcup_{i=1}^{m} D_i$ if, for each $i \in [m]$, each of the following conditions are met:

1. the value of $|D_i|$ can be computed in time $g_1(k)f_1(n)$ for some computable function g_1 and a polynomial function f_1, and
2. there exists an algorithm for sampling elements uniformly at random from D_i in time $g_2(k)f_2(n)$ for some computable function g_2 and a polynomial function f_2, and
3. for any $s \in \bigcup_{i\in[m]} D_i$, it is possible to determine whether $s \in D_i$ in time $g_3(k)f_3(n)$ for some computable function g_3 and a polynomial function f_3.

In the same paper, the authors showed that their result can be used to obtain an FPTRAS for counting copies of a subgraph with bounded treewidth in a graph parameterised by the order of the subgraph. In [97], Jerrum and Meeks use the Arvind-Raman result to obtain an FPTRAS for counting sets of k vertices in a graph which induce a connected subgraph.

Let n be a positive integer, and let $\mathcal{S} = \{S_1, \ldots, S_n\}$ be a collection of n sets containing elements from some universe U. A *k-packing* P in $\mathcal{S}$ is a collection of k sets from $\mathcal{S}$ such that no two sets in P have common elements. The authors of [98] use the Arvind-Raman result to obtain an FPTRAS for the problem of counting k-packings in $\mathcal{S}$ parameterised by k. The subgraph packing problem is a related problem on graphs. Let $G = (V(G), E(G))$ be a graph, and let $H = (V(H), E(H))$ be a connected graph. A *subgraph packing* of G based on H is a set of k vertex-disjoint copies of H in G. A similar application [98] of the Arvind-Raman result yields an FPTRAS for counting subgraph packings parameterised by k.

5.4 Contributions

In Section 5.5, we describe an FPTRAS for the union of sets problem. Our algorithm is a generalisation of the FPTRAS described by Arvind and Raman [21] for the same problem.

The FPTRAS described by Arvind and Raman relies on the existence of FPT subroutines for counting and sampling from each set exactly. Our result instead relies on the existence of more general FPT subroutines for approximately counting and sampling from the sets.

Section 5.6 uses our FPTRAS for union of sets to obtain an FPTRAS for the problem of counting stable matchings admitted by an instance of TYPED SMTI. Recall from Section 4.5 of Chapter 4 that the stability of a matching admitted by an instance of TYPED SMTI can be determined by looking at only the pairs of types - called a type set - present in the matching. As a result, we showed that the set of stable matchings admitted by an instance of TYPED SMTI is equal to the union, over all stable type sets, of the number of matchings satisfying each type set. We saw that the problem of counting the number of matchings satisfying a type set is equivalent to counting perfect matchings in the associated acceptability graph. It follows from existing results in the literature that we can sample from and approximate the cardinality of the set of perfect matchings in each such graph efficiently. The problem of deciding whether a matching satisfies a type set is in FPT parameterised by the number of agent types. It then follows directly from the main result of Section 5.5 that there exists an FPTRAS for counting solutions to TYPED SMTI parameterised by the number of agent types.

Finally, in Section 5.7, we consider the problem of counting stable matchings in instances of TYPED SMTI in which agents may individually declare some of their available partners as unacceptable. We conjecture that if the number of partners which individual agents may delete from their preference lists is a constant, then we can efficiently approximate the number of stable matchings in this setting. Using a reduction from the problem of finding a clique containing a vertex of each colour in a graph whose vertices are assigned colours, we show that the problem of finding a maximum size stable matching admitted by an instance of TYPED SMTI in which agents may delete up to 2 agents from their preference list is W[1]-hard parameterised by the number of agent types. As a consequence, we argue that an FPT algorithm for exactly counting stable matchings in this setting is unlikely to exist.

5.5 An FPTRAS for Union of Sets

In this section, we describe a generalisation of the FPTRAS given by Arvind and Raman [21] for the following problem.

UNION OF SETS	
Input:	Positive integers n, k and m such that $0 \leq k \leq n$ and $m = nh(k)$ for some computable function h, and m sets $D_{n,k} = \{D_1, \ldots, D_m\}$ whose elements are binary string of length $n^{\mathcal{O}(1)}$.
Parameter:	k.
Question:	What is the size of the set $\bigcup_{i=1}^{m} D_i$?

In what follows, we describe the existence of an FPTRAS for the union of sets problem subject to the following more general set of conditions than those described by Arvind and Raman [21]:

- there exists an FPTRAS with parameter k for approximating the size of $|D_i|$ for each $i \in [m]$, and
- there exists an FPTAUS with parameter k for sampling almost uniformly at random from D_i for each $i \in [m]$, and
- for any element $s \in \bigcup_{i \in [m]} D_i$, there exists an FPT algorithm parameterised by k for deciding whether $s \in D_i$.

In Section 5.6, we will use this result to obtain an FPTRAS for approximating the number of stable matchings in an instance of TYPED SMTI. Our algorithm follows the same general approach used in the FPTRAS given by Arvind and Raman [21], and the FPRAS due to Karp and Luby [93]. Both algorithms are based on the following Monte-Carlo technique of repeated random sampling. Let S be a set of known size, and let U be the subset of S whose value we would like to approximate. Let f be the function defined on the elements of S such that $f(s) = 1$ if $s \in U$, and $f(s) = 0$ otherwise. For sufficiently large N, we complete N trials, where a single trial involves choosing an element s uniformly at random from S and computing the value of $f(s)$. The output of the algorithm is given by $|S| \times \sum_{j \in [N]} X_j / N$, where X_j denotes the outcome of the jth trial. Note that in our setting, we do not have an exact value of $|S|$ - this value too must be approximated. Moreover, we do not have access to an algorithm for selecting elements from S uniformly at random - instead, we rely on being able to sample from S according to an almost uniform distribution.

We begin by defining the sets S and U for the union of sets problem. Let n, k and m be positive integers such that $0 \leq k \leq n$ and $m = nh(k)$ for some computable function h. Let $\{D_1, \ldots, D_m\}$ be a collection of m sets containing binary strings of length $n^{\mathcal{O}(1)}$. We

use S to denote the multiset containing every element $s \in \bigcup_{i=1}^{m} D_i$ once for each set in $\{D_1, \ldots, D_m\}$ containing s. We define an element of S as a pair (s, i) where $1 \leq i \leq m$ and $s \in D_i$. We then define a function $\rho : S \to \{0, 1\}$ on the elements of S such that $\rho(s, i) = 1$ if i is the smallest index such that $s \in D_i$, and $\rho(s, i) = 0$ otherwise. Let U be the subset of S where f has value 1, and let $|U|$ denote the size of U. Since, for each element $s \in \bigcup_{i=1}^{m} D_i$, we have that $\rho(s, i) = 1$ for exactly one value of i, it follows that $|U| = |\bigcup_{i=1}^{m} D_i|$. As one of the inputs to our algorithm, we will require a lower bound μ on the value of $|U|/|S|$. The following lemma provides a rudimentary lower bound on this ratio in case a better bound is not available.

Lemma 5.1. *Let n, k and m be positive integers such that $0 \leq k \leq n$ and $m = nh(k)$ for some computable function h. Let $\{D_1, \ldots, D_m\}$ be a collection of m sets containing binary string of length $n^{\mathcal{O}(1)}$. Let S denote the multiset containing every element s in $\bigcup_{i=1}^{m} D_i$ once for each set in $\{D_1, \ldots, D_m\}$ containing s. Let f be a function on the elements of S such that $\rho(s, i) = 1$ if i is the smallest index such that $s \in D_i$, and $\rho(s, i) = 0$ otherwise. Let U denote the subset of S on which f has value 1. We have that*

$$\frac{1}{m} \leq \frac{|U|}{|S|} \leq 1.$$

Proof. Observe that each element s in U appears at least once as an element of S. In the case where each element in S is a member of exactly one set in $\{D_1, \ldots, D_m\}$, we have $|U|/|S| = 1$. Conversely, each element in U may appear once in every set in $\{D_1, \ldots, D_m\}$. In this case, since there are m sets, we have that $|U|/|S| = 1/m$. □

Suppose that, for each $i \in [m]$, there exists an FPTRAS for approximating the size of D_i. We shall denote the output of such an algorithm with inputs D_i, ϵ' and δ' by $approx_{(\epsilon', \delta')}(D_i)$. Suppose also that there exists an FPTAUS for sampling almost uniformly at random from D_i for each $i \in [m]$. We denote the output of a single call to the FPTAUS with inputs D_i and ϵ' by $approx\_sample_{(\epsilon')}(D_i)$. Given the sets $\{D_1, \ldots, D_m\}$, a lower bound μ on $|U|/|S|$, and real numbers ϵ and δ such that $0 < \delta, \epsilon < 1$ as input, we claim that the following algorithm returns an ϵ-approximation of $|\bigcup_{i=1}^{m} D_i|$ with probability at least $(1 - \delta)$.

In the remainder of this section, we prove that Algorithm 3 describes an FPTRAS for the union of sets problem. We begin by bounding the value of the estimate of $|S|$. We then bound the probability that a particular element $(s, i) \in S$ is selected in a single "trial". Using these bounds, we are able to derive a bound on the probability that the output of the algorithm exceeds the allowable bounds of $(1 \pm \epsilon)|U|$. This allows us to bound the probability that our algorithm fails - we show that this occurs with probability at most δ. It then remains to show that the runtime of our algorithm can be written in the form $g(k)f(n, 1/\epsilon, \log(1/\delta))$ where f is a polynomial function and g is any computable function.

Algorithm 3: An FPTRAS for Union of Sets

input : sets $\{D_1, \ldots, D_m\}$, real numbers ϵ and δ with $0 < \epsilon, \delta < 1$, and a lower bound μ on $|U|/|S|$

output: an ϵ-approximation of $|\bigcup_{i=1}^{m} D_i|$

1 Set $\epsilon' = \epsilon/10$;
2 Set $\delta' = \delta/(3m)$;
3 **for** $1 \leq i \leq m$ **do**
4 | Let $|D_i|_{(\epsilon',\delta')} = approx_{(\epsilon',\delta')}(D_i)$;
5 **end**
6 Set $|S|_{(\epsilon',\delta')} = \sum_{i\in[m]} |D_i|_{(\epsilon',\delta')}$;
7 **for** $1 \leq i \leq m$ **do**
8 | Set $p_i = \frac{|D_i|_{(\epsilon',\delta')}}{\sum_{j\in[m]} |D_j|_{(\epsilon',\delta')}}$;
9 **end**
10 Set $N = \left\lceil \frac{39\ln(3/\delta)}{\epsilon^2\mu} \right\rceil$;
11 **for** $1 \leq j \leq N$ **do**
12 | Randomly select i from $[m]$ with probability p_i;
13 | Set $s = approx\_sample_{(\epsilon')}(D_i)$;
14 | **if** $\rho(s, i) = 1$ **then**
15 | | Set $X_j = 1$;
16 | **else**
17 | | Set $X_j = 0$;
18 **end**
19 Set $X = \sum_{j\in[N]} X_j$;
20 **return** $Y = |S|_{(\epsilon',\delta')} \times X/N$;

The following lemma bounds the value of the estimate $|S|_{(\epsilon',\delta')}$ of $|S|$ made at Line 6 of the algorithm. Note that if any call to $approx_{(\epsilon',\delta')}(D_i)$ fails at Line 3 then our algorithm also fails, so we shall assume otherwise at this stage.

Lemma 5.2. *Let $\{D_1, \ldots, D_m\}, \epsilon, \delta$ and μ be the input to Algorithm 3. Assuming that no call to $approx_{(\epsilon',\delta')}(D_i)$ fails, the value of $|S|_{(\epsilon',\delta')}$ set at Line 6 is such that*

$$(1 - \epsilon')|S| \leq |S|_{(\epsilon',\delta')} \leq (1 + \epsilon')|S|.$$

Proof. By definition, for each $i \in [m]$, as long as the call to $approx_{(\epsilon',\delta')}(D_i)$ does not fail, we have that

$$(1 - \epsilon')|D_i| \leq |D_i|_{(\epsilon',\delta')} \leq (1 + \epsilon')|D_i|.$$

Since we set $|S|_{(\epsilon',\delta')} = \sum_{i\in[m]} |D_i|_{(\epsilon',\delta')}$ at Line 6, the result follows. □

In the following lemma, we bound the probability $p_{(s,i)}$ of an element $(s, i) \in S$ being

selected in a single iteration of the loop at Line 11.

Lemma 5.3. *Let $\{D_1, \ldots, D_m\}, \epsilon, \delta$ and μ be the input to Algorithm 3. Assuming that no call to $approx_{(\epsilon',\delta')}(D_i)$ fails, the probability $p_{(s,i)}$ of choosing a particular element $(s,i) \in S$ in a single iteration of the loop at Line 11 lies in the range*

$$\frac{(1-\epsilon')^2}{(1+\epsilon')\sum_{j\in[m]}|D_j|} \leq p_{(s,i)} \leq \frac{(1+\epsilon')^2}{(1-\epsilon')\sum_{j\in[m]}|D_j|}.$$

Proof. In what follows we derive an upper bound on the value of $p_{(s,i)}$. The lower bound on $p_{(s,i)}$ can be derived using the same logic. Let $p_{s|i}$ denote the conditional probability of choosing s from D_i given that i has been chosen from $[m]$. We have that

$$p_{(s,i)} = p_{s|i} \times p_i.$$

It follows that to obtain an upper bound on $p_{(s,i)}$, it suffices to derive upper bounds on each of $p_{s|i}$ and p_i. At Line 11 of Algorithm 3, we set

$$p_i = \frac{|D_i|_{(\epsilon',\delta')}}{\sum_{j\in[m]}|D_j|_{(\epsilon',\delta')}}.$$

It follows that the value of p_i is greatest when the value of $|D_i|_{(\epsilon',\delta')}$ overestimates the value of $|D_i|$ as much as possible, and the value of $|D_j|_{(\epsilon',\delta')}$ underestimates the value of $|D_j|$ as much as possible for each $j \neq i$. Since we have assumed that the call to $approx_{(\epsilon',\delta')}(D_i)$ does not fail, we may overestimate $|D_i|$ by a factor of at most $(1+\epsilon')$. Similarly, we may underestimate each $|D_j|$ by a factor of at most $(1-\epsilon')$. Thus, we have that

$$p_i \leq \frac{(1+\epsilon')|D_i|}{(1-\epsilon')\sum_{j\in[m]}|D_j|}. \tag{5.1}$$

The probability $p_{s|i}$ of choosing s from D_i given that we have chosen i is equal to the probability that the call to $approx\_sample_{(\epsilon')}(D_i)$ selects the element s from D_i. Thus, by definition we have that

$$p_{s|i} \leq \frac{(1+\epsilon')}{|D_i|}. \tag{5.2}$$

It follows from (5.1) and (5.2) that

$$p_{(s,i)} \leq \frac{(1+\epsilon')^2}{(1-\epsilon')\sum_{j\in[m]}|D_j|}.$$

□

In the following lemma, we use the bounds from Lemma 5.3 on the probability of picking a specific element from the set S to bound the probability of picking any element (s, i) from S with $\rho(s, i) = 1$.

Lemma 5.4. *Let $\{D_1, \ldots, D_m\}, \epsilon, \delta$ and μ be the input to Algorithm 3. Assuming that no call to $approx_{(\epsilon',\delta')}(D_i)$ fails, at any single iteration of the loop at Line 11, the probability p of choosing an element (s, i) from S such that $\rho(s, i) = 1$ lies in the range*

$$\frac{(1-\epsilon')^2}{(1+\epsilon')}\frac{|U|}{|S|} \leq p \leq \frac{(1+\epsilon')^2}{(1-\epsilon')}\frac{|U|}{|S|}.$$

Proof. By Lemma 5.3, the probability $p_{(s,i)}$ of picking the pair (s, i) from S lies in the range

$$\frac{(1-\epsilon')^2}{(1+\epsilon')\sum_{j\in[m]}|D_j|} \leq p_{(s,i)} \leq \frac{(1+\epsilon')^2}{(1-\epsilon')\sum_{j\in[m]}|D_j|}. \tag{5.3}$$

By definition, we have that $\sum_{j\in[m]}|D_j| = |S|$, and that the number of elements in S such that $\rho(s, i) = 1$ is equal to $|U|$. The probability p of selecting a pair (s, i) from S such that $\rho(s, i) = 1$ is equal to the sum $\sum_{\rho(s,i)=1} p_{(s,i)}$. Hence, it follows from (5.3) that the value of p lies in the range

$$\frac{(1-\epsilon')^2}{(1+\epsilon')}\frac{|U|}{|S|} \leq p \leq \frac{(1+\epsilon')^2}{(1-\epsilon')}\frac{|U|}{|S|}.$$

□

A *poisson trial* X_i is a random variable which takes value 1 with probability p_i and 0 with probability $1 - p_i$. We say that the trials $X_1, \ldots, X_N$ are *independent* if

$$P(X_1 = k_1, \ldots, X_N = k_N) = P(X_1 = k_1) \times \cdots \times P(X_N = k_N).$$

A random variable X is said to *stochastically dominate* [20] a random variable Y if $Pr(X \geq a) \geq Pr(Y \geq a)$ for all a. As a consequence, we have that $E[X] \geq E[Y]$. The following corollary follows directly from Lemma 5.4.

Corollary 5.5. *Let $\{D_1, \ldots, D_m\}, \epsilon, \delta$ and μ be the input to Algorithm 3. Suppose that no call to $approx_{(\epsilon',\delta')}(D_i)$ fails. The random variable $X = \sum_{j\in[N]} X_j$ at Line 19 stochastically dominates a random variable X_{low} which is the sum of N independent Poisson trials with*

$$E[X_{low}] = \frac{(1-\epsilon')^2 N|U|}{(1+\epsilon')|S|}$$

and is stochastically dominated by a random variable X_{high} which is the sum of N indepen-

dent Poisson trials with

$$E[X_{high}] = \frac{(1+\epsilon')^2 N|U|}{(1-\epsilon')|S|}.$$

The following theorem follows directly from Theorems 4.4 and 4.5 of [20] and is known as a *Chernoff Bound*.

Theorem 5.6 ([20])**.** *Let $X_1, \ldots, X_N$ be independent Poisson trials and let $X = \sum_{i=1}^{N} X_i$. For any $0 < \gamma < 1$ we have that*

(1)

$$Pr(X \geq (1+\gamma)E[X]) \leq e^{-(\gamma^2 E[X])/3}$$

and

(2)

$$Pr(X \leq (1-\gamma)E[X]) \leq e^{-(\gamma^2 E[X])/2}.$$

The following lemmas describe an upper bound on the probability that the output of Algorithm 3 lies outside of the allowable limits even when no call to $approx_{(\epsilon',\delta')}(D_i)$ fails. In both cases, we use the Chernoff bound in Theorem 5.6 to bound the probabilities. These bounds will later be used to bound the total probability that our algorithm fails.

Lemma 5.7. *Let $\{D_1, \ldots, D_m\}, \epsilon, \delta$ and μ be the input to Algorithm 3, and let Y denote the output. Assuming that no call to $approx_{(\epsilon',\delta')}(D_i)$ fails, the probability with which Y exceeds that of $|U|$ by more than a factor of $(1+\epsilon)$ is at most $\delta/3$.*

Proof. At Line 20, we set $Y = |S|_{(\epsilon',\delta')} \times X/N$. By Lemma 5.2, we have that

$$|S|_{(\epsilon,\delta')} \leq (1+\epsilon')|S|,$$

It follows that

$$Pr(Y \geq (1+\epsilon)|U|) \leq Pr\left(X \geq \frac{|U|(1+\epsilon)N}{|S|(1+\epsilon')}\right).$$

By Corollary 5.5, the random variable X is stochastically dominated by a random variable X_{high} with

$$E[X_{\text{high}}] = \frac{(1+\epsilon')^2 N|U|}{(1-\epsilon')|S|}. \tag{5.4}$$

Thus, it suffices to show that

$$Pr\left(X_{\text{high}} \geq \frac{|U|(1+\epsilon)N}{|S|(1+\epsilon')}\right) \leq \delta/3. \tag{5.5}$$

In what follows, we show that we can apply the Chernoff bound to the left-hand side of (5.5) to obtain a value that is at most the right-hand side. To apply the Chernoff bound, we must first write the left-hand side of (5.5) in the form $Pr(X_{\text{high}} \geq (1+\gamma)E[X_{\text{high}}])$, so let

$$\frac{|U|(1+\epsilon)N}{|S|(1+\epsilon')} = (1+\gamma)E[X_{\text{high}}]. \tag{5.6}$$

Rearranging (5.6) gives

$$\gamma = \frac{|U|(1+\epsilon)N}{|S|(1+\epsilon')E[X_{\text{high}}]} - 1.$$

It follows from (5.4) that

$$\gamma = \frac{(1+\epsilon)(1-\epsilon')}{(1+\epsilon')^3} - 1.$$

In order to apply the Chernoff bound, it remains to show that $0 < \gamma < 1$. Suppose for a contradiction that this is not the case. Suppose first that $\gamma < 0$. Since $0 < \epsilon < 1$, and we set $\epsilon' = \epsilon/10$, we have that

$$\begin{aligned}
&\frac{(1+\epsilon)(1-\epsilon')}{(1+\epsilon')^2} - 1 < 0 \\
\rightarrow &(1+\epsilon)(1-\epsilon/10) < (1+\epsilon/10)^2 \\
\rightarrow &\frac{1}{10}(1+\epsilon)(10-\epsilon) < \frac{1}{100}(10+\epsilon)^2 \\
\rightarrow &100 + 90\epsilon - 10\epsilon^2 < 100 + 20\epsilon + \epsilon^2 \\
\rightarrow &70/11 < \epsilon.
\end{aligned}$$

Since we have assumed that $\epsilon < 1$, we have a contradiction. Now suppose that $\gamma > 1$, so we have that

$$\begin{aligned}
&\frac{(1+\epsilon)(1-\epsilon')}{(1+\epsilon')^2} - 1 > 1 \\
\rightarrow &(1+\epsilon)(1-\epsilon/10) > 2(1+\epsilon/10)^2 \\
\rightarrow &\frac{1}{10}(1+\epsilon)(10-\epsilon) > \frac{1}{50}(10+\epsilon)^2 \\
\rightarrow &50 + 45\epsilon - 5\epsilon^2 > 100 + 20\epsilon + \epsilon^2 \\
\rightarrow &50 - 25\epsilon + 6\epsilon^2 < 0.
\end{aligned}$$

Again, since $\epsilon < 1$, we have a contradiction. It follows that $0 < \gamma < 1$. Now applying the Chernoff bound to $Pr(X_{\text{high}} \geq (1+\gamma)E[X_{\text{high}}])$ gives

$$Pr(X_{\text{high}} \geq (1+\gamma)E[X_{\text{high}}]) \leq e^{-\left(\gamma^2 \frac{(1+\epsilon')^2 N|U|}{(1-\epsilon')|S|}\right)/3}.$$

Since $\epsilon' = \epsilon/10$ and $\epsilon < 1$, we have that

$$\begin{aligned}
\gamma &= \frac{(1+\epsilon)(1-\epsilon')}{(1+\epsilon')^3} - 1 \\
&= \frac{\epsilon - \frac{4\epsilon}{10} + \frac{\epsilon^2}{10} - \frac{3\epsilon^2}{100} - \frac{\epsilon^3}{1000}}{1 + \frac{3\epsilon}{10} + \frac{3\epsilon^2}{100} + \frac{\epsilon^3}{1000}} \\
&\geq \frac{\epsilon - \epsilon(\frac{5}{10} + \frac{3}{100} + \frac{1}{1000})}{\frac{14}{10}} \\
&> \frac{4/10}{14/10}\epsilon \\
&= \frac{2\epsilon}{7}.
\end{aligned}$$

It follows that $\gamma^2 > \epsilon^2/13$. Since $0 < \epsilon' < 1$, we have that $(1+\epsilon')^2/(1-\epsilon') > 1$. Thus, since $\mu \leq |U|/|S|$, we have that

$$e^{-\left(\gamma^2 \frac{(1+\epsilon')^2 N|U|}{(1-\epsilon')|S|}\right)/3} \leq e^{-\frac{\epsilon^2}{13} \cdot \frac{39 \ln(3/\delta)}{3\epsilon^2 \mu} \cdot \mu} \leq \delta/3.$$

The result follows. □

We now prove that an identical bound holds for the probability that the value of the output of Algorithm 3 is more than a factor of $(1-\epsilon)$ smaller than $|U|$.

Lemma 5.8. *Let $\{D_1, \ldots, D_m\}, \epsilon, \delta$ and μ be the input to Algorithm 3, and let Y denote the output. Assuming that no call to $approx_{(\epsilon',\delta')}(D_i)$ fails, the probability with which Y is more than a factor of $(1-\epsilon)$ smaller than $|U|$ is at most $\delta/3$.*

Proof. The proof of this result follows the same logic as that of Lemma 5.7. By Lemma 5.2, we have that $(1-\epsilon')|S| \leq |S|_{(\epsilon,\delta')}$. Since we set $Y = |S|_{(\epsilon',\delta')} \times X/N$. It follows that

$$Pr(Y \leq (1-\epsilon)|U|) \leq Pr\left(X \leq \frac{(1-\epsilon)N|U|}{(1-\epsilon')|S|}\right).$$

By Corollary 5.5, the random variable X stochastically dominates a random variable X_{low} with

$$E[X_{\text{low}}] = \frac{(1-\epsilon')^2 N|U|}{(1+\epsilon')|S|}. \tag{5.7}$$

Thus, it suffices to show that

$$Pr\left(X_{\text{low}} \leq \frac{(1-\epsilon)N|U|}{(1-\epsilon')|S|}\right) \leq \delta/3. \tag{5.8}$$

To apply the Chernoff bound to the left-hand side of (5.8), we must write it in the form $Pr(X_{\text{low}} \leq (1-\gamma)E[X_{\text{low}}])$, so let

$$\frac{(1-\epsilon)N|U|}{(1-\epsilon')|S|} = (1-\gamma)E[X_{\text{low}}]. \tag{5.9}$$

Rearranging (5.9) gives

$$\gamma = 1 - \frac{(1-\epsilon)N|U|}{(1-\epsilon')|S|E[X_{\text{low}}]}.$$

It follows from (5.7) that

$$\gamma = 1 - \frac{(1-\epsilon)(1+\epsilon')}{(1-\epsilon')^3}.$$

We must now show that $0 < \gamma < 1$. Suppose for a contradiction that $\gamma < 0$. Since $\epsilon' = \epsilon/10$, we have that

$$\begin{aligned}
&1 - \frac{(1-\epsilon)(1+\epsilon')}{(1+\epsilon')^3} < 0 \\
\rightarrow &(1+\epsilon/10)^3 < (1-\epsilon)(1+\epsilon/10) \\
\rightarrow &\frac{1}{1000}(10+\epsilon)^3 < \frac{1}{10}(1-\epsilon)(10+\epsilon) \\
\rightarrow &1000 + 300\epsilon + 30\epsilon^2 + \epsilon^3 < 1000 - 900\epsilon - 100\epsilon^2 \\
\rightarrow &\epsilon^3 + 130\epsilon^2 + 1200\epsilon < 0.
\end{aligned}$$

Since $\epsilon > 0$, we have a contradiction. Now suppose that $\gamma > 1$. We have that

$$\begin{aligned}
&1 - \frac{(1-\epsilon)(1+\epsilon')}{(1+\epsilon')^2} > 1 \\
\rightarrow &\frac{(1-\epsilon)(1+\epsilon')}{(1+\epsilon')^2} < 0.
\end{aligned}$$

Since $0 < \epsilon, \epsilon' < 1$, we have a contradiction. Applying the Chernoff bound to $Pr(X_{\text{low}} \leq (1-\gamma)E[X_{\text{low}}])$ gives

$$Pr(X_{\text{low}} \leq (1-\gamma)E[X_{\text{low}}]) \leq e^{-\left(\gamma^2 \frac{(1-\epsilon')^2 N|U|}{(1+\epsilon')|S|}\right)/2}.$$

Since $\epsilon' = \epsilon/10$ and $\epsilon < 1$, we have that

$$\begin{aligned}\gamma &= 1 - \frac{(1-\epsilon)(1+\epsilon')}{(1-\epsilon')^3} \\ &= 1 - \frac{1 - \frac{9\epsilon}{10} - \frac{\epsilon^2}{10}}{1 - \frac{3\epsilon}{10} + \frac{3\epsilon^2}{100} - \frac{\epsilon^3}{1000}} \\ &\geq 1 - \frac{1 - \frac{9\epsilon}{10}}{1 - \frac{4\epsilon}{10}} \\ &= \frac{5\epsilon}{10 - 4\epsilon}.\end{aligned}$$

It follows that $\gamma > 5\epsilon/6$, and hence $\gamma^2 > 25\epsilon^2/36$. In addition, we have that

$$\begin{aligned}\frac{(1-\epsilon')^2}{(1+\epsilon')} &= \frac{1 - \frac{2\epsilon}{10} + \frac{\epsilon^2}{100}}{1 + \frac{\epsilon}{10}} \\ &> 8/11.\end{aligned}$$

Thus, since $\mu \leq |U|/|S|$ and $N = \left\lceil \frac{39\ln(3/\delta)}{\epsilon^2 \mu} \right\rceil$, we have that

$$e^{-\left(\gamma^2 \frac{(1-\epsilon')^2 N|U|}{(1+\epsilon')|S|}\right)/2} \leq e^{-\frac{25\epsilon^2}{36} \cdot \frac{8}{11} \cdot \frac{39\ln(3/\delta)}{2\epsilon^2 \mu} \cdot \mu} \leq \delta/3$$

as required. □

The following lemma describes an upper bound on the probability that any internal call to $approx_{(\epsilon',\delta')}(D_i)$ fails.

Lemma 5.9. *Let $\{D_1, \ldots, D_m\}, \epsilon, \delta$ and μ be the input to Algorithm 3. The total probability of any call to $approx_{(\epsilon',\delta')}(D_i)$ failing is at most $\delta/3$.*

Proof. At Line 2, we set $\delta' = \delta/(3m)$. By definition, a single call to $approx_{(\epsilon',\delta')}(D_i)$ fails with probability at most δ'. Since there are m iterations of the loop at Line 3, the result follows by a union bound. □

We are now ready to bound the total probability of failure of our algorithm.

Lemma 5.10. *Let $\{D_1, \ldots, D_m\}, \epsilon$ and δ be the input to Algorithm 3. The algorithm fails to output a value within the required range with probability at most δ.*

Proof. Algorithm 3 fails if and only if at least one of the following events occurs:

(i) the value of Y exceeds that of $|U|$ by more than a factor of $(1+\epsilon)$, or

(ii) the value of Y is smaller than that of $|U|$ by more than a factor of $(1-\epsilon)$, or

(iii) any call to $approx_{(\epsilon',\delta')}(D_i)$ fails.

It follows from Lemma 5.7 that (i) occurs with probability at most $\delta/3$. It follows from Lemma 5.8 that (ii) occurs with probability at most $\delta/3$. Finally, it follows from Lemma 5.9 that (iii) occurs with probability at most $\delta/3$. The result then follows by a union bound. □

It remains to bound the runtime of our algorithm. We first obtain bounds in terms of n, δ, ϵ and k for the time taken by each internal call to $approx_{(\epsilon',\delta')}(D_i)$ and $approx\_sample_{(\epsilon')}(D_i)$.

Lemma 5.11. *Let ϵ, δ, μ and $\{D_1, \ldots, D_m\}$ be the input to Algorithm 3. Each call to $approx_{(\epsilon',\delta')}(D_i)$ takes time $g_1(k)f_1(n, 1/\epsilon, \log(1/\delta))$ for some computable function g_1 and a polynomial function f_1.*

Proof. By definition, given D_i, ϵ' and δ' as input, a single invocation of $approx_{(\epsilon',\delta')}(D_i)$ takes time $g(k)f(n, 1/\epsilon', \log(1/\delta'))$ for some computable function g and a polynomial function f. Since $\epsilon' = \epsilon/10$ and $\delta' = \delta/(3m)$, where $m = nh(k)$ for some computable function h of k, the result follows. □

The proof of the following lemma follows the same logic as that of Lemma 5.11.

Lemma 5.12. *Let ϵ, μ and $\{D_1, \ldots, D_m\}$ be the input to Algorithm 3. Each call to $approx\_sample_{(\epsilon')}(D_i)$ terminates in time $g_2(k)f_2(n, \log(1/\epsilon))$ for some computable function g_2 and a polynomial function f_2.*

The following lemma uses Lemma 5.11 and Lemma 5.12 to derive a bound on the runtime of our algorithm.

Lemma 5.13. *Let ϵ, δ, μ and $\{D_1, \ldots, D_m\}$ be the input to Algorithm 3. Algorithm 3 terminates in time $g(k)f(n, 1/\epsilon, \log(1/\delta))$ for some computable function g and a polynomial function f.*

Proof. There are $2^{n^{\mathcal{O}(1)}}$ possible binary strings of length $n^{\mathcal{O}(1)}$. It follows that if any intermediate value in our algorithm cannot be represented using $\mathcal{O}(n^{\mathcal{O}(1)})$ bits, then our algorithm has failed, so we may assume that this is not the case.

By Lemma 5.11, each call to $approx_{(\epsilon')}(D_i)$ takes time $g_1(k)f_1(n, \log(1/\epsilon))$ for some computable function g_1 and a polynomial function f_1. Since $m = nh(k)$ for some computable function h, it follows that we can compute the estimates $|D_i|_{(\epsilon',\delta')}$ in time

$$nh(k) \times g_1(k)f_1(n, 1/\epsilon, \log(1/\delta)).$$

We can then compute the sum

$$|S|_{(\epsilon',\delta')} = \sum_{i\in[m]} |D_i|_{(\epsilon',\delta')}$$

in time

$$\mathcal{O}(h(k)n^{\mathcal{O}(1)}).$$

We now bound the time needed to compute the probability estimates p_i for each $1 \leq i \leq m$ at Line 8. We have already computed the value of $|D_i|_{(\epsilon',\delta')}$ for each $i \in [m]$, as well as the sum $\sum_{i\in[m]} |D_i|_{(\epsilon',\delta')}$. It follows that computing the values of the probability estimates takes time

$$\mathcal{O}(h(k)n^{\mathcal{O}(1)}).$$

It remains to bound the runtime of the loop at Line 11. By Lemma 5.12, each call to $approx\_sample_{(\epsilon')}(D_i)$ terminates in time $g_2(k)f_2(n, \log(1/\epsilon))$ for some computable function g_2 and a polynomial function f_2. We have assumed that deciding membership of any element $s \in \bigcup_{i\in[k]} D_i$ i a set D_i takes time $g_3(k)f_3(n)$ for some polynomial function f_3 and a computable function g_3. Hence, computing the value of X_j takes time

$$nh(k)g_3(k)f_3(n).$$

By Lemma 5.1, we have that

$$\frac{1}{m} \leq \mu \leq 1.$$

Thus, since

$$N = \left\lceil \frac{39\ln(3/\delta)}{\epsilon^2\mu} \right\rceil$$

it follows that the loop at Line 11 takes time at most

$$\begin{aligned} &nh(k)1/\epsilon\log(1/\delta) \times (g_2(k)f_2(n, 1/\epsilon, \log(1/\delta)) + g_3'(k)f_3'(n)) \\ &= g_2'(k)f_2'(n, 1/\epsilon, \log(1/\delta)) \end{aligned}$$

for some polynomial function f_2' and some computable function g_2'.

Finally, given the value of $|S|_{(\epsilon',\delta')}$, as well as the value of X_j for each $j \in [N]$, computing

the output Y takes time

$$\mathcal{O}(n^{\mathcal{O}(1)}nh(k)1/\epsilon\log(1/\delta)).$$

The result follows.

□

We are now ready to prove that Algorithm 3 describes an FPTRAS for the union of sets problem.

Theorem 5.14. *Let n, m and k be positive integers such that $m = nh(k)$ for some computable function h, and let $D_1, \ldots, D_m$ be a collection of m sets whose elements are binary strings of length $n^{\mathcal{O}(1)}$. Suppose that each of the following conditions holds.*

1. *There exists an* FPTRAS *for estimating the size of $|D_i|$ for each $i \in [m]$, and*
2. *There exists an* FPTAUS *for sampling elements almost uniformly at random from D_i for each $i \in [m]$, and*
3. *For any element $s \in \bigcup_{i\in[m]} D_i$, there exists an* FPT *algorithm parameterised by k for deciding whether $s \in D_i$.*

Then, for any real numbers ϵ and δ such that $0 < \epsilon, \delta < 1$, there exists an algorithm which, with probability at least $(1-\delta)$, approximates the size of $|\bigcup_{i\in[m]} D_i|$ within a factor of $(1\pm\epsilon)$ in time $g(k)f(n, 1/\epsilon, \log(1/\delta))$ for some computable function g and a polynomial function f.

Proof. By Lemma 5.10, with probability at least $(1 - \delta)$, Algorithm 3 returns a value within a factor of $(1 \pm \epsilon)$. By Lemma 5.13, the algorithm terminates in time $g(k)f(n, 1/\epsilon, \log(1/\delta))$ for some computable function g and a polynomial function f. The result follows. □

5.6 An FPTRAS for #TYPED SMTI

In this section, we describe an FPTRAS for the problem of counting (weakly) stable matchings admitted by an instance of TYPED SMTI. Our algorithm is based on the FPTRAS for the union of sets problem described in Section 5.5. Note that we will rely on the definitions and notation used in relation to TYPED SMTI given in Chapter 4 (Section 4.5.5). We will also make use of some of the results described in Section 4.5.5.

Let I be an instance of TYPED SMTI. Recall that a stable type set in I is a set of pairs of types in I which guarantees stability. Let $\mathcal{T} = \{T_1, \ldots, T_{|\mathcal{T}|}\}$ denote the set of stable type sets in I. Recall that we use $\mathcal{M}^T$ to denote the set of (stable) matchings admitted by I which

satisfy (i.e. contain only the pairs of types in) the stable type set T. We begin by showing that the problem of counting stable matchings admitted by I is equivalent to computing the size of the union of the sets $\{\mathcal{M}^1, \ldots, \mathcal{M}^{|\mathcal{T}|}\}$. We then show that each of the conditions in Theorem 5.14 hold for these sets, and hence there exists an FPTRAS for #TYPED SMTI. The following observations follow directly from definitions.

Observation 5.15. Let I be an instance of TYPED SMTI, and let $\mathcal{T}$ denote the set of stable type sets in I. Each stable matching admitted by I satisfies at least one type set in $\mathcal{T}$.

Observation 5.16. Let I be an instance of TYPED SMTI, and let T be a stable type set in $\mathcal{T}$. Each matching that satisfies T is stable.

Thus we have the following.

Corollary 5.17. *Let I be an instance of TYPED SMTI. The number of stable matchings admitted by I is equal to $\left|\bigcup_{T\in\mathcal{T}} \mathcal{M}^T\right|$.*

It remains to prove that each of the conditions in Theorem 5.14 hold for the sets $\{\mathcal{M}^1, \ldots, \mathcal{M}^{|\mathcal{T}|}\}$. The following lemma proves the existence of an FPT algorithm for deciding membership in each of the sets $\mathcal{M}^1, \ldots, \mathcal{M}^{|\mathcal{T}|}$ parameterised by the number of agent types.

Lemma 5.18. *Let I be an instance of TYPED SMTI with at most n agents and at most k agent types. Let M be a matching admitted by I, and let T be a type set in $\mathcal{T}$. We can determine whether $M \in \mathcal{M}^T$ in time $\mathcal{O}(nk^2)$.*

Proof. A matching admitted by I contains at most $\lfloor n/2 \rfloor$ pairs of agents, and a type set contains at most $(k+1)^2$ pairs of types. It follows that we can check if the types of each pair of agents in M form a pair in T in time $\mathcal{O}(nk^2)$. □

Corollary 4.33 from Section 4.5 describes a relationship between the number of matchings satisfying a stable type set and the number of perfect matchings in each of at most n balanced bipartite graphs constructed from the problem instance. In [90], Jerrum et al. describe an FPAUS for sampling from the set of perfect matchings in a bipartite graph almost uniformly at random. As a consequence, they also obtain an FPRAS for approximating the number of perfect matchings in a bipartite graph.

Theorem 5.19 ([90])**.** *Let $G = (V(G), E(G))$ be a balanced bipartite graph on n vertices, and let ϵ and δ be real numbers such that $0 < \epsilon, \delta < 1$. There exists an algorithm which takes G, ϵ and δ as input, and with probability at least $(1 - \delta)$ returns an ϵ-approximation of the number of perfect matchings in G in time $f(n, 1/\epsilon, \log(1/\delta))$ for some polynomial function f.*

Theorem 5.20 ([90]). *Let $G = (V(G), E(G))$ be a balanced bipartite graph on n vertices, and let $\epsilon > 0$ be a real number. Let $\mathcal{M}_G$ denote the set of all perfect matchings in G. There exists an algorithm which takes G and ϵ as input and selects a perfect matching M_G from $\mathcal{M}_G$ with probability p such that $(1-\epsilon)/|\mathcal{M}_G| \leq p \leq (1+\epsilon)/|\mathcal{M}_G|$ in time $f(n, 1/\epsilon)$ for some polynomial function f.*

Using Theorems 5.19 and 5.6, we are able to obtain an FPRAS and an FPAUS for approximating and sampling from each set $\mathcal{M}^T$.

Lemma 5.21. *Let I be an instance of TYPED SMTI with at most n agents and at most k agent types, and let T be a stable type set in $\mathcal{T}$. There exists an* FPRAS *for approximating the size of $\mathcal{M}^T$.*

Proof. By Corollary 4.33, we have that

$$|\mathcal{M}^T| = \sum_{0 \leq c < \min(n_w, n_m)} \frac{|\mathcal{M}_{G_c^T}|}{c!(|n_w - n_m| + c)!}$$

where $|\mathcal{M}_{G_c^T}|$ is the number of perfect matchings in the type set graph G_c^T (defined in Section 4.5). Let ϵ and δ be real numbers such that $0 < \epsilon, \delta < 1$, and let $\epsilon' = \epsilon$ and $\delta' = \delta/n$. Observe that ϵ' and δ' are real numbers, and that $0 < \epsilon', \delta' < 1$. By Theorem 5.19, there exists an algorithm which takes G_c^T, ϵ' and δ' as input and, with probability at least $(1-\delta')$, returns a value $|\mathcal{M}_{G_c^T}|_{(\epsilon',\delta')}$ within a factor of $(1 \pm \epsilon')$ of $|\mathcal{M}_{G_c^T}|$ in time $f(n, 1/\epsilon', \log(1/\delta'))$ for some polynomial function f. Let

$$|\mathcal{M}^T|_{(\epsilon',\delta')} = \sum_{0 \leq c < \min(n_w, n_m)} \frac{|\mathcal{M}_{G_c^T}|_{(\epsilon',\delta')}}{c!(|n_w - n_m| + c)!}. \tag{5.10}$$

The probability with which $|\mathcal{M}^T|_{(\epsilon',\delta')}$ lies outside of the range $(1 \pm \epsilon)|\mathcal{M}^T|$ is at most the probability with which any one of the values $|\mathcal{M}_{G_c^T}|_{(\epsilon',\delta')}$ lies outside the range $(1 \pm \epsilon')|\mathcal{M}_{G_c^T}|$. Since $c \leq n$, this occurs with probability at most $\delta' n$. Since $\delta' = \delta/n$, it follows that with probability at least $(1-\delta)$, the value of $|\mathcal{M}^T|_{(\epsilon',\delta')}$ is an ϵ-approximation of $|\mathcal{M}^T|$. It remains to show that computing the value of $|\mathcal{M}^T|_{(\epsilon',\delta')}$ takes time $g(n, 1/\epsilon, \log(1/\delta))$ for some polynomial function g. Computing the approximations $|\mathcal{M}_{G_c^T}|_{(\epsilon',\delta')}$ takes time

$$\begin{aligned} &n \times f(n, 1/\epsilon, \log(1/(\delta/n))) \\ &= f'(n, 1/\epsilon, \log(1/\delta)) \end{aligned} \tag{5.11}$$

for some polynomial function f'. Since $c < \min(n_w, n_m)$, it follows that we can compute each of $c!$ and $(|n_w - n_m| + c)!$ in time $\mathcal{O}(n^2)$, and that each value is at most n^n. It follows

that the denominator of the fraction in (5.10) can be computed in time

$$\begin{aligned}&\mathcal{O}(2n^2 + n\log n)\\ &= \mathcal{O}(n^2)\end{aligned}$$

and has value at most n^{2n}. If the value of $|\mathcal{M}_{G_c^T}|_{(\epsilon,\delta')}$ exceeds $(2n)^{(2n)}$ (the largest possible number of perfect matchings in G_c^T), then we may instead set its value to $(2n)^{(2n)}$. This can be checked in time $\mathcal{O}(n\log n)$. Hence, computing the value of each term in the sum in (5.10) takes time

$$\begin{aligned}&\mathcal{O}(n^2 + n\log n + 2n\log n \times n\log n)\\ &= \mathcal{O}(n^2\log^2 n).\end{aligned}$$

If the value of the sum is more than n^n (the largest possible number of matchings in $\mathcal{M}^T$), then we may immediately return n^n. This can be checked in time $\mathcal{O}(n\log n)$. Otherwise, the sum can be computed in time

$$\begin{aligned}&\mathcal{O}(n \times (n^2\log^2 n + n\log n))\\ &= \mathcal{O}(n^3\log^2 n).\end{aligned} \tag{5.12}$$

It follows from (5.11) and (5.12) that our algorithm takes time

$$\begin{aligned}&\mathcal{O}(f'(n, 1/\epsilon, \log(1/\delta)) + n^3\log^2 n)\\ &= \mathcal{O}(g(n, 1/\epsilon, \log(1/\delta)))\end{aligned}$$

for some polynomial function g. The result follows. □

Lemma 5.22. *Let I be an instance of TYPED SMTI with at most n agents and at most k agent types, and let T be a stable type set in $\mathcal{T}$. There exists an* FPAUS *for sampling almost uniformly from the set $\mathcal{M}^T$.*

Proof. We begin by noting that each matching in the set $\mathcal{M}^T$ of size $\min(n_w, n_m) - c$ corresponds to an equal proportion of the perfect matchings in the type set graph $\mathcal{M}_{G_c^T}$. It is therefore sufficient to describe a method which, given $\mathcal{M}^T$ and a real number $0 < \epsilon < 1$, samples from the set of perfect matchings among all type set graphs G_c^T in such a way that the probability p_{M_G} of picking a particular perfect matching M_G lies in the range

$$\frac{(1-\epsilon)}{\sum_{0\le c<\min(n_m,n_w)} |\mathcal{M}_{G_c^T}|} \le p_{M_G} \le \frac{(1+\epsilon)}{\sum_{0\le c<\min(n_m,n_w)} |\mathcal{M}_{G_c^T}|}.$$

Informally, we proceed by choosing a value c such that c is selected with probability close to

$|\mathcal{M}_{G_c^T}| / \sum_{0 \leq c < \min(n_m, n_w)} |\mathcal{M}_{G_c^T}|$, and then selecting a perfect matching from the set $\mathcal{M}_{G_c^T}$ with close to equal probability.

Let ϵ be a real number such that $0 < \epsilon < 1$, and let $\epsilon_1 = \epsilon_2 = \epsilon/n^2$ and $\delta = \epsilon/(2n)^{3n}$. Observe that ϵ_1, ϵ_2 and δ are real numbers with $0 < \epsilon_1, \epsilon_2, \delta < 1$. By Theorem 5.6, there exists an algorithm which, given G_c^T and ϵ_1 as input, samples from the set $\mathcal{M}_{G_c^T}$ in time $f_1(n, 1/\epsilon_1)$ for some polynomial function f_1 such that each perfect matching is selected with probability p where $(1-\epsilon_1)/|\mathcal{M}_{G_c^T}| \leq p \leq (1+\epsilon_1)/|\mathcal{M}_{G_c^T}|$. Suppose that we have already chosen a value of c, and let $p_{M_G|c}$ denote the probability that we then choose the perfect matching M_G from $\mathcal{M}_{G_c^T}$ by applying the FPAUS from Theorem 5.6 to G_c^T and ϵ_1. We have that

$$\frac{(1-\epsilon_1)}{|\mathcal{M}_{G_c^T}|} \leq p_{M_G|c} \leq \frac{(1+\epsilon_1)}{|\mathcal{M}_{G_c^T}|}. \tag{5.13}$$

By Theorem 5.19, there exists an algorithm which, given G_c^T, δ and ϵ_2 as input, approximates the value of $|\mathcal{M}_{G_c^T}|$ within a factor of $(1 \pm \epsilon_2)$ with probability at least $(1-\delta)$ in time $f_2(n, 1/\epsilon_2, \log(1/\delta))$ for some polynomial function f_2. Let $|\mathcal{M}_{G_c^T}|_{(\epsilon_2,\delta)}$ denote the approximation of $|\mathcal{M}_{G_c^T}|$ provided by a single invocation of this algorithm, and set

$$p_c = \frac{|\mathcal{M}_{G_c^T}|_{(\epsilon_2,\delta)}}{\sum_{0 \leq c < \min(n_w, n_m)} |\mathcal{M}_{G_c^T}|_{(\epsilon_2,\delta)}}.$$

Observe that the probabilities p_c can be obtained using at most n invocations of the algorithm. Thus, it follows from Theorem 5.19 that

$$\frac{(1-\epsilon_2)|\mathcal{M}_{G_c^T}|}{(1+\epsilon_2)\sum_{0 \leq c < \min(n_w, n_m)} |\mathcal{M}_{G_c^T}|} \leq p_c \leq \frac{(1+\epsilon_2)|\mathcal{M}_{G_c^T}|}{(1-\epsilon_2)\sum_{0 \leq c < \min(n_w, n_m)} |\mathcal{M}_{G_c^T}|} \tag{5.14}$$

with probability at least $(1 - n\delta)$.

Now, let p_{M_G} denote the probability of selecting a perfect matching M_G from any one of the graphs G_c^T. We first obtain a lower bound on the probability. With probability at least $(1-n\delta)$, every invocation of the algorithm for estimating each $|\mathcal{M}_{G_c^T}|$ succeeds. In this case, we have that

$$p_c \geq \frac{(1-\epsilon_2)|\mathcal{M}_{G_c^T}|}{(1+\epsilon_2)\sum_{0 \leq c < \min(n_w, n_m)} |\mathcal{M}_{G_c^T}|}.$$

In addition, by (5.13) we have that

$$p_{M_G|c} \geq \frac{(1-\epsilon_1)}{|\mathcal{M}_{G_c^T}|}.$$

It follows that the probability p_{M_G} is such that

$$p_{M_G} \geq \frac{(1-\delta)(1-\epsilon_1)(1-\epsilon_2)}{(1+\epsilon_2)\sum_{0\leq c<\min(n_w,n_m)} |\mathcal{M}_{G_c^T}|}. \tag{5.15}$$

We must show that (5.15) is at least $(1-\epsilon)/\sum_{0\leq c<\min(n_w,n_m)} |\mathcal{M}_{G_c^T}|$. Since $\epsilon_1 = \epsilon_2 = \epsilon/n^2$ and $\delta = \epsilon/(2n)^{3n}$, we have that

$$\begin{aligned}
\frac{(1-\delta)(1-\epsilon_1)(1-\epsilon_2)}{(1+\epsilon_2)} - (1-\epsilon) &= \frac{(1-\epsilon/(2n)^{3n})(1-\epsilon/n^2)(1-\epsilon/n^2) - (1+\epsilon/n^2)(1-\epsilon)}{(1+\epsilon/n^2)} \\
&\geq \frac{(1+\epsilon^2/n^4 - 3\epsilon/n^2) - (1-\epsilon+\epsilon/n^2)}{(1+\epsilon/n^2)} \\
&= \frac{\epsilon(n^2(n^2-4)+\epsilon)}{(1+\epsilon/n^2)n^4}.
\end{aligned}$$

Since $\epsilon > 0$, it follows that if $n \geq 2$ then $p_{M_G} \geq (1-\epsilon)/\sum_{0\leq c<\min(n_w,n_m)} |\mathcal{M}_{G_c^T}|$. If $n < 2$, then counting the number of matchings is trivial, so we may assume without loss of generality that this is not the case.

We now obtain an upper bound on the value of p_{M_G}. With probability at most $n\delta$, the probability p_c lies outside of the bounds given in (5.14). In this case, the most we can say is that $p_c \leq 1$. Otherwise, by (5.14) we have that

$$p_c \leq \frac{(1+\epsilon_2)|\mathcal{M}_{G_c^T}|}{(1-\epsilon_2)\sum_{0\leq c<\min(n_w,n_m)} |\mathcal{M}_{G_c^T}|}.$$

By (5.13), we have that

$$p_{M_G|c} \leq \frac{(1+\epsilon_1)}{|\mathcal{M}_{G_c^T}|}.$$

It follows that

$$\begin{aligned}
p_{M_G} &\leq n\delta \times \frac{(1+\epsilon_1)}{|\mathcal{M}_{G_c^T}|} + \frac{(1+\epsilon_2)|\mathcal{M}_{G_c^T}|}{(1-\epsilon_2)\sum_{0\leq c<\min(n_w,n_m)}|\mathcal{M}_{G_c^T}|} \times \frac{(1+\epsilon_1)}{|\mathcal{M}_{G_c^T}|} \\
&= n\delta \times \frac{(2n^{2n})}{(2n^{2n})} \times \frac{(1+\epsilon_1)}{|\mathcal{M}_{G_c^T}|} + \frac{(1+\epsilon_2)(1+\epsilon_1)}{(1-\epsilon_2)\sum_{0\leq c<\min(n_w,n_m)}|\mathcal{M}_{G_c^T}|} \\
&\leq \frac{\epsilon(1+\epsilon_1)}{2\times\sum_{0\leq c<\min(n_w,n_m)}|\mathcal{M}_{G_c^T}|} + \frac{(1+\epsilon_2)(1+\epsilon_1)}{(1-\epsilon_2)\sum_{0\leq c<\min(n_w,n_m)}|\mathcal{M}_{G_c^T}|} \\
&= \frac{\epsilon(1+\epsilon_1)(1-\epsilon_2)+2(1+\epsilon_2)(1+\epsilon_1)}{2(1-\epsilon_2)\sum_{0\leq c<\min(n_w,n_m)}|\mathcal{M}_{G_c^T}|} \\
&= \frac{\epsilon(1+\epsilon/n^2)(1-\epsilon/n^2)+2(1+\epsilon/n^2)(1+\epsilon/n^2)}{2(1-\epsilon/n^2)\sum_{0\leq c<\min(n_w,n_m)}|\mathcal{M}_{G_c^T}|} \\
&= \frac{\epsilon-\epsilon^3/n^4+2+4\epsilon/n^2+2\epsilon^2/n^2}{2(1-\epsilon/n^2)\sum_{0\leq c<\min(n_w,n_m)}|\mathcal{M}_{G_c^T}|}. \qquad (5.16)
\end{aligned}$$

It remains to show that (5.16) is at most $(1+\epsilon)/\sum_{0\leq c<\min(n_w,n_m)}|\mathcal{M}_{G_c^T}|$. Since

$$\begin{aligned}
&2(1-\epsilon/n^2)(1+\epsilon)-(\epsilon-\epsilon^3/n^4+2+4\epsilon/n^2+2\epsilon^2/n^2) \\
=&\epsilon(1-6/n^2-4\epsilon/n^2+\epsilon^2/n^4),
\end{aligned}$$

it follows that, if $6/n^2+4\epsilon/n^2<1$, then p_{M_G} is at most $(1+\epsilon)/\sum_{0\leq c<\min(n_w,n_m)}|\mathcal{M}_{G_c^T}|$. This occurs as long as $n\geq 4$. If $n<4$, then exactly counting matchings admitted by I is trivial, so we may suppose that this is not the case. It follows that our algorithm provides an ϵ-approximation of $|\mathcal{M}^T|$.

It remains to prove that our algorithm terminates in time $f(n,1/\epsilon)$ for some polynomial function f. Our algorithm makes at most n calls to the FPRAS from Theorem 5.19. It follows that this component of the algorithm requires time at most

$$\begin{aligned}
&n\times f_2(n,1/(\epsilon/n^2),\log(1/(\epsilon/(2n)^{3n}))) \\
&= f_2'(n,1/\epsilon)
\end{aligned}$$

for some polynomial function f_2'. The algorithm also makes a single call to the FPAUS from Theorem 5.6. It follows that this component of the algorithm takes time

$$\begin{aligned}
&f_1(n,1/(\epsilon/n^2)) \\
&= f_1'(n,1/\epsilon)
\end{aligned}$$

for some polynomial function f_1'.

If the output of the FPRAS from Theorem 5.19 cannot be represented using $\mathcal{O}(n\log n)$

bits, then the algorithm has failed. Hence, we can calculate the probabilities p_c in time $\mathcal{O}(n^2 \log^2 n)$. The result follows. □

We are now able to prove our main result.

Theorem 5.23. *There exists an* FPTRAS *for #TYPED SMTI.*

Proof. Let I be an instance of TYPED SMTI. By Corollary 5.17, the number of stable matchings admitted by I is equal to $\left|\bigcup_{T \in \mathcal{T}} \mathcal{M}^T\right|$. It follows from Lemma 4.22 that $1 \leq |\mathcal{T}| \leq 2^{k(k+1)}$. Thus, by Theorem 5.14, if the following conditions hold for the sets $\mathcal{M}^1, \ldots, \mathcal{M}^{\mathcal{T}}$ then there exists an FPTRAS for estimating the number of stable matchings admitted by I.

(1) There exists an FPTRAS for estimating the size of $|\mathcal{M}^T|$ for each $T \in \mathcal{T}$, and

(2) There exists an FPTAUS for sampling elements almost uniformly at random from $\mathcal{M}^T$ for each $T \in \mathcal{T}$, and

(3) There exists an algorithm for determining whether a matching M admitted by I belongs to the set $\mathcal{M}^T$ for any $T \in \mathcal{T}$ in time $n^{\mathcal{O}(1)} f(k)$ for some computable function f.

That (1) holds follows from Lemma 5.21. That (2) holds follows from Lemma 5.22. Finally, Lemma 5.18 confirms that (3) holds. □

5.7 TYPED MAX SMTI with 2 Deletions is W[1]-Hard

In this section, we consider whether we can extend the approximation result from Section 5.6 to the setting where agents may individually declare a small number of their available partners as unacceptable. If we allow deletions in agents' preference lists then the neighbourhood diversity of the acceptability graph associated with an instance may no longer be bounded by the number of agent types, since the neighbourhood of two vertices corresponding to a pair of agents of the same type may differ. It follows that we cannot use the same approach as in Section 4.5 to obtain an XP algorithm for exact counting in the setting with deletions. In what follows, we describe how the FPTRAS from Section 5.6 might be extended to obtain an "XP randomised approximation algorithm" in the setting with deletions when the number of deletions is treated as a constant. We then prove that the problem of finding a maximum size stable matching in this setting is W[1]-hard parameterised by the number of agent types. As a consequence, we argue that it is unlikely that there exists an FPT algorithm for exactly counting stable matchings in this setting.

Once we allow deletions in agents' preference lists, the stability of a matching cannot be determined by looking at only the pairs of agent types in the matching. As in the setting

without deletions, any matching which satisfies a stable type set is stable. However, in this setting, there may be pairs of agents (m, w) who are not matched, and whose types are preferable to the types of their respective partners, but who do not form a blocking pair as long as at least one of them has deleted the other from their preference list. Generalising this notion, a set M_i of men of type i and a set M_j of women of type j can be matched with agents of types less desirable than j and i respectively, so long as in each man-woman pair, at least one agent finds the other unacceptable. Thus, in addition to counting matchings satisfying a stable type set, we must now also count matchings with additional pairs of types which do not break stability due to the presence of deletions.

We can approximate the number of matchings satisfying each stable type set using the same approach as in the setting without deletions. To count the set of stable matchings satisfying each type set T which is not stable, we must consider, for each of the additional "unstable" pairs of types in T, the number of ways that a subset of agents could be matched into pairs of these types without breaking stability. In an instance of TYPED SMTI with at most n agents, at most k types, and at most r deletions per agent, there are $n^{f(r,k)}$ possible subsets for some function f. The number of matchings of the remaining agents satisfying the remaining "stable" set of pairs of types in T can then be approximated as before. Let I be an instance of TYPED SMTI with at most n agents, at most k agent types, and at most r deletions, and let ϵ and δ be real numbers with $0 < \epsilon, \delta < 1$. Given I, ϵ and δ as input, we conjecture that our algorithm returns an $(1 \pm \epsilon)$ approximation of the number of stable matchings admitted by I with probability at least $(1 - \delta)$ in time $n^{f(r,k)}g(k)h(n, 1/\epsilon, \log(1/\delta))$ for some computable function g and a polynomial function h.

In the remainder of this section, we prove that computing the size of a maximum cardinality stable matching admitted by an instance of TYPED SMTI with 2 deletions per agent is W[1]-hard parameterised by the number of agent types needed to describe the instance.

TYPED MAX SMTI WITH 2 DELETIONS	
Input:	An instance I of TYPED SMTI containing at most n agents with at most k agent types, where each agent may declare up to two agents from the other set as unacceptable.
Parameter:	k.
Question:	What is the size of a maximum cardinality stable matching admitted by I?

In what follows, we show that TYPED MAX SMTI WITH 2 DELETIONS is W[1]-hard parameterised by the number of agent types. This is achieved via a reduction from a well-known W[1]-hard problem, MULTICOLOURED CLIQUE. Given a graph $G = (V(G), E(G))$ and a partition of $V(G)$ into colour classes, we say that a clique in G is *multicoloured* if it contains one vertex in each colour class. We define the MULTICOLOURED CLIQUE problem

as follows.

MULTICOLOURED CLIQUE	
Input:	A graph $G = (V(G), E(G))$ on n vertices and a partition $\mathcal{V} = \{V_1, \ldots, V_k\}$ of $V(G)$ into colour classes such that vertices in V_i have colour i for $i \in [k]$.
Parameter:	k.
Question:	Does G contain a multicoloured clique on k vertices?

The W[1]-hardness of MULTICOLOURED CLIQUE can be demonstrated via a reduction from CLIQUE [99]. We note that reducing a parameterised problem from MULTICOLOURED CLIQUE is a common technique used to prove W[1]-hardness.

Theorem 5.24. *TYPED MAX SMTI* WITH 2 DELETIONS *is* W[1]*-hard parameterised by the number of agent types.*

Proof. In what follows, we prove that the problem of deciding whether an instance of TYPED SMTI WITH 2 DELETIONS admits a complete stable matching is W[1]-hard parameterised by the number of agent types via a reduction from MULTICOLOURED CLIQUE. It follows that the problem of finding a maximum size stable matching is also W[1]-hard.

Let $G = (V(G), E(G))$ and $\mathcal{V} = \{V_1, \ldots, V_k\}$ be an instance of MULTICOLOURED CLIQUE with $|V(G)| = n$ and $|E(G)| = m$. In what follows, we construct an instance I of TYPED MAX SMTI WITH 2 DELETIONS from G and $\mathcal{V}$ such that G contains a multicoloured clique on k vertices if and only if I admits a complete stable matching. The set of women in I contains:

- a woman w_v of type i for each vertex $v \in V_i$ and each $i \in [k]$, and
- a set of $\binom{k}{2}$ "bad" women of type B_w, and
- a set of $m - \binom{k}{2}$ "good" women of type G_w.

Observe that I contains a total of $n + m$ women with $k + 2$ distinct types. The set of men in I contains:

- a man $m_{u,v}$ of type (i, j) for each edge $uv \in E(G)$ such that $u \in V_i$, $v \in V_j$ and $1 \leq i < j \leq k$, and
- a "bad" man of type $B_{m,i}$ for each $i \in [k]$, and
- a set of $n - k$ "good" men of type G_m.

Observe that I contains a total of $n + m$ men with $\binom{k}{2} + k + 1$ distinct types. We now construct the preference lists for each type. Note that we write $\{(i, j) : 1 \leq i < j \leq k\}$ to mean that all types of the form (i, j) are contained in a single tie in the preference list.

Women		Men	
Type i:	$G_m, \{(i', j) : 1 \leq i' < j \leq k\}, B_{m,i}$	Type (i, j):	G_w, i, j, B_w
Type G_w:	$\{(i, j) : 1 \leq i < j \leq k\}$	Type G_m:	$\{1, \ldots, k\}$
Type B_w:	$\{(i, j) : 1 \leq i < j \leq k\}$	Type $B_{m,i}$:	i

We only allow men that correspond to edges in G to declare available partners from their preference lists as unacceptable. In particular, the man $m_{u,v}$ corresponding to edge $uv \in E(G)$ finds the two women w_u and w_v unacceptable. Observe that I can be constructed from G and $\mathcal{V}$ in time $\mathcal{O}(n^2)$. We now prove that G contains a multicoloured clique on k vertices if and only if I admits a complete stable matching.

In the first direction, we prove that the existence of a multicoloured clique in G implies the existence of a complete stable matching of the agents in I. Let $U = \{u_1, \ldots, u_k\}$ be a set of k vertices in G which form a multicoloured clique. We may assume that the vertex u_i has colour i for each $1 \leq i \leq k$. We construct a complete matching M of the agents in I as follows. For each $u_i \in U$, we assign woman w_{u_i} to the man of type $B_{m,i}$ in I. The remaining $n - k$ women corresponding to vertices in G are assigned to men of type G_m. Let W denote the set of $\binom{k}{2}$ edges in $E(G)$ with both endpoints in U. Each man corresponding to an edge in W is matched with a woman of type B_w. The remaining $m - \binom{k}{2}$ men corresponding to edges in $E(G)$ are matched with women of type G_w. Observe that M matches all agents in I and that the matched pairs are of the following four pairs of types:

- $(i, B_{m,i})$, $1 \leq i \leq k$
- (i, G_m), $1 \leq i \leq k$
- $((i, j), B_w)$, $1 \leq i < j \leq k$
- $((i, j), G_w)$, $1 \leq i < j \leq k$

We now argue that M is stable. No man of type G_m or $B_{m,i}$ nor any woman of B_w or G_w can form a blocking pair since these agents are all matched with their most preferred type. Similarly, a blocking pair cannot involve a woman of type i who is matched with a man of type G_m or a man of type (i, j) who is matched with a woman of type G_w. It follows that any blocking pair must involve a man m of type (i, j) whose partner is of type B_w, and a woman w of type i (or j) whose partner is of type $B_{m,i}$ (or $B_{m,j}$). It follows from the construction of M that the edge e_m corresponding to m is contained in W and the vertex v_w corresponding to w is in U. Since m is matched with an agent of type B_w, he would prefer to be matched with any woman of type i or j except the two women he has declared unacceptable. However, by our construction, v_w is an endpoint of e_m and so m has deleted w from his preference list. It follows that (m, w) does not form a blocking pair, and so M is stable.

In the other direction, we show that if there is a complete stable matching M admitted by

I, then G contains a multicoloured clique. Since all agents are matched under M, it follows from the construction of the agents' preference lists that all women of types G_w and B_w must be matched with men of types (i, j) for pairs $1 \leq i < j \leq k$, and all men of types G_m and $B_{m,i}$, for each $1 \leq i \leq k$, must be matched with women of types in $[k]$. Recall that there are $\binom{k}{2}$ women of type B_w. Let W denote the set of $\binom{k}{2}$ edges corresponding to the men of types (i, j) for pairs $1 \leq i < j \leq k$ who are matched with women of type B_w, and let U denote the set of k vertices corresponding to women of types in $[k]$ who are matched with men of types $B_{m,i}$ for $1 \leq i \leq k$. Since I contains only one man of type $B_{m,i}$ for each $i \in [k]$, and men of type $B_{m,i}$ find only type i women acceptable, U must contain exactly one vertex corresponding to a woman of type i for each $1 \leq i \leq k$. We shall use w_{u_i} to denote the woman (of type i) corresponding to the vertex $u_i \in U$. It follows from our construction that w_{u_i} must be matched with the single man of type $B_{m,i}$ for each $1 \leq i \leq k$, and that U contains exactly one vertex of each colour.

We will now argue that the vertices in U and the edges in W form a multicoloured clique in G. We begin by proving that all edges in W have both endpoints in U. Suppose for a contradiction that this is not the case. Then there is a man m of type (i, j) who is matched with a woman of type B_w and who has not deleted both of the women w_{u_i} and w_{u_j} from his preference list. Suppose without loss of generality that m has not deleted w_{u_i} from his preference list. Then m and w_{u_i} mutually prefer each other to their partners in M, a contradiction. Finally, since W contains $\binom{k}{2}$ edges, all with both endpoints in U, and U contains k vertices, each of a different colour, it follows that the vertices in U and the edges in W form a multicoloured clique in G. □

To prove Theorem 5.24, we showed that there exists a multicolour clique in the graph constructed from an instance of TYPED SMTI WITH 2 DELETIONS if and only if there exists a stable matching that matches all men of type $B_{m,i}$ and all women of type B_w. There are $\mathcal{O}(k^2)$ agents in the sets $B_{m,i}$ and B_w. Suppose that there exists an FPT algorithm for counting solutions to an instance of TYPED SMTI WITH 2 DELETIONS parameterised by the number of agent types. By running the counting algorithm once with each subset of $B_{m,i} \cup B_w$ removed from the instance, we can use an inclusion-exclusion argument to determine whether there exists a stable matching that matches all men of type $B_{m,i}$ and all women of type B_w. Since we have shown that the latter problem is W[1]-hard, it seems unlikely that there exists an FPT algorithm for counting stable matchings in this setting.

5.8 Remarks and Open Problems

In this chapter, we asked about the complexity of approximately counting solutions to typed stable matching problems. In Section 5.6, we described an FPTRAS for efficiently comput-

ing an arbitrarily close approximation to the number of solutions to an instance of TYPED SMTI for instances with a small number of agent types. Our FPTRAS for #TYPED SMTI relied upon the FPTRAS for the union of sets problem described in Section 5.5. It is an open problem whether there exists an FPTRAS for #TYPED SRTI or #TYPED HRT. We believe that the cloning technique described in Chapter 4 can be used to extend the FPTRAS for #TYPED SMTI to the hospitals/residents setting. Recall that the FPTRAS for #TYPED SMTI relied on the existence of an FPRAS for counting perfect matchings in a bipartite graph. In [100], it is observed that there is no known FPRAS for counting perfect matchings in general graphs, so we cannot immediately use the same approach to obtain an FPTRAS in the stable roommates setting.

In Section 5.7, we proved that under a relaxation of TYPED MAX SMTI in which individual agents may declare up to 2 of their available partners as unacceptable, the problem of finding a maximum size stable matching is W[1]-hard parameterised by the number of agent types. We argued that the existence of an FPT algorithm for exact counting in this setting is unlikely. However, we believe that there exists an XP algorithm for approximately counting solutions parameterised by the number of agent types for any constant number of deletions. The existence of an FPTRAS, or an exact XP algorithm, for counting solutions to instances of TYPED SMTI with a constant number of deletions is an open question.

Chapter 6

Monochromatic Partitioning Problems

6.1 Motivation

The problem of partitioning a large edge-coloured graph into a small number of monochromatic subgraphs is a well-studied problem in combinatorics. Of particular interest has been to determine the minimum number of monochromatic paths or cycles needed to partition the vertex set of a graph G whose edges have been coloured using a fixed number of colours. When G is a complete graph or complete bipartite graph, the problem is particularly well-understood. For instance, if G is a complete graph whose edges are coloured red and blue, then the vertex set of G can always be partitioned into a red cycle and a blue cycle [101, 102, 103].

Very little is currently known about the complexity of monochromatic partitioning problems. What is known suggests that such problems are hard to solve in general - the problem of deciding whether the vertices of an edge-coloured graph can be partitioned into c monochromatic paths for some positive integer c is NP-complete, even when only 2 colours are used and the host graph is complete or complete bipartite [13]. Here, we ask about the complexity of counting the number of ways to partition an edge-coloured graph into monochromatic subgraphs. Counting the number of partitions of an edge-coloured graph into monochromatic subgraphs is at least as difficult as deciding whether such a partition exists. Hence, we do not expect to find a polynomial-time algorithm for counting monochromatic partitions in general. The treewidth of a graph is a measure of how "tree-like" the graph is. Many graph problems that are intractable in general admit polynomial-time algorithms when restricted to graphs with bounded treewidth. In this chapter, we achieve tractability for the problem of counting monochromatic partitions of edge-coloured graphs with small treewidth into monochromatic paths.

6.2 Notation and Definitions

In this section, we provide the notation and definitions needed to describe our and others' results in the area of monochromatic partitioning problems. In Section 6.2.1, we define the terms and notation that we will use to describe monochromatic partitioning problems. In Section 6.2.2, we define terms relating to treewidth and tree decompositions which will be needed in the proof of our main result.

6.2.1 Monochromatic Partitioning Problems

Let $G = (V(G), E(G))$ be a graph. An *edge-colouring* of G is an assignment of integer labels called *colours* to the elements of the set $E(G)$. If at most r distinct colours are used, then we say that G is *r-edge-coloured*. Note that an r-edge-colouring need not be a proper edge-colouring. If each vertex in G is incident to edges with at most r distinct colours, then we say that G is *r-locally edge-coloured*. Note that we do not restrict the total number of colours in this case. It follows that every r-edge-coloured graph is r-locally edge-coloured, but that the converse is not necessarily true. A subgraph $H = (V(H), E(H))$ of an edge-coloured graph G is called *monochromatic* if every edge in the set $E(H)$ receives the same colour in G.

Let $G = (V(G), E(G))$ be an edge-coloured graph. A *vertex partition* of $V(G)$ into c monochromatic paths is a set of c paths such that each path is monochromatic, and every vertex in $V(G)$ is contained in exactly one path. A *vertex cover* of $V(G)$ by c monochromatic paths is a set of c paths such that each path is monochromatic, and every vertex in $V(G)$ is contained in at least one path. Note that in a vertex cover the paths need not be disjoint. The existence of a partition of G into c monochromatic paths implies the existence of a vertex cover by c monochromatic paths.

The *monochromatic path partition number* of an edge-coloured graph is equal to the minimum number of disjoint monochromatic paths required to partition its vertex set. The *monochromatic path cover number* of a graph is equal to the minimum number of (not necessarily disjoint) monochromatic paths needed to cover its vertex set. Analogous terms are used to describe the minimum number of monochromatic trees or cycles required to partition or cover the vertex set of G. We regard singletons and single edges as monochromatic subgraphs in each case so that these numbers are well-defined. Note that if we can partition the vertex set of an edge-coloured graph into c monochromatic cycles then we can certainly partition its vertex set into c monochromatic paths.

6.2.2 Tree Decompositions and Treewidth

A *tree decomposition* [74] of a graph $G = (V(G), E(G))$ is a pair $(T, \{X_t\}_{t \in V(T)})$ where $T = (V(T), E(T))$ is a tree with vertex set $V(T)$ (also called the *nodes* of T) and edge set $E(T)$, and $\{X_t\}_{t \in V(T)}$ is a collection of non-empty subsets of $V(G)$ satisfying the following properties:

- $V(G) = \bigcup_{t \in V(T)} X_t$, and
- for each $uv \in E(G)$, there is some $t \in V(T)$ such that $u, v \in X_t$, and
- for each $v \in V(G)$, let $T(v) = \{t \in V(T) : v \in X_t\}$ - the set $T(v)$ induces a connected subtree in T.

The *width* of the tree decomposition is defined as $(\max_{t \in V(T)} |X_t|) - 1$. The *treewidth* of G is equal to the minimum width over all tree decompositions of G. Intuitively, the treewidth of a graph can be thought of as a measure of how close the graph is to being a tree. A graph has treewidth 1 if and only if it is a tree or a forest. Other examples of graphs with low treewidth include cactus graphs (graphs in which every edge is contained in at most one cycle), pseudo-forests (multigraphs in which every connected component contains at most one cycle), and series-parallel graphs [104]. Every complete graph on n vertices has treewidth $n - 1$.

Let $(T, \{X_t\}_{t \in V(T)})$ be a tree decomposition of a graph G. We may designate an arbitrary vertex in T as the *root* node r of T. We then define a parent-child relationship between adjacent pairs of nodes in T according to their relative distance from the root node. A node t in T is the *parent* of a node t' if t is the neighbour of t' on the unique path from r to t'. We say that t' is the *child* of t. The *leaves* of T are the nodes whose set of children is empty. We say that $(T, \{X_t\}_{t \in V(T)})$ is a *nice tree decomposition* [105] of G if $X_r = \varnothing$, and every non-root node $t \in V(T)$ is of one of the following four types.

- **Leaf node:** a leaf node t is such that $X_t = \varnothing$.
- **Introduce node:** an introduce node t has exactly one child t' and is such that $X_t = X_{t'} \cup \{v\}$ for some vertex $v \in V(G)$. We say that the vertex v is *introduced* at t.
- **Forget node:** a forget node t has exactly one child t' and is such that $X_t = X_{t'} \setminus \{v\}$ for some vertex $v \in V(G)$. We say that the vertex v is *forgotten* at t.
- **Join node:** a join node t has exactly two children t_1 and t_2, and is such that $X_t = X_{t_1} = X_{t_2}$.

Let t be a node of T. We use T_t to denote the subtree of T containing t and all the nodes in T which are connected to r by a path containing t. We shall use V_t to denote the set of vertices in G which are contained in bags indexed by the nodes in T_t.

6.3 Literature Review

In this section, we survey the literature on monochromatic partitioning problems. In Section 6.3.1, we cover results on monochromatic partitioning problems for complete edge-coloured host graphs. We then move onto monochromatic partitioning problems for graphs other than the complete graph. Specifically, we survey results on monochromatic partitioning problems for complete bipartite graphs (Section 6.3.2), graphs with large minimum degree (Section 6.3.3), and host graphs with fixed independence number (Section 6.3.4). In Section 6.3.5, we survey the small number of results on partitioning locally edge-coloured graphs into monochromatic subgraphs. Finally, in Section 6.3.6, we survey the literature on the computational complexity of monochromatic partitioning problems. An excellent survey on these and other monochromatic partitioning problems was provided by Gyárfás in 2016 [106].

6.3.1 Partitioning Complete Edge-Coloured Graphs into Monochromatic Subgraphs

The most well-studied monochromatic partitioning problems ask about the minimum number of monochromatic paths or cycles needed to partition the vertices of a complete r-edge-coloured host graph into monochromatic paths and cycles. The first result of this kind appeared as a footnote of a paper due to Gerencsér and Gyárfás in 1967 [107].

Theorem 6.1 ([107])**.** *Any 2-edge-coloured K_n can be partitioned into two disjoint monochromatic paths of different colours.*

In a later paper, Gyárfás [108] asked whether their result can be extended to include complete graphs whose edges have been coloured using more than two colours.

Conjecture 6.2 ([108])**.** *The monochromatic path partition number of an r-edge-coloured K_n is equal to r.*

Conjecture 6.3 ([108])**.** *The monochromatic path cover number of an r-edge-coloured K_n is equal to r.*

Figure 6.1 contains an example of a 2-edge-coloured K_4 which demonstrates that both conjectures are optimal i.e. we cannot cover the vertex set using fewer than two monochromatic paths or cycles.

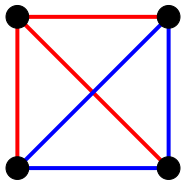

Figure 6.1: A 2-edge-coloured K_4 with monochromatic path partition number 2

The $r = 3$ case of Conjecture 6.2 (and hence Conjecture 6.3) was eventually settled by Pokrovskiy in 2012 [109]. The $r \geq 4$ case of both conjectures remains open. In 1991, Erdős et al. [110] asked whether the following stronger version of Conjecture 6.2 holds.

Conjecture 6.4 ([110])**.** *The monochromatic cycle partition number of an r-edge-coloured K_n is equal to r.*

Again, Figure 6.1 shows that the conjecture is optimal. The $r = 2$ case of Conjecture 6.4 was first verified only for sufficiently large host graphs [101, 102]. The conjecture was settled in full by Bessy and Thomassé [103], who made the following stronger observation.

Theorem 6.5 ([103])**.** *The vertex set of any 2-edge-coloured K_n can be partitioned into two disjoint monochromatic cycles of different colours.*

It follows that the monochromatic cycle cover number of a 2-edge-coloured K_n is also equal to 2. In 2014, Pokrovskiy [109] described a counterexample to Conjecture 6.4 for $r = 3$. However, each of Pokrovskiy's counterexamples admits a partition into four monochromatic cycles. Moreover, Letzter [111, 112] showed that three disjoint monochromatic cycles partition all but at most 60 vertices of any 3-edge-coloured K_n. Hence, it is possible that a weaker version of Conjecture 6.4 holds. In [109], Pokrovskiy proposed the following.

Conjecture 6.6. *([109]) For each r, there exists a constant c_r such that at most r monochromatic cycles are needed to partition all but c_r vertices of any r-edge-coloured K_n.*

The best known upper bound on the monochromatic cycle partition number of an r-edge-coloured K_n for general r is $\lceil 25r^2 \log r \rceil$ [110, 113]. For large enough n, this has been improved to $100r \log r$ [114]. The problem of partitioning complete edge-coloured graphs into monochromatic trees was first examined by Erdős et al. in [110]. Note that the monochromatic tree partition (respectively cover) number of a graph is less than or equal to its monochromatic path partition (respectively cover) number, since any path is a tree.

Conjecture 6.7 ([110])**.** *The monochromatic tree cover number of any r-edge-coloured K_n is equal to $r - 1$.*

Conjecture 6.8 ([110]). *The monochromatic tree partition number of any r-edge-coloured K_n is equal to $r-1$.*

The authors of [110] showed that the monochromatic tree cover number of an r-edge-coloured K_n is at least $r-1$ (so both conjectures are best possible) and at most r. It follows from a result of Tuza [110, 115] that Conjecture 6.7 is true for all $r \leq 5$. The conjecture remains open for $r > 5$. The $r = 2$ case of Conjecture 6.8 follows from the fact that either a graph or its complement is connected, an old remark of Erdős and Rado [110]. The $r = 3$ case is settled in [110]. The conjecture remains open for all $r \geq 4$ but has been verified in part by Haxell and Kohayakawa [116] on the set of sufficiently large graphs.

6.3.2 Partitioning Complete Bipartite Graphs into Monochromatic Subgraphs

Initially explored as a stepping stone towards solving related problems on complete graphs, recent years have seen significant progress in solving monochromatic partitioning problems on complete bipartite host graphs. The first result of this kind appeared in a paper due to Erdős et al. [110] in 1991.

Theorem 6.9 ([110]). *The monochromatic cycle cover number of an r-edge-coloured $K_{n,n}$ is $\mathcal{O}(r^2)$.*

In the same paper, it was asked whether a similar bound exists for the monochromatic cycle partition number.

Conjecture 6.10 ([110]). *The monochromatic cycle partition number of an r-edge-coloured $K_{n,n}$ is a function of r.*

A positive answer to Conjecture 6.10 was given by Haxell [117], who showed that $\mathcal{O}(r^2 \log^2 r)$ cycles suffice. In addition, Haxell showed that 1695 disjoint monochromatic cycles partition the vertices of any 3-edge-coloured $K_{n,n}$. For sufficiently large host graphs, Lang et al. [118] showed that at most 18 cycles are needed. They also showed that there is a partition of all but $o(n)$ vertices of a 3-edge-coloured $K_{n,n}$ into five disjoint monochromatic cycles. In 2014, Pokrovskiy [109] proved the following result.

Theorem 6.11 ([109]). *The monochromatic path partition number of a 2-edge-coloured $K_{n,n}$ is equal to 3.*

In the same paper, Pokrovskiy asked whether his result can be generalised to more than two colours.

Conjecture 6.12 ([109]). *The monochromatic path partition number of an r-edge-coloured $K_{n,n}$ is equal to $2r-1$.*

Conjecture 6.12 is optimal [109], and remains open for all $r \geq 3$. The best known bound on the monochromatic cycle and path partition numbers of an r-edge-coloured $K_{n,n}$ is $\mathcal{O}(r^2 \log r)$ [119]. In [110], Erdős et al. described the following bound on the monochromatic tree partition number of an r-edge-coloured $K_{n,n}$.

Theorem 6.13 ([110]). *The monochromatic tree partition number of an r-edge-coloured $K_{n,n}$ is $\mathcal{O}(r^2)$.*

For sufficiently large n, the monochromatic tree partition number of an r-edge-coloured $K_{n,n}$ is at most $2r$ [116, 117]. It follows from Theorem 6.13 that the monochromatic tree cover number of an r-edge-coloured $K_{n,n}$ is $\mathcal{O}(r^2)$. The authors of [120] suggested the following improvement on this value.

Conjecture 6.14 ([120]). *The monochromatic tree cover number of an r-edge-coloured $K_{n,n}$ is at most $2r-2$.*

In the same paper, it was shown that Conjecture 6.14 is best possible and holds for all $r \leq 5$. In addition, the authors gave the following upper bound.

Theorem 6.15 ([120]). *The monochromatic tree cover number of an r-edge-coloured $K_{n,n}$ is at most $2r-1$.*

6.3.3 Minimum Degree Conditions for Monochromatic Subgraph Partitioning

Balogh et al. [121] asked about a generalisation of the above monochromatic partitioning problems where the host graph has large minimum degree. Results of this kind ask about the relationship between the minimum degree of an edge-coloured graph and the number of monochromatic subgraphs needed to partition its vertices. We use $\delta(G)$ to denote the minimum degree of a graph G. The following conjecture is an analogue of the result due to Bessy and Thomassé [103] on partitioning complete 2-edge-coloured graphs into monochromatic cycles.

Conjecture 6.16 ([121]). *The vertices of any graph G with $\delta(G) \geq 3/4n$ whose edges have been coloured in red and blue can be partitioned into a red cycle and a disjoint blue cycle.*

Conjecture 6.16 is best possible [112] and has been verified asymptotically [121]. In [122], Pokrovskiy asked about the number of monochromatic cycles needed to partition the vertices of an even sparser 2-edge-coloured graph.

Conjecture 6.17. *[[122]] The monochromatic cycle partition number of any sufficiently large 2-edge-coloured graph G with $\delta(G) \geq 2n/3$ is equal to 3.*

Conjecture 6.18 ([122])**.** *The monochromatic cycle partition number of any sufficiently large 2-edge-coloured graph G with $\delta(G) \geq n/2$ is equal to 4.*

Both conjectures are best possible [122]. Conjecture 6.17 has been answered approximately by Allen et al. [123]. For general r, the following observation was made in [112].

Theorem 6.19 ([112])**.** *The monochromatic cycle partition number of an r-edge-coloured graph G of order n with $\delta(G) \geq n/2 + cr \log n$ for some constant c is $\mathcal{O}(r^2)$.*

The authors of [112] also provided a construction showing that Theorem 6.19 is best possibl.

6.3.4 Partitioning Graphs with Bounded Independence Number into Monochromatic Subgraphs

The *independence number* $\alpha(G)$ of a graph G is equal to the size of its largest independent set. The following two results describe all that is currently known about the relationship between the independence number of an edge-coloured graph and the number of monochromatic subgraphs needed to partition its vertices.

Theorem 6.20 ([124])**.** *An r-edge-coloured graph G with independence number $\alpha(G)$ has monochromatic cycle partition number at most $25(\alpha(G)r)^2 \log(\alpha(G)r)$.*

Theorem 6.21 ([121])**.** *At most $2\alpha(G)$ monochromatic cycles are needed to partition the vertices of any sufficiently large 2-edge-coloured graph G.*

6.3.5 Partitioning Locally Edge-Coloured graphs into Monochromatic Subgraphs

Generalisations of monochromatic partitioning problems to locally edge-coloured host graphs were first considered by Conlon and Stein in [125]. The following is an analogous result to Theorem 6.5 for r-locally edge-coloured complete host graphs.

Theorem 6.22 ([125])**.** *The vertex set of any 2-locally edge-coloured K_n can be partitioned into two disjoint monochromatic cycles of different colours.*

The authors of [125] also provide an upper bound on the monochromatic cycle number of r-locally edge-coloured complete host graphs, generalising a result of Erdős et al. [110].

Theorem 6.23 ([125]). *The vertex set of any r-locally edge-coloured K_n can be partitioned into $\mathcal{O}(r^2 \log r)$ disjoint monochromatic cycles.*

In [126], Lang and Stein improve upon the bound in Theorem 6.23.

Theorem 6.24 ([126]). *The vertex set of any r-locally edge-coloured K_n can be partitioned into $\mathcal{O}(r^2)$ disjoint monochromatic cycles.*

In the same paper, the authors provide an analogous result to Theorem 6.11 for 2-locally edge-coloured complete bipartite host graphs. They also provide a bound on the monochromatic cycle partition number of an r-locally edge-coloured $K_{n,n}$.

Theorem 6.25 ([126]). *The vertex set of any 2-locally edge-coloured $K_{n,n}$ can be partitioned into three disjoint monochromatic paths.*

Theorem 6.26 ([126]). *The vertex set of any r-locally edge-coloured $K_{n,n}$ can be partitioned into $\mathcal{O}(r^2)$ disjoint monochromatic cycles.*

The *radius* of a graph $G = (V(G), E(G))$ is equal to the minimum, over all vertices v in $V(G)$, of the maximum distance from any other vertex in $V(G)$ to v. In a recent paper, Sárközy [127] provided the following upper bounds on the number of monochromatic trees or cycles needed to partition an r-locally edge-coloured graph.

Theorem 6.27 ([127]). *The vertex set of any r-locally edge-coloured K_n with $n \geq r^{2(r+2)}$ can be partitioned into r monochromatic trees of radius at most 2, such that each tree is of a different colour.*

Theorem 6.28 ([127]). *The monochromatic cycle partition number of an r-locally edge-coloured K_n is $\mathcal{O}(r \log r)$.*

6.3.6 Complexity Results on Monochromatic Partitioning Problems

We use PMP to denote the computational problem of deciding whether an edge-coloured input graph can be partitioned into c or fewer monochromatic paths for some positive integer c. The equivalent problems of counting partitions into monochromatic cycles or trees are denoted by PMC and PMT respectively. If the number r of colours used to label the edges is fixed, then we denote each of these problems by r-PMP, r-PMC and r-PMT respectively. It follows from the above results that, for certain types of input graphs, we already have bounds on the value of c for which the answer to these problems is definitely "yes". If the input graph is a complete graph then the answer to each of r-PMP and r-PMC is "yes" for all

$c \geq \lceil 25r^2 \log r \rceil$ [110, 113]. It follows from [107, 103] that the answer to each of 2-PMP and 2-PMC is "yes" for any $c \geq 2$ when the input graph is complete, and the answer to 3-PMP is "yes" whenever $c \geq 3$ [109]. For partitions of complete graphs into trees, the answer to r-PMT is "yes" for all $c \geq r$ [110].

Given a set S, and a collection of m subsets $C_1, \ldots, C_m$ of S whose union is equal to S, the *set cover problem* asks whether there exist k sets from among the sets $C_1, \ldots, C_m$ whose union is S. It was shown by Jin and Li [128] that PMP and PMC are NP-complete in general via a reduction from the set cover problem. They also showed that PMT is NP-complete for bipartite host graphs using a reduction from set cover. In addition, it was shown that there is no constant factor approximation algorithm for any of PMP, PMC or PMT unless $\mathsf{P} = \mathsf{NP}$ [128].

Given a graph G, the *Hamiltonian path problem* asks whether G contains a spanning path. The *Hamiltonian cycle problem* asks whether G contains a spanning cycle. Both problems are NP-complete in general [15]. It follows that for $c = 1$, even 1-PMP and 1-PMC are NP-complete. In [129], it is shown that r-PMP, r-PMC and r-PMT are NP-complete for any fixed $r \geq 5$. A more recent paper due to Jin et al. [13] showed that both 2-PMP and 2-PMC are NP-complete for complete graphs via a reduction to the Hamiltonian path and Hamiltonian cycle problems respectively. It was shown by Golumbic [130] that the Hamiltonian path and Hamiltonian cycle problems are NP-complete even for bipartite graphs. In [13], it is shown that 2-PMP and 2-PMC are NP-complete for complete bipartite host graphs via a reduction to the Hamiltonian path (respectively Hamiltonian cycle) problem in bipartite graphs. In the same paper, it is shown that 2-PMT is NP-complete for bipartite host graphs for $c = 2$ via a reduction to 3SAT. Conversely, it is shown that 2-PMT is polynomial-time solvable for complete multipartite graphs (including complete bipartite graphs but not including complete graphs).

6.4 Contributions

In Section 6.5, we describe an FPT algorithm to count the number of ways to partition an r-locally edge-coloured graph into at most c monochromatic paths for any integer c, parameterised by r and the treewidth of the input graph. Our algorithm follows a standard dynamic programming approach for solving problems on graphs with bounded treewidth. As a consequence of our result, we obtain an FPT algorithm for counting partitions of r-edge-coloured graphs under the same parameterisation. We also obtain an XP algorithm for counting partitions of any edge-coloured graph into monochromatic paths parameterised by the treewidth of the input graph.

It is natural to ask whether our FPT result is a corollary of Courcelle's famous meta-theorem

on solving problems on graphs with bounded treewidth. *Monadic second-order logic* (MS_2 logic) [17] is an extension of first-order logic which allows quantification over sets of variables as well as over individual variables. Many important graph properties are expressible in MS_2 logic. The following statement is known as *Courcelle's theorem*.

Theorem 6.29 ([131]). *Let $G = (V(G), E(G))$ be a graph with n vertices and treewidth w. Let Φ be an MS_2 sentence of length $|\Phi|$ describing some graph property. We can decide whether G has property Φ in time $nf(w, |\Phi|)$ for some computable function f.*

It follows that if a graph property can be written as an MS_2 sentence whose length is independent of the size of the graph, then the graph property is decidable in linear time for graphs of bounded treewidth. A counting variant of Courcelle's theorem is provided in [35]. Note that the underlying algorithm of Courcelle's theorem has an extremely large dependence on $|\Phi|$ and w [132], and so is rarely useful in practice. Our FPT result does not enforce that the number of monochromatic paths partitioning the vertices of the input graph should be independent of the order of the graph. Hence, it seems unlikely that our problem can be written as an MS_2 formula whose length is independent of n. As such, we do not believe that our result follows from Courcelle's theorem.

6.5 An FPT Algorithm for Partitions into Monochromatic Paths

Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph. Recall that a partition of $V(G)$ into c monochromatic paths is a set of c paths such that each path is monochromatic, and every vertex in $V(G)$ is contained in exactly one path. We define the following problem.

LOCAL-#PMP	
Input:	An r-locally edge-coloured graph $G = (V(G), E(G))$ with at most n vertices, and a positive integer c.
Parameters:	The treewidth w of G, and r.
Question:	In how many ways can we partition the set $V(G)$ into at most c (vertex-disjoint) monochromatic paths?

In what follows, we describe an FPT dynamic programming algorithm for local-#PMP parameterised by r and w. We begin by providing a summary of our method.

6.5.1 Proof Overview

Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with at most n vertices and treewidth at most w. We use existing results to efficiently obtain a nice tree decomposition $(T, \{X_t\}_{t \in V(T)})$ of G with width at most w and $\mathcal{O}(n)$ nodes. Our algorithm then works by recursively computing the number of partial solutions (i.e. partitions into monochromatic paths) for the subset of $V(G)$ present in the bags indexed by each subtree of T, starting with the subtrees containing only the leaves. A "valid state" of a bag describes how a partial solution might interact with the set of vertices in the bag. We will see that the number of valid states of any bag is bounded by a function of c, r and w. The "signature" of a bag maps each valid state to the number of partial solutions which interact with the bag in the way described by the state. For each type of node t in T, we show how to efficiently compute the signature of t from the signature(s) of its child(ren). The number of partitions of $V(G)$ into at most c monochromatic paths can then be obtained directly from the signature of the root node of T.

In Section 6.5.2, we provide the definitions and preliminaries needed to describe our algorithm, including an observation on the time needed to compute the signature of a leaf node. In Sections 6.5.3 to 6.5.5, we describe how to compute the signature of an introduce, forget and join node respectively from the signature(s) of their child node(s). In Section 6.5.6, we bring together the observations from the previous sections to obtain an FPT dynamic programming algorithm for local-#PMP parameterised by r and the treewidth of the input graph.

6.5.2 Preliminaries

In this section, we provide some additional definitions and preliminary results needed to describe our algorithm. The following results describe the complexity of obtaining a nice tree decomposition of a graph with bounded treewidth.

Theorem 6.30 ([133])**.** *Let $G = (V(G), E(G))$ be a graph with at most n vertices and treewidth at most w. There exists an algorithm for finding a tree decomposition of G with width at most w and $\mathcal{O}(n)$ nodes in time $\mathcal{O}(w^{\mathcal{O}(w^3)}n)$.*

Note that the runtime bound in Theorem 6.30 is not specified in the original paper, but is given in [134]. The following result describes the complexity of obtaining a nice tree decomposition of a graph when a tree decomposition with width at most w is provided.

Theorem 6.31 ([105]). *Let $G = (V(G), E(G))$ be a graph with at most n vertices and with treewidth at most w. There exists an algorithm which, given a tree decomposition of G with width at most w and $\mathcal{O}(n)$ nodes, outputs a nice tree decomposition of G with width at most w and $\mathcal{O}(n)$ nodes in $\mathcal{O}(w^2 n)$ time.*

Again, the runtime bound in Theorem 6.31 does not appear in [105], but is given in Lemma 7.4 of [18]. Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Note that there can be at most rn different coloured edges in $E(G)$. Let t be a node in $V(T)$, and let σ_t be a partition of V_t into monochromatic paths. Let p be a path in σ_t. We call any edge uv on p such that $u, v \in X_t$ a *true-edge* of p in X_t. Note that any true-edge uv on p is also an edge in $E(G)$. If p contains a (u, v)-subpath of colour i containing at least one vertex that is not u or v, whose intersection with X_t is only u and v, then we call uv a *pseudo-edge* of colour i on p in X_t. In addition, if p contains a subpath of colour i whose intersection with X_t is a single vertex $v \in X_t$, and the subpath contains at least one other vertex, then we may also call vv a pseudo-edge of colour i on p in X_t. Note that a pseudo-edge need not be an edge in $E(G)$. We call the sequence of true- and pseudo-edges on p formed from vertices in X_t a *pseudo-path*. We say that a pseudo-path is a *monochromatic pseudo-path* if every true-edge and every pseudo-edge on the path have the same colour. A monochromatic pseudo-path in X_t describes how a single monochromatic path in the partition σ_t interacts with the set of vertices in X_t.

A *state* of a bag X_t describes how a partition of V_t into monochromatic paths might interact with X_t. Formally, a state of X_t is a pair $st(t) = (f, k)$ where

- $f : X_t^2 \to \{-rn, \ldots, 0, 1\}$ is a surjective function describing the pairs of vertices in X_t which are on monochromatic pseudo-paths. For each pair $(u, v) \in X_t^2$ we have that $f(uv) = f(vu)$, where
  - $f(uv) = 1$ if uv is a true-edge on a monochromatic pseudo-path in X_t, and
  - $f(uv) = -i$ if uv is a pseudo-edge of a monochromatic pseudo-path of colour i in X_t, and
  - $f(uv) = 0$ if uv is neither a true-edge nor a pseudo-edge of any monochromatic pseudo-path in X_t.
- $k \in \mathbb{Z}^+$ describes the number of "completed" paths so far i.e. the number of monochromatic paths in the partition of V_t which do not use any vertices in X_t. Note that we require that $k \leq c$ since we are allowed at most c paths in total.

A state $st(t) = (f, k)$ of X_t is said to be *valid* if

- for each $(u, v) \in X_t^2$ such that $f(uv) = 1$, the edge uv is present in $E(G)$, and

- the subgraphs formed from pairs $(u, v) \in X_t^2$ such that $f(uv) \neq 0$ partition X_t into at most k monochromatic pseudo-paths.

Considering only valid states prevents us from counting false partial solutions. We denote the set of all valid states of X_t by $ST(t)$. We denote the size of this set by $|ST(t)|$. We say that a partition σ_t of V_t into monochromatic paths *satisfies* a state $st(t) = (f, k)$ if the following hold:

- for each pair $(u, v) \in X_t^2$ we have that
  - if uv is a true-edge of a monochromatic pseudo-path of σ_t in X_t then $f(uv) = 1$.
  - if uv is a pseudo-edge of colour i on a monochromatic pseudo-path of σ_t in X_t then $f(uv) = -i$.
  - if uv is not an edge of any path in σ_t then $f(uv) = 0$.
- there are exactly k paths in σ_t which do not have any vertices in X_t

Observe that a single partition of V_t into monochromatic paths satisfies exactly one valid state of X_t. It follows that to determine the number of partial solutions at a particular node $t \in V(T)$, it suffices to know the number of partitions satisfying each valid state of X_t. We will use $S(st(t))$ to denote the set of all partitions of V_t into monochromatic paths satisfying $st(t)$. We use $|S(st(t))|$ to denote the number of such partitions. The *signature* $sig(t)$ of a node t is a mapping from each valid state $st(t)$ of X_t to the number of partitions of V_t into monochromatic paths satisfying the state $st(t)$. Observe that the number of partitions of $V(G)$ into at most c monochromatic paths can be obtained from the signature $sig(r)$ of the root node r of T. Given any node $t \in V(T)$, the following lemma describes the complexity of deciding whether a state $st(t) = (f, k)$ of X_t is valid.

Lemma 6.32. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be a node in T, and let $st(t) = (f, k)$ be a state of X_t. We can determine whether $st(t)$ is a valid state of X_t in time $\mathcal{O}(w^2)$.*

Proof. In order to check whether $st(t)$ is a valid state of X_t, we must check whether the subgraphs formed from pairs $(u, v) \in X_t^2$ such that $f(uv) \neq 0$ partition X_t into at most k monochromatic pseudo-paths. To achieve this, we first obtain the set of connected components formed from pairs $(u, v) \in X_t^2$ such that $f(uv) \neq 0$. By definition, since G has treewidth at most w, the set X_t contains at most $(w + 1)$ vertices. Hence, we can obtain the set of components in time $\mathcal{O}(w^2)$ using a standard depth-first search. It then remains to check that these components form non-intersecting monochromatic pseudo-paths. Since X_t contains at most $(w + 1)$ vertices, there can be at most $(w + 1)$ components, and each component can contain at most $(w + 1)$ vertices. It follows that we can verify that no two

components share a vertex in time $\mathcal{O}(w^2)$. In addition, we can verify that every component forms a monochromatic pseudo-path in time $\mathcal{O}(w^2)$. The result follows. □

The following lemma describes an upper bound on the number of valid states of a bag.

Lemma 6.33. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. For any $t \in V(T)$, we have that $|ST(t)| \leq c(r+2)^{(w+1)^2}$.*

Proof. Let $st(t) = (f, k)$ be a valid state of X_t. By definition, there are at most $(w+1)^2$ pairs in X_t^2. Since each vertex is incident to at most r edges of different colours, it follows that for each pair $(u, v) \in X_t^2$, the function $f(uv)$ can take one of at most $(r+2)$ different values. Since we also require that $k \leq c$, it follows that there can be at most $c(r+2)^{(w+1)^2}$ possible valid states of X_t. □

The following lemma bounds the time needed to construct the set of valid states of a bag.

Lemma 6.34. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. For any $t \in V(T)$, we can construct the set $ST(t)$ of valid states of X_t in time $\mathcal{O}(cw^2(r+2)^{(w+1)^2})$.*

Proof. For any valid state $st(t) = (f, k)$ of X_t we require that $k \leq c$. It follows that we should only consider states such that $k \leq c$. Since there are at most $(w+1)$ vertices in X_t, and each vertex is incident to edges of at most r different colours, it follows that there are at most $c(r+2)^{(w+1)^2}$ possible states. For each of these possibilities, it follows from Lemma 6.32 that we can check if the state is valid in time $\mathcal{O}(w^2)$. The result follows. □

It follows from the definition of a nice tree decomposition that any bag indexed by a leaf node is empty. As such, it is straightforward to compute the signature of a leaf node.

Lemma 6.35. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be a leaf node in T. We can compute the signature $sig(t)$ of t in time $\mathcal{O}(c)$.*

Proof. By definition, we have that $X_t = \varnothing$. It follows that there can be at most c states of X_t (one for each value of k), and that each such state is valid. Moreover, the only state which corresponds to a non-zero number of partitions is the state with $k = 0$. For this particular state, we have that $|S(st(t))| = 1$. The result follows. □

6.5.3 Introduce Nodes

In this section, we describe how to compute the signature of an introduce node from the signature of its child. Let t be an introduce node in T, and let $st(t) = (f, k)$ be a valid state of X_t. Let t' denote the child of t in T. In the following lemma, we describe how to construct a state $st(t') = (f', k')$ of $X_{t'}$ such that the number of partitions of V_t into monochromatic paths satisfying $st(t)$ is equal to the number of partitions of $V_{t'}$ into monochromatic paths satisfying $st(t')$.

Lemma 6.36. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be an introduce node in T, and let t' be the child of t in T. Let v be the vertex introduced at t, and let $st(t) = (f, k)$ be a valid state of X_t. Let $st(t') = (f', k')$ be the state of $X_{t'}$ such that $k' = k$ and $f'(uu') = f(uu')$ for each $(u, u') \in X_{t'}^2$. We have that*

$$|S(st(t))| = |S(st(t'))|.$$

Proof. In what follows, we describe a bijection between the set $S(st(t))$ and the set $S(st(t'))$. Specifically, we show that for each partition $\sigma_t \in S(st(t))$, removing v from σ_t produces a partition $\sigma_{t'} \in S(st(t'))$. Conversely, we show that for each partition $\sigma_{t'} \in S(st(t'))$ we can add v (and one or more edges incident to v according to the value of $f(uv)$ for each $u \in X_t$) to $\sigma_{t'}$ to obtain a partition $\sigma_t \in S(st(t))$.

In the first direction, let σ_t be a partition of V_t into monochromatic paths satisfying $st(t)$. Let $\sigma_{t'}$ be the partition of $V_{t'}$ into monochromatic paths formed by removing v (and its incident edges) from σ_t. Note that $\sigma_{t'}$ is indeed a partition of $V_{t'}$ into monochromatic paths since removing a vertex and its incident edges from a path produces one or more disjoint paths. We must now show that $\sigma_{t'}$ satisfies $st(t')$. It follows from the definition of a tree decomposition that there can be no path in σ_t which contains v and vertices in $V_t \setminus X_t$ but which does not intersect $X_{t'}$. Hence, the number of paths in σ_t without a vertex in X_t is equal to the number of paths in $\sigma_{t'}$ without a vertex in $X_{t'}$. By construction, we have that $f'(uu') = f(uu')$ for all pairs $(u, u') \in X_{t'}^2$. It follows that $\sigma_{t'}$ satisfies $st(t')$.

In the other direction, let $\sigma_{t'}$ be a partition of $V_{t'}$ into monochromatic paths satisfying $st(t')$. Let σ_t be the partition of V_t into monochromatic paths formed by adding v to $\sigma_{t'}$ and, for each $u \in X_t$ such that $f(uv) = 1$, adding the edge uv to the path p in $\sigma_{t'}$ containing u so that v is adjacent to u on p. Since v is not contained in $X_{t'}$, there can be no path in σ_t on which v is adjacent to some vertex $u \in V_t \setminus X_t$. It follows that v cannot be incident to a pseudo-edge, so we do not need to consider edges such that $f(uv) < 0$. Since $st(t)$ is valid and $\sigma_{t'}$ partitions $V_{t'}$ into monochromatic paths, it follows that σ_t is indeed a partition of V_t into monochromatic paths. In addition, our construction does not increase the number of

paths without a vertex in the current bag. It follows that the partition σ_t satisfies $st(t)$. □

The following lemma uses the relationship described in Lemma 6.36 to bound the time needed to compute the signature of an introduce node from the signature of its child.

Lemma 6.37. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be an introduce node in T, and let t' be the child of t. Suppose that the signature of t' is known. We can compute the signature of t in time $\mathcal{O}(cw^2(r+2)^{(w+1)^2})$.*

Proof. By Lemma 6.33, there are at most $c(r+2)^{(w+1)^2}$ valid states of X_t. It follows from Lemma 6.34 that we can obtain the set of valid states of X_t in time $\mathcal{O}(cw^2(r+2)^{(w+1)^2})$. For each valid state $st(t) = (f, k)$, it follows from Lemma 6.36 and the fact that X_t contains at most $(w+1)$ vertices that we can construct a valid state $st(t') = (f', k')$ of $X_{t'}$ such that $|S(st(t))| = |S(st(t'))|$ in time $\mathcal{O}(w^2)$. Hence, we can compute the signature of X_t in time $\mathcal{O}(cw^2(r+2)^{(w+1)^2})$. □

6.5.4 Forget Nodes

In this section, we describe how to compute the signature of a forget node from the signature of its child. We will first require some further definitions. Let t be a forget node in T, and let t' be the child of t in T. Let v denote the vertex forgotten at t, and let $st(t) = (f, k)$ be a valid state of X_t. We say that a partition $\sigma_{t'}$ of $V_{t'}$ into monochromatic paths satisfies $st(t)$ in X_t if the following hold:

- for each pair $(u, u') \in X_t^2$ we have that
  - if $f(uu') = 1$ then uu' is a true-edge of a monochromatic pseudo-path in $\sigma_{t'}$ in X_t
  - if $f(uu') = -i$ then uu' is a pseudo-edge of colour i on a monochromatic pseudo-path in $\sigma_{t'}$ in X_t
  - if $f(uu') = 0$ then uu' is not an edge on any path in $\sigma_{t'}$
- there are exactly k paths in $\sigma_{t'}$ which do not have any vertices in X_t

We shall use $S(t'|st(t))$ to denote the set of partitions of $V_{t'}$ into monochromatic paths which satisfy $st(t)$ in X_t. We will use $|S(t'|st(t))|$ to denote the number of such partitions. We use $ST(t'|st(t))$ to denote the set of valid states of $X_{t'}$ such that a partition of $V_{t'}$ into monochromatic paths satisfying some state $st(t') \in ST(t'|st(t))$ also satisfies $st(t)$ in X_t. We call the set $ST(t'|st(t))$ the set of *forget-inherited states of $X_{t'}$ given $st(t)$*.

In what follows, we describe the set of forget-inherited states of $X_{t'}$ given $st(t)$ exactly. We then show that the number of partitions of V_t into monochromatic paths satisfying $st(t)$ in X_t

can be obtained directly from the number of partitions of $V_{t'}$ satisfying each possible forget-inherited state of $X_{t'}$ given $st(t)$. The following lemma describes the set of forget-inherited states of $X_{t'}$ given $st(t)$ in the case where v is the only vertex on its path which is contained in $X_{t'}$.

Lemma 6.38. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be a forget node in T, and let t' be the child of t in T. Let v denote the vertex forgotten at t. Let $st(t) = (f, k)$ be a valid state of X_t, and let $st(t') = (f', k')$ be any valid state of $X_{t'}$ such that*

- *$k = k' + 1$, and*
- *$f'(uu') = f(uu')$ for all $(u, u') \in X_{t'}^2$ such that $u, u' \neq v$, and*
- *$f'(uv) = 0$ for all $u \in X_{t'}$ such that $u \neq v$, and*
- *$f'(vv) \leq 0$.*

We have that

$$S(st(t')) \subseteq S(t'|st(t)).$$

Proof. We must show that any partition of $V_{t'}$ into monochromatic paths satisfying $st(t')$ in $X_{t'}$ also satisfies $st(t)$ in X_t. Let $\sigma_{t'}$ be such a partition. Since $f'(uv) = 0$ for all $u \in X_{t'}$ such that $u \neq v$, it follows that v is the only vertex in $X_{t'}$ on its path in $\sigma_{t'}$. Hence, the number of paths in $\sigma_{t'}$ which do not have any vertices in X_t is equal to $(k' + 1)$. Since all other paths in $\sigma_{t'}$ form the same set of monochromatic pseudo-paths in $X_{t'}$ as in X_t, and we have that $f'(uu') = f(uu')$ for all $(u, u') \in X_t^2$, it follows that $\sigma_{t'}$ satisfies $st(t)$ in X_t. □

The next lemma describes the set of forget-inherited states of $X_{t'}$ given $st(t)$ in the case where v is not the only vertex on its path which is contained in $X_{t'}$.

Lemma 6.39. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be a forget node in T, and let t' be the child of t in T. Let v be the vertex forgotten at t. Let $st(t) = (f, k)$ be a valid state of X_t, and let $st(t') = (f', k')$ be any valid state of $X_{t'}$ such that*

- *$k = k'$, and*
- *there exists some pair $(z, z') \in X_{t'}^2$ with $z, z' \neq v$ and some $i \in [rn]$ such that*
  - *$f(zz') = -i$, and*
  - *$f'(zz') = 0$, and*
  - *$f'(zv) = 1$ and zv has colour i in $E(G)$, or $f'(zv) = -i$, and*

  - $f'(z'v) = 1$ *and* $z'v$ *has colour* i *in* $E(G)$*, or* $f'(z'v) = -i$*, and*
- $f'(uv) = 0$ *for all* $u \in X_{t'}$ *such that* $u \neq z, z'$*, and*
- $f'(uu') = f(uu')$ *for all pairs* $(u, u') \in X_t^2$ *such that* $\{u, u'\} \neq \{z, z'\}$.

We have that

$$S(st(t')) \subseteq S(t'|st(t)).$$

Proof. As for Lemma 6.39, we proceed by showing that any partition of $V_{t'}$ into monochromatic paths satisfying $st(t')$ in $X_{t'}$ also satisfies $st(t)$ in X_t. Let $\sigma_{t'}$ be such a partition. Since there exists some pair $(z, z') \in X_{t'}^2$ such that $f'(vz) \in \{1, -i\}$ and $f'(vz') \in \{1, -i\}$, it follows that v is not the only vertex in $X_{t'}$ on its path in $\sigma_{t'}$. In particular, v is adjacent to z and z' (where possibly $z = z'$) on a monochromatic pseudo-path of colour i in $X_{t'}$. It follows that the number of paths in $\sigma_{t'}$ which do not have any vertices in X_t is equal to k'. The removal of v from the monochromatic pseudo-path of colour i in $X_{t'}$ containing v creates a pseudo-edge zz' of colour i in X_t in both the case where $z = z'$ and the case where $z \neq z'$. It follows that $\sigma_{t'}$ satisfies the state $st(t) = (f, k)$ in X_t with $f(zz') = -i$. Since $f'(uv) = 0$ for all $u \neq z, z'$, it follows that $\sigma_{t'}$ satisfies the state $st(t) = (f, k)$ in X_t where $f(uu') = f'(uu')$ for all pairs $(u, u') \in X_t^2$ such that $\{u, u'\} \neq \{z, z'\}$. □

In the following lemma, we prove that the set of states of $X_{t'}$ described in Lemmas 6.38 and 6.39 form the set of forget-inherited states of $X_{t'}$ given $st(t)$.

Lemma 6.40. *Let* $G = (V(G), E(G))$ *be an* r*-locally edge-coloured graph with treewidth at most* w*, and let* $(T, \{X_t\}_{t \in V(T)})$ *be a nice tree decomposition of* G *with width at most* w*. Let* t *be a forget node in* T*, and let* t' *be the child of* t *in* T*. Let* v *be the vertex forgotten at* t*, and let* $st(t) = (f, k)$ *be a valid state of* X_t*. Let* $ST(t'|st(t))$ *be the set of states* $st(t') = (f', k')$ *of* $X_{t'}$ *such that either*

- $k = k' + 1$*, and*
- $f'(uu') = f(uu')$ *for all pairs* $(u, u') \in X_{t'}^2$ *such that* $u, u' \neq v$*, and*
- $f'(uv) = 0$ *for all* $u \in X_{t'}$ *such that* $u \neq v$*, and*
- $f'(vv) \leq 0$*,*

or else

- $k = k'$*, and*
- *there exists some pair* $(z, z') \in X_{t'}^2$ *and some* $i \in [rn]$ *such that* $f(zz') = -i$ *and* $f'(zz') = 0$ *and*
- $f'(zv) = 1$ *and* zv *has colour* i *in* $E(G)$*, or* $f'(zv) = -i$*, and*
- $f'(z'v) = 1$ *and* $z'v$ *has colour* i *in* $E(G)$*, or* $f'(z'v) = -i$*, and*

- $f'(uv) = 0$ *for all* $u \in X_{t'}$ *such that* $u \neq z, z'$*, and*
- $f'(uu') = f(uu')$ *for all pairs* $(u, u') \in X_t^2$ *such that* $\{u, u'\} \neq \{z, z'\}$.

We have that

$$S(t'|st(t)) = \bigcup_{st(t') \in ST(t'|st(t))} S(st(t')).$$

Proof. In what follows, we show that any partition of $V_{t'}$ into monochromatic paths satisfying $st(t)$ in X_t must satisfy a state of $X_{t'}$ contained in the set $ST(t'|st(t))$. Since each partition of $V_{t'}$ into monochromatic paths satisfies exactly one valid state of $X_{t'}$, the result follows. Let $\sigma_{t'}$ be a partition of $V_{t'}$ into monochromatic paths satisfying $st(t)$, and let $st(t') = (f', k')$ be the state satisfied by $\sigma_{t'}$ in $X_{t'}$.

Suppose first that v is a singleton in $\sigma_{t'}$. We must have that $f'(uv) = 0$ for all $u \in X_{t'}$. In addition, since $X_t = X_{t'} \setminus \{v\}$, we have that $f'(uu') = f(uu')$ for all pairs $(u, u') \in X_t^2$. Since v is a singleton in $X_{t'}$, the number of paths in $\sigma_{t'}$ without a vertex in X_t is one greater than the number of paths without a vertex in $X_{t'}$. That is, we have that $k = k' + 1$. It follows that $st(t') \in ST(t'|st(t))$.

Now suppose that v is not a singleton in $\sigma_{t'}$, but that v is the only vertex on its path in $\sigma_{t'}$ which is also contained in $X_{t'}$. Thus, v is an endpoint of a monochromatic pseudo-path of colour i in $X_{t'}$ for some $i \in [rn]$. It follows that $f'(vv) = -i$ and that $f'(uv) = 0$ for all $u \in X_{t'}$ such that $u \neq v$. In addition, we must have that $f'(uu') = f(uu')$ for all pairs $(u, u') \in X_t^2$. Finally, since $X_t = X_{t'} \setminus \{v\}$ and v is the only vertex on its path in $X_{t'}$, it follows that the number of paths in $\sigma_{t'}$ without a vertex in X_t is one greater than the number of paths without a vertex in $X_{t'}$. That is, we have that $k = k' + 1$. It follows that $st(t') \in ST(t'|st(t))$.

Now suppose that v is not the only vertex on its path in $\sigma_{t'}$ which is also contained in $X_{t'}$. Suppose first that v is an endpoint of a path p of colour i in $\sigma_{t'}$, and let z be the vertex next to v on the corresponding pseudo-path in $X_{t'}$. Since $X_t = X_{t'} \setminus \{v\}$, we have that $z \in X_t$. Hence, we must have that $f(zz) = -i$. If v is next to z on p then we have that $f'(zv) = 1$ and, since p is of colour i, the edge zv must be assigned the colour i in $E(G)$. Otherwise we have that $f'(zv) = -i$. It follows that $f'(zv) \in \{1, -i\}$ and $f'(uv) = 0$ for all $u \in X_{t'}$ such that $u \neq z$. In addition, we must have that $f(uu') = f'(uu')$ for all $(u, u') \in X_t^2$ such that $\{u, u'\} \neq \{z, z\}$. Finally, since v is not the only vertex on p in $\sigma_{t'}$ which is also contained in $X_{t'}$, the number of paths in $\sigma_{t'}$ without a vertex in X_t is equal to the number of paths without a vertex in $X_{t'}$. That is, we have that $k' = k$. It follows that $st(t') \in ST(t'|st(t))$.

Finally, suppose that v is not the only vertex on its path p of colour i in $\sigma_{t'}$ which is also contained in $X_{t'}$ and that v is also not an endpoint of the corresponding monochromatic pseudo-path p' in $X_{t'}$. Let z and z' be the vertices on either side of v on p'. Since $X_t =$

$X_{t'} \setminus \{v\}$, we have that $(z, z') \in X_t^2$ and, since v is not an endpoint of p', we have that $z \neq z'$. The edge zv on p' may be either a true-edge (in which case $f'(zv) = 1$ and zv has colour i in $E(G)$) or a pseudo-edge (in which case $f'(zv) = -i$). Similarly, edge $z'v$ on p' may be either a true-edge (in which case $f'(z'v) = 1$ and $z'v$ has colour i in $E(G)$) or a pseudo-edge (in which case $f'(z'v) = -i$). In addition, since v cannot be next to any other vertices on p', we have that $f'(uv) = 0$ for all $u \in X_{t'}$ such that $u \neq z, z'$. Since v lies between z and z' on the monochromatic pseudo-path p' in $X_{t'}$, but is not present in X_t, we have that $f(zz') = -i$ and $f'(zz') = 0$. In addition, we have that $f(uu') = f'(uu')$ for all pairs $(u, u') \in X_t^2$ such that $\{u, u'\} \neq \{z, z'\}$. Finally, since v is not the only vertex on its path in $\sigma_{t'}$ which is also contained in $X_{t'}$, the number of paths in $\sigma_{t'}$ without a vertex in X_t is equal to the number of paths without a vertex in $X_{t'}$. That is, we have that $k' = k$. It follows that $st(t') \in ST(t'|st(t))$. □

Given a valid state $st(t)$ of X_t, the following lemma bounds the time needed to obtain the set of forget-inherited states of $X_{t'}$ given $st(t)$.

Lemma 6.41. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be a forget node in T, and let t' be the child of t in T. Let v be the vertex forgotten at t. We can obtain the set $ST(t'|st(t))$ of forget-inherited states of $X_{t'}$ given $st(t)$ in time $\mathcal{O}(cw^2(r+2)^{(w+1)^2})$.*

Proof. By Lemma 6.33, there are at most $c(r+2)^{(w+1)^2}$ valid states of $X_{t'}$. Let $st(t') = (f', k')$ be a valid state of $X_{t'}$. To determine whether $st(t')$ is a forget-inherited state of $X_{t'}$ given $st(t)$, we must check whether $st(t')$ meets either of the conditions described in Lemma 6.40. We can check whether $k' = k - 1$ or $k' = k$ (or neither, in which case $st(t')$ is not a forget-inherited state) in time $\mathcal{O}(1)$. It remains to check whether the values of the functions f' and f meet the relevant set of conditions described in Lemma 6.40. Since there are at most $(w+1)^2$ pairs in each of $X_{t'}^2$ and X_t^2, it follows that this takes time $\mathcal{O}(w^2)$. The result follows. □

In the following lemma, we use the relationship given in Lemma 6.40 to obtain the number of partitions of V_t into monochromatic paths satisfying $st(t)$ from the number of partitions of $V_{t'}$ into monochromatic paths satisfying each forget-inherited state of $X_{t'}$ given $st(t)$. This will allow us to compute the signature of t from the signature of t'.

Lemma 6.42. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be a forget node in T, and let t' be the child of t in T. Let v be the vertex forgotten at t, and let $ST(t'|st(t))$ denote the set of forget-inherited states of $X_{t'}$ given $st(t)$. We have that*

$$|S(st(t))| = \sum_{st(t') \in ST(t'|st(t))} |S(st(t'))|.$$

Proof. Since t is a forget node, each vertex contained in V_t is also contained in $V_{t'}$. It follows that the number of partitions of V_t into monochromatic paths satisfying $st(t)$ in X_t is equal to the number of partitions of $V_{t'}$ into monochromatic paths which satisfy $st(t)$ in X_t. That is, we have that

$$|S(t'|st(t))| = |S(st(t))|.$$

By Lemma 6.40, we have that

$$S(t'|st(t)) = \bigcup_{st(t') \in ST(t'|st(t))} S(st(t')).$$

Since a partition can only satisfy one state, it follows that

$$|S(st(t))| = \sum_{st(t') \in ST(t'|st(t))} |S(st(t'))|.$$

□

Using the relationship described in Lemma 6.42, we can now bound the time needed to compute the signature of a forget node from the signature of its child.

Lemma 6.43. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with at most n vertices and treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be a forget node in T, and let t' be the child of t in T. Suppose that the signature of t' is known. We can compute the signature of t in time $\mathcal{O}(c^2 w^2 (r+2)^{2(w+1)^2} n \log n)$.*

Proof. Let $st(t) = (f, k)$ be a valid state of X_t. By Lemma 6.41, we can obtain the set $ST(t'|st(t))$ of forget-inherited states of $X_{t'}$ given $st(t)$ in time $\mathcal{O}(cw^2(r+2)^{(w+1)^2})$. By Lemma 6.42, we have that

$$|S(st(t))| = \sum_{st(t') \in ST(t'|st(t))} |S(st(t'))|. \tag{6.1}$$

By Lemma 6.33, there are at most $c(r+2)^{(w+1)^2}$ states in $ST(t'|st(t))$, so there are at most $c(r+2)^{(w+1)^2}$ values to sum in (6.1). For any state $st(t') \in ST(t'|st(t))$, there can be at most n^n partitions of $V_{t'}$ which satisfy $st(t')$. It follows that at most $n \log n$ bits are needed to represent the value of $|S(st(t'))|$ for each $st(t') \in ST(t'|st(t))$. Moreover, since the sum in (6.1) is equal to $|S(st(t))|$, its value is also at most n^n. Hence, at most $n \log n$ bits are needed to represent its value. It follows that computing the sum in (6.1) takes time $\mathcal{O}(c(r+2)^{(w+1)^2} n \log n)$. Hence, computing the number of partitions of V_t into monochromatic paths which satisfy a particular valid state $st(t)$ of t takes time

$$\mathcal{O}(c(r+2)^{(w+1)^2} n \log n). \tag{6.2}$$

By Lemma 6.33, there are at most $c(r+2)^{(w+1)^2}$ valid states of X_t. By Lemma 6.34, we can obtain the set of valid states of X_t in time

$$\mathcal{O}(cw^2(r+2)^{(w+1)^2}). \tag{6.3}$$

For each valid state $st(t)$ of X_t, it follows from Lemma 6.41 that we can obtain the set of forget-inherited states of $X_{t'}$ given $st(t)$ in time

$$\mathcal{O}(cw^2(r+2)^{(w+1)^2}). \tag{6.4}$$

Finally, it follows from (6.2), (6.3) and (6.4) that we can compute the signature of t in time

$$\begin{aligned}&\mathcal{O}(cw^2(r+2)^{(w+1)^2} + c(r+2)^{(w+1)^2} \times (cw^2(r+2)^{(w+1)^2} + c(r+2)^{(w+1)^2} n \log n)) \\ &= \mathcal{O}(c^2 w^2 (r+2)^{2(w+1)^2} n \log n).\end{aligned}$$

□

6.5.5 Join Nodes

In this section, we describe how to compute the signature of a join node from the signatures of its children. Let t be a join node in T, and let t_1 and t_2 be the children of t in T. Let $st(t) = (f, k)$ be a valid state of X_t. Let the set $ST(t_1, t_2|st(t))$ consist of all pairs $st(t_1) = (f_1, k_1)$ and $st(t_2) = (f_2, k_2)$ of valid states of X_{t_1} and X_{t_2} respectively such that, for each pair $(st(t_1), st(t_2)) \in ST(t_1, t_2|st(t))$, we have that

- $k = k_1 + k_2$ and,
- for each pair $(u, v) \in X_t^2$, we have that if $f(uv) \in \{0, 1\}$ then $f_1(uv) = f_2(uv) = f(uv)$; otherwise either $f_1(uv) = f(uv)$ and $f_2(uv) = 0$, or else $f_2(uv) = f(uv)$ and $f_1(uv) = 0$.

We call the set $ST(t_1, t_2|st(t))$ the set of *joinable pairs of states of* X_{t_1} *and* X_{t_2} *given* $st(t)$. In what follows, we show that each partition σ_t of V_t into monochromatic paths satisfying a valid state $st(t)$ of X_t is equal to the union of a partition σ_{t_1} of V_{t_1} and a partition σ_{t_2} of V_{t_2} which respectively satisfy a joinable pair of states $st(t_1)$ and $st(t_2)$ given $st(t)$. Figure 6.2 illustrates an example of a single monochromatic path contained in such a union. We will see that this allows us to determine the number of partitions of V_t which satisfy $st(t)$ directly from the number of partitions satisfying each pair of joinable pairs of states of X_{t_1} and X_{t_2} given $st(t)$.

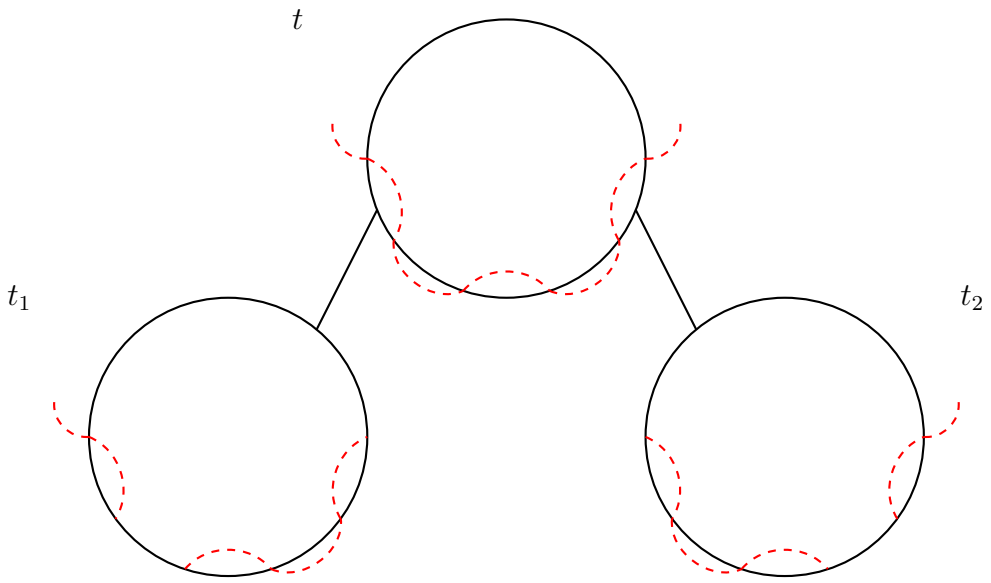

Figure 6.2: An example of a single monochromatic path in the union of a partition σ_{t_1} of V_{t_1} into monochromatic paths and a partition σ_{t_2} of V_{t_2} into monochromatic paths where σ_{t_1} and σ_{t_2} satisfy a joinable pair of states $st(t_1)$ and $st(t_2)$ of X_{t_1} and X_{t_2} respectively given some valid state $st(t)$ of X_t

Given a valid state $st(t)$ of X_t, the following two lemmas prove that the set of partitions of V_t into monochromatic paths satisfying $st(t)$ is equal to the set of unions formed from partitions of V_{t_1} and V_{t_2} which satisfy a joinable pair of states of X_{t_1} and X_{t_2} given $st(t)$.

Lemma 6.44. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be a join node in T, and let t_1 and t_2 be the children of t in T. Let $st(t) = (f, k)$ be a valid state of X_t. Let $st(t_1) = (f_1, k_1)$ and $st(t_2) = (f_2, k_2)$ be a joinable pair of states of X_1 and X_2 respectively given $st(t)$. Let σ_{t_1} (respectively σ_{t_2}) be a partition of V_{t_1} (respectively V_{t_2}) into monochromatic paths which satisfies $st(t_1)$ (respectively $st(t_2)$). The partition σ_t of V_t into monochromatic paths formed from the union of σ_{t_1} and σ_{t_2} satisfies $st(t)$.*

Proof. It follows from the definition of a joinable pair of states that if uv is a true-edge of a monochromatic pseudo-path of σ_{t_1} in X_{t_1} and of σ_{t_2} in X_{t_2} for some pair $(u, v) \in X_t^2$ then

we must have that $f(uv) = 1$. In addition, since $X_t = X_{t_1} = X_{t_2}$, if uv is a true-edge in X_{t_1} and in X_{t_2} then it is also a true-edge of a monochromatic pseudo-path of σ_t in X_t.

If uv is not an edge of any pseudo-path of σ_{t_1} in X_{t_1} or σ_{t_2} in X_{t_2} then by the definition of a joinable pair of states we must have that $f(uv) = 0$. In addition, uv is also not an edge of any pseudo-path of the union σ_t of σ_{t_1} and σ_{t_2}.

If uv is a pseudo-edge on a pseudo-path of colour i in exactly one of σ_{t_1} in X_{t_1} and σ_{t_2} in X_{t_2} then by the definition of a joinable pair of states we must have that $f(uv) = -i$. In addition, by construction, uv is a pseudo-edge on a pseudo-path of colour i in σ_t.

By the definition of a joinable pair of states, we must have that $k = (k_1 + k_2)$. Finally, since $V_t \setminus X_t = (V_{t_1} \setminus X_t) \cup (V_{t_2} \setminus X_t)$ and (by the properties of a tree decomposition) any path with vertices in both $V_{t_1} \setminus X_t$ and $V_{t_2} \setminus X_t$ must intersect X_t, we have that the number of paths in σ_t without a vertex in X_t is equal to $(k_1 + k_2)$. It follows that σ_t satisfies $st(t)$. □

In the following lemma, we show that any partition of V_t into monochromatic paths satisfying a valid state $st(t)$ of X_t is the union of a partition of V_{t_1} and a partition of V_{t_2} which satisfy a joinable pair of states of X_{t_1} and X_{t_2} respectively given $st(t)$.

Lemma 6.45. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be a join node in T, and let t_1 and t_2 be the children of t in T. Let $st(t) = (f, k)$ be a valid state of X_t, and let σ_t be a partition of V_t into monochromatic paths satisfying $st(t)$. There exists a joinable pair of states $st(t_1) = (f_1, k_1)$ and $st(t_2) = (f_2, k_2)$ of X_{t_1} and X_{t_2} respectively given $st(t)$ such that σ_t is the union of a partition σ_{t_1} of V_{t_1} into monochromatic paths satisfying $st(t_1)$ and a partition σ_{t_2} of V_{t_2} into monochromatic paths satisfying $st(t_2)$.*

Proof. We first construct the states $st(t_1)$ and $st(t_2)$ from $st(t)$. We will then construct partitions σ_{t_1} and σ_{t_2} of V_{t_1} and V_{t_2} respectively into monochromatic paths and prove that these partitions satisfy $st(t_1)$ and $st(t_2)$ respectively. For each $(u, v) \in X_t^2$ such that $f(uv) \in \{0, 1\}$, set $f_1(uv) = f_2(uv) = f(uv)$. If $f(uv) = -i$ for some $i \in [rn]$, then set $f_1(uv) = -i$ and $f_2(uv) = 0$ if the (u, v)-subpath of σ_t contains vertices in $V_{t_1} \setminus X_t$, and set $f_2(uv) = -i$ and $f_1(uv) = 0$ if the (u, v)-subpath of σ_t contains vertices in $V_{t_2} \setminus X_t$. Observe that $st(t_1)$ and $st(t_2)$ are a joinable pair of states of X_{t_1} and X_{t_2} given $st(t)$. The partitions σ_{t_1} and σ_{t_2} are formed from the partition induced by σ_t in each of V_{t_1} and V_{t_2}. That σ_{t_1} and σ_{t_2} satisfy $st(t_1)$, and $st(t_2)$ and that their union is equal to σ_t, follows directly from our construction. □

It follows from the previous two lemmas that the set of partitions of V_t into monochromatic paths satisfying $st(t)$ is equal to the set of unions of partitions of V_{t_1} and V_{t_2} respectively which satisfy a joinable pair of states of X_{t_1} and X_{t_2} given $st(t)$. We now show how this

relationship can be used to compute the number of partitions of V_t into monochromatic paths satisfying $st(t)$.

Lemma 6.46. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be a join node in T, and let t_1 and t_2 be the children of t in T. Let $st(t) = (f, k)$ be a valid state of X_t, and let $ST(t_1, t_2 | st(t))$ be the set of joinable pairs of states of X_{t_1} and X_{t_2} given $st(t)$. We have that*

$$|S(st(t))| = \sum_{(st(t_1), st(t_2)) \in ST(t_1, t_2 | st(t))} |S(st(t_1))| \times |S(st(t_2))|.$$

Proof. It follows from Lemmas 6.44 and 6.45 that the set of partitions of V_t into monochromatic paths satisfying $st(t)$ is equal to the set of unions of partitions of V_{t_1} and V_{t_2} which satisfy a pair $st(t_1)$ and $st(t_2)$ respectively such that $(st(t_1), st(t_2)) \in ST(t_1, t_2 | st(t))$. Since a partition satisfies exactly one state, it follows that the number of such unions is equal to

$$|S(st(t_1))| \times |S(st(t_2))|.$$

The result follows. □

In the following lemma, we bound the time needed to obtain the set of joinable pairs of states of X_{t_1} and X_{t_2} given $st(t)$.

Lemma 6.47. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be a join node in T, and let t_1 and t_2 be the children of t in T. Let $st(t) = (f, k)$ be a valid state of X_t. We can obtain the set $ST(t_1, t_2 | st(t))$ of joinable pairs of states of X_{t_1} and X_{t_2} given $st(t)$ in time $\mathcal{O}(c^2 w^2 (r+2)^{(w+1)^2})$.*

Proof. By Lemma 6.34, we can obtain the set of all valid states of X_{t_1} and X_{t_2} in time $\mathcal{O}(cw^2(r+2)^{(w+1)^2})$. Observe that a valid state $st(t_1)$ of X_{t_1} can form a joinable pair with at most one valid state $st(t_2)$ of X_{t_2}. Hence, it follows from Lemma 6.33 that there are at most $c(r+2)^{(w+1)^2}$ pairs of valid states of X_{t_1} and X_{t_2}. For each such pair $st(t_1) = (f_1, k_1)$ and $st(t_2) = (f_2, k_2)$, we must determine whether they form a joinable pair of states. We can determine whether $k = k_1 + k_2$ in time $\mathcal{O}(\log c)$. Since there are at most $(w+1)^2$ pairs in X_t^2, it follows that we can check whether the functions f_1 and f_2 meet the conditions required for a joinable pair in time $\mathcal{O}(w^2)$. It follows that we can obtain the set $ST(t_1, t_2 | st(t))$ in time

$$\begin{aligned} &\mathcal{O}(cw^2(r+2)^{(w+1)^2} + c(r+2)^{(w+1)^2} \times (w^2 + \log c)) \\ &= \mathcal{O}(c^2 w^2 (r+2)^{(w+1)^2}). \end{aligned}$$

□

We now use the observations made in Lemmas 6.46 and 6.47 to bound the time needed to compute the signature of a join node from the signatures of its children.

Lemma 6.48. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with at most n vertices and treewidth at most w, and let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. Let t be a join node in T, and let t_1 and t_2 be the children of t in T. Suppose that the signatures of t_1 and t_2 are known. We can compute the signature of t in time $\mathcal{O}(c^2 w^2 (r+2)^{2(w+1)^2} n^2 \log^2 n)$.*

Proof. Let $st(t) = (f, k)$ be a valid state of X_t. By Lemma 6.46 we have that

$$|S(st(t))| = \sum_{(st(t_1), st(t_2)) \in ST(t_1, t_2 | st(t))} |S(st(t_1))| \times |S(st(t_2))|. \tag{6.5}$$

The value of each of $|S(st(t_1))|$ and $|S(st(t_2))|$ is at most n^n for any $st(t_1)$ and $st(t_2)$. It follows that the value of each can be represented using at most $n \log n$ bits. Hence, for each pair $(st(t_1), st(t_2)) \in ST(t_1, t_2 | st(t))$, the product

$$|S(st(t_1))| \times |S(st(t_2))| \tag{6.6}$$

can be computed in time $\mathcal{O}(n^2 \log^2 n)$. It follows from the relationship in (6.5) that the product in (6.6) has value at most n^n. It follows from Lemma 6.33 and the definition of a joinable pair that the set $ST(t_1, t_2 | st(t))$ contains at most $c(r+2)^{(w+1)^2}$ pairs. By definition, the value of the sum in (6.5) is at most n^n. It follows that, given the set $ST(t_1, t_2 | st(t))$, we can compute the sum in (6.5) in time

$$\begin{aligned} &\mathcal{O}(c(r+2)^{(w+1)^2} \times (n^2 \log^2 n + n \log n)) \\ &= \mathcal{O}(c(r+2)^{(w+1)^2} n^2 \log^2 n). \end{aligned} \tag{6.7}$$

By Lemma 6.47, we can obtain the set $ST(t_1, t_2 | st(t))$ given $st(t)$ in time $\mathcal{O}(c^2 w^2 (r+2)^{(w+1)^2})$. It then follows from (6.7) that we can determine the value of $|S(st(t))|$ for any $st(t) \in ST(t)$ in time

$$\begin{aligned} &\mathcal{O}(c^2 w^2 (r+2)^{(w+1)^2} + c(r+2)^{(w+1)^2} n^2 \log^2 n) \\ &= \mathcal{O}(cw^2 (r+2)^{(w+1)^2} n^2 \log^2 n). \end{aligned}$$

By Lemma 6.34, we can obtain the set of valid states of X_t in time $\mathcal{O}(cw^2 (r+2)^{(w+1)^2})$. It follows from Lemma 6.33 that there are at most $c(r+2)^{(w+1)^2}$ such states. Hence, we can

compute the signature of t in time

$$\mathcal{O}(cw^2(r+2)^{(w+1)^2} + c(r+2)^{(w+1)^2} \times cw^2(r+2)^{(w+1)^2} n^2 \log^2 n)$$
$$= \mathcal{O}(c^2 w^2 (r+2)^{2(w+1)^2} n^2 \log^2 n).$$

□

6.5.6 Main Results

In this section, we describe our main result and its key corollaries. Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with at most n vertices and treewidth at most w, and let c be a positive integer. Let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G with width at most w. In the previous sections, we described how to compute the signature of each type of node in T from the signature(s) of its child(ren). We now use these relationships to describe an FPT dynamic programming algorithm for computing the number of partitions of $V(G)$ into at most c monochromatic paths parameterised by w and r.

Theorem 6.49. *Let $G = (V(G), E(G))$ be an r-locally edge-coloured graph with at most n vertices and treewidth at most w. We can count the number of ways to partition $V(G)$ into at most c monochromatic paths in time $\mathcal{O}(w^{\mathcal{O}(w^3)} n + c^2 w^2 (r+2)^{2(w+1)^2} n^3 \log^2 n)$.*

Proof. It follows from Theorems 6.30 and 6.31 that we can obtain a nice tree decomposition $(T, \{X_t\}_{t \in V(T)})$ of G with width at most w and $\mathcal{O}(n)$ nodes in time

$$\mathcal{O}(w^{\mathcal{O}(w^3)} n). \tag{6.8}$$

To compute the number of partitions of $V(G)$ into at most c monochromatic paths, we must first compute the signature $sig(r)$ of the root node r of T. Recall that each partition of V_r into (at most c) monochromatic paths satisfies exactly one valid state of X_r. It follows that we can then compute the number of partitions of $V(G)$ into at most c monochromatic paths by taking the sum, over all valid states $st(r)$ of X_r, of the number of partitions which satisfy $st(r)$. To compute the signature of r, we recursively compute the signature of every node in T starting at the leaves. By Lemma 6.35, we can compute the signature of a leaf node in time $\mathcal{O}(c)$. By Lemma 6.37, given the signature of its child, we can compute the signature of an introduce node in time $\mathcal{O}(cw^2(r+2)^{(w+1)^2})$. By Lemma 6.43, computing the signature of a forget node from the signature of its child takes time $\mathcal{O}(c^2 w^2 (r+2)^{2(w+1)^2} n \log n)$. By Lemma 6.48, given the signatures of its children, computing the signature of a join node takes time $\mathcal{O}(c^2 w^2 (r+2)^{2(w+1)^2} n^2 \log^2 n)$. Since there are $\mathcal{O}(n)$ nodes in $V(T)$, it follows

that we can compute the signature of the root node of T in time

$$\begin{aligned}&\mathcal{O}(n \times c^2 w^2 (r+2)^{2(w+1)^2} n^2 \log^2 n)\\ &= \mathcal{O}(c^2 w^2 (r+2)^{2(w+1)^2} n^3 \log^2 n).\end{aligned} \tag{6.9}$$

It remains to compute the sum

$$\sum_{st(r) \in ST(r)} |S(st(r))|. \tag{6.10}$$

By definition, the value of the sum in (6.10) is at most n^n and hence can be represented using at most $n \log n$ bits. By Lemma 6.33, there are at most $c(r+2)^{(w+1)^2}$ valid states of r. It follows that, once we have computed the signature of r, we can compute the sum in (6.10) in time

$$\mathcal{O}(c(r+2)^{(w+1)^2} n \log n). \tag{6.11}$$

Finally, it follows from (6.8), (6.9) and (6.11) that we can count the number of ways to partition $V(G)$ into at most c monochromatic paths in time

$$\begin{aligned}&\mathcal{O}(w^{\mathcal{O}(w^3)} n + c^2 w^2 (r+2)^{2(w+1)^2} n^3 \log^2 n + c(r+2)^{(w+1)^2} n \log n)\\ &= \mathcal{O}(w^{\mathcal{O}(w^3)} n + c^2 w^2 (r+2)^{2(w+1)^2} n^3 \log^2 n).\end{aligned}$$

□

Note that from the above method we can also determine the number of ways to partition $V(G)$ into at most - or exactly - k paths for any $k \leq c$ by taking the sum in (6.10) over only the set of states with the relevant number of paths. In addition, if we know the number of ways that an edge-coloured graph can be partitioned into c paths, then we also know whether such a partition exists. Since an r-edge-coloured graph is r-locally edge-coloured, our algorithm can also be used to count partitions of an r-edge-coloured graph into at most c monochromatic paths. Again, if we can count the number of such partitions, then we also know whether such a partition exists. Finally, let $G = (V(G), E(G))$ be an edge-coloured graph on n vertices. Each vertex in $V(G)$ is incident to edges of at most $n-1$ different colours. Thus, it follows from Theorem 6.49 that the problem of counting partitions of an edge-coloured graph into at most c monochromatic paths is in XP parameterised by the treewidth of the input graph. In addition, the problem of deciding if such a partition exists is in XP under the same parameterisation.

6.6 Remarks and Open Problems

In this chapter, we described an FPT algorithm for counting the number of partitions of an r-locally edge-coloured graph with treewidth at most w into c monochromatic paths parameterised by r and w. Our algorithm can also be used to solve the corresponding decision problem, as well as both the decision and counting variants on r-edge-coloured graphs. The same algorithm provides an XP solution parameterised by the treewidth of the input graph for both counting and decision when the input is an edge-coloured graph (with no restriction on the number of colours used). To the best of our knowledge, these are the first counting results for monochromatic partitioning problems.

A possible direction for further research is to consider whether our algorithm can be extended to count partitions of r-locally edge-coloured graphs, or even just r-edge-coloured graphs, into other classes of subgraphs. When counting partitions into monochromatic cycles, we note that a partial solution for a single bag might involve both monochromatic paths and cycles, while the full solution may contain only monochromatic cycles. To avoid counting (full) solutions containing monochromatic paths which are not cycles, we would need to ensure that we do not count partial solutions in which a monochromatic path has an endpoint "beneath" the bag. For this, it may be sufficient to ensure that we do not "forget" any vertex which is the endpoint of a monochromatic path. We would need also to change the definition of a state of a bag. Under the current definition, in the cycles setting there might be multiple pseudo-edges between a single pair of vertices which are on a cycle. There may also be both a true- and a pseudo-edge between a single pair of vertices. Nevertheless, it seems likely that with these and perhaps some other modifications to our algorithm, we can count partitions into monochromatic cycles. In the setting where we wish to count partitions into monochromatic trees, it may be sufficient to redefine a pseudo-edge as a tree (instead of a path) which intersects the bag on the specified vertices.

It would also be interesting to investigate whether our XP algorithm for counting partitions of edge-coloured graphs can be replaced by an FPT algorithm, or whether the problem is $\#\mathsf{W}[1]$-hard parameterised by treewidth and the number of colours alone. The bottleneck in our approach is the number of possible states of each bag in the tree decomposition. It might seem that since the number of vertices in each bag (and hence the neighbourhood of any vertex within the bag) is bounded by the treewidth w, we can bound the number of states of a bag in terms of some function of w. However, if a vertex v in the bag is the only vertex in the bag on its path, then v might be on a path of one of at most n possible colours beneath the bag. It follows that in order to obtain an FPT algorithm for this problem, we must find a way to record these kinds of intersections of a partition within a bag that does not depend exponentially upon the number of colours.

For any graph G, the vertex cover, pathwidth [135] and cutwidth [136] of G are each at least

the treewidth of the G. It follows that the problem of counting partitions of an r-locally edge-coloured graph into c monochromatic cycles is also in FPT parameterised by r and any one of these parameters. It is also possible that we might achieve fixed parameter tractability for the same problem parameterised by these parameters alone (without r). It would also be interesting to investigate whether we can obtain tractable parameterised algorithms for counting monochromatic partitions of edge-coloured graphs using some more general parameter than treewidth. A possible candidate is cliquewidth. The cliquewidth [32] of a graph is a similar graph parameter to treewidth. An FPT algorithm parameterised by cliquewidth would allow us to count monochromatic partitions of dense graphs, such as complete or complete bipartite graphs.

Chapter 7

Conclusion

In this thesis, we studied the parameterised complexity of counting subgraphs, stable matchings, and monochromatic partitions of edge-coloured graphs. In each setting, we described tractable algorithms for parameterised variants of these problems, or provided evidence suggesting that such algorithms are unlikely to exist.

In Chapter 3, we studied the subgraph counting problem in the setting where the host graph has a small number of vertices with high-degree. In Section 3.5, we gave examples of real-world networks with this structure, motivating the search for an efficient subgraph counting algorithm in this setting. In Section 3.6, we described an FPT algorithm for subgraph counting parameterised by the order of the pattern graph in host graphs with few high-degree vertices. We conjectured that our algorithm can be easily extended to efficiently count induced copies of subgraphs in the same setting.

In Chapter 4, we studied stable matching problems in the setting where agents' preferences can be described by a small number of types. In Section 4.5, we described an XP algorithm for counting stable matchings admitted by an instance of TYPED SMTI parameterised by the number of agent types. Section 4.6 extended this result to the stable roommates setting. We conjectured that the algorithm from Section 4.5 can also be easily extended to handle instances of hospitals/residents. It remains an open problem whether we can achieve fixed parameter tractability in any of these settings. In Section 4.7, we described algorithms for the problems of counting super-stable and strongly stable matchings in instances of stable matching problems with ties and incomplete preference lists. For TYPED SMTI and TYPED SRTI, we obtained parameterised algorithms for counting and finding super-stable matchings in time depending solely upon the number of agent types needed to describe an instance. For TYPED HRT, we described algorithms for finding and counting super-stable matchings in time depending upon the number of agent types and the natural logarithm of the number of agents. Under strong stability, we extended the XP result from Section 4.6 to count the number of strongly stable matchings admitted by an instance of TYPED SRTI.

In Chapter 5, we asked about the complexity of approximating the number of solutions to typed stable matching problems. In Section 5.5, we extended a well-known algorithm due to Arvind and Raman [21] for approximating the cardinality of the union of a collection of finite sets to allow for a weaker set of conditions on the sets. In Section 5.6, we used this result to obtain an efficient algorithm for approximating the number of solutions to an instance of TYPED SMTI. We conjectured that a straightforward extension of this algorithm can be used to approximate the number of stable matchings admitted by an instance of TYPED HRT. It is an open problem whether such an algorithm exists for approximating the number of solutions to an instance of TYPED SRTI. In Section 5.7, we considered a generalisation of TYPED SMTI in which individual agents may declare a constant number of their available partners as unacceptable. We conjectured that the algorithm from Section 5.6 can be extended to approximately count stable matchings in this setting, and provided evidence suggesting that exact counting in this setting is computationally hard.

Finally, in Chapter 6, we described an efficient algorithm for counting partitions of edge-coloured graphs with small treewidth into monochromatic paths. We believe that this is the first parameterised result in the area of monochromatic partitioning problems. We conjectured that variants of our approach can be used to efficiently count partitions of edge-coloured graphs into monochromatic cycles and trees.

Chapter 8

Bibliography

[1] B. Esfahbod, "Euler diagram for P, NP, NP-complete, and NP-hard set of problems," 2007. CC license: https://creativecommons.org/licenses/by-sa/3.0.

[2] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical review E*, vol. 70, p. (Art. 066111) 6, 2004.

[3] E. Bloedorn, N. J. Rothleder, D. DeBarr, and L. Rosen, "Relational Graph Analysis With Real-World Constraints: An Application in IRS Tax Fraud Detection," in *Proceedings of the AAAI-05 Workshop on Link Analysis*, pp. 30–39, The AAAI Press, 2005.

[4] E. Wong, B. Baur, S. Quader, and C.-H. Huang, "Biological network motif detection: principles and practice," *Briefings in Bioinformatics*, vol. 13, no. 2, pp. 202–215, 2012.

[5] L. G. Valiant, "The Complexity of Computing the Permanent," *Theoretical Computer Science*, vol. 8, no. 2, pp. 189–201, 1979.

[6] A. E. Roth, "The Evolution of the Labor Market for Medical Interns and Residents: a Case Study in Game Theory," *Journal of Political Economy*, vol. 92, no. 6, pp. 991–1016, 1984.

[7] M. Delorme, S. García, J. Gondzio, J. Kalcsics, D. Manlove, and W. Pettersson, "Mathematical models for stable matching problems with ties and incomplete lists," *European Journal of Operational Research*, vol. 277, no. 2, pp. 426–441, 2019.

[8] D. Gale and L. S. Shapley, "College Admissions and the Stability of Marriage," *American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[9] R. W. Irving, "An Efficient Algorithm for the "Stable Roommates" Problem," *Journal of Algorithms*, vol. 6, no. 4, pp. 577–595, 1985.

[10] D. F. Manlove, R. W. Irving, K. Iwama, S. Miyazaki, and Y. Morita, "Hard variants of stable marriage," *Theoretical Computer Science*, vol. 276, no. 1-2, pp. 261–279, 2002.

[11] K. Meeks and B. Rastegari, "Solving hard stable matching problems involving groups of similar agents," *Theoretical Computer Science*, vol. 844, pp. 171–194, 2020.

[12] R. W. Irving and P. Leather, "The Complexity of Counting Stable Marriages," *SIAM Journal on Computing*, vol. 15, no. 3, pp. 655–667, 1986.

[13] Z. Jin, M. Kano, X. Li, and B. Wei, "Partitioning 2-edge-colored complete multipartite graphs into monochromatic cycles, paths and trees," *Journal of Combinatorial Optimization*, vol. 11, no. 4, pp. 445–454, 2006.

[14] B. Bollobás, *Modern Graph Theory*, vol. 184 of *Graduate Texts in Mathematics*. Springer, 1998.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of* NP*-Completeness*. W. H. Freeman and Co., 1979.

[16] R. G. Downey and M. R. Fellows, *Fundamentals of Parameterized Complexity*. Texts in Computer Science, Springer, 2013.

[17] J. Flum and M. Grohe, *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series, Springer, 2006.

[18] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*. Springer, 2015.

[19] R. G. Downey and M. R. Fellows, *Parameterized Complexity*. Monographs in Computer Science, Springer, 1999.

[20] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[21] V. Arvind and V. Raman, "Approximation Algorithms for Some Parameterized Counting Problems," in *Algorithms and Computation*, vol. 2518 of *Lecture Notes in Computer Science*, pp. 453–464, Springer, 2002.

[22] L. G. Valiant, "The Complexity of Enumeration and Reliability Problems," *SIAM Journal on Computing*, vol. 8, no. 3, pp. 410–421, 1979.

[23] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu, "The ubiquity of large graphs and surprising challenges of graph processing: extended survey," *The VLDB Journal*, vol. 29, no. 2, pp. 595–618, 2020.

[24] S. Mangan and U. Alon, "Structure and function of the feed-forward loop network motif," *Proceedings of the National Academy of Sciences*, vol. 100, no. 21, pp. 11980–11985, 2003.

[25] B. Gelbord, "Graphical techniques in intrusion detection systems," in *Proceedings of the 15th International Conference on Information Networking*, pp. 253–258, IEEE, 2001.

[26] P. Bajardi, A. Barrat, F. Natale, L. Savini, and V. Colizza, "Dynamical Patterns of Cattle Trade Movements," *PLOS One*, vol. 6, no. 5, p. (Art. e19869) 19, 2011.

[27] R. Curticapean and D. Marx, "Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts," in *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pp. 130–139, IEEE, 2014.

[28] M. Grohe and N. Schweikardt, "First-Order Query Evaluation with Cardinality Conditions," in *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pp. 253–266, ACM, 2018.

[29] J. Flum and M. Grohe, "The parameterized complexity of counting problems," *SIAM Journal on Computing*, vol. 33, no. 4, pp. 892–922, 2004.

[30] R. G. Downey and M. R. Fellows, "Fixed-parameter tractability and completeness. II. On completeness for $\mathsf{W}[1]$," *Theoretical Computer Science*, vol. 141, no. 1-2, pp. 109–131, 1995.

[31] R. Curticapean, "Counting Matchings of Size k is #W[1]-hard," in *Proceedings of the International Colloquium on Automata, Languages, and Programming*, vol. 7965 of *Lecture Notes in Computer Science*, pp. 352–363, Springer, 2013.

[32] B. Courcelle, J. Engelfriet, and G. Rozenberg, "Handle-Hewriting Hypergraph Grammars," *Journal of Computer and System Sciences*, vol. 46, no. 2, pp. 218–270, 1993.

[33] J. Nešetřil and P. O. de Mendez, "On nowhere dense graphs," *European Journal of Combinatorics*, vol. 32, no. 4, pp. 600–617, 2011.

[34] J. Nešetřil and P. O. de Mendez, "The Grad of a Graph and Classes with Bounded Expansion," *Electronic Notes in Discrete Mathematics*, vol. 22, pp. 101–106, 2005.

[35] B. Courcelle, J. A. Makowsky, and U. Rotics, "On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic," *Discrete Applied Mathematics*, vol. 108, no. 1-2, pp. 23–52, 2001.

[36] N. Alon, R. Yuster, and U. Zwick, "Finding and Counting Given Length Cycles," *Algorithmica*, vol. 17, no. 3, pp. 209–223, 1997.

[37] R. Duan, H. Wu, and R. Zhou, "Faster Matrix Multiplication via Asymmetric Hashing," *arXiv:2210.10173*, (Preprint) 2022.

[38] V. Vassilevska Williams and R. Williams, "Finding, Minimizing, and Counting Weighted Subgraphs," *SIAM Journal on Computing*, vol. 42, no. 3, pp. 831–854, 2013.

[39] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto, "Counting Connected Subgraphs with Maximum-Degree-Aware Sieving," in *29th International Symposium on Algorithms and Computation*, vol. 123 of *Leibniz International Proceedings in Informatics*, pp. 17:1–12, Schloss Dagstuhl, 2018.

[40] H. Straubing, D. Thérien, and W. Thomas, "Regular Languages Defined with Generalized Quantifiers," *Information and Computation*, vol. 118, no. 2, pp. 289–301, 1995.

[41] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." `http://snap.stanford.edu/data`, June 2014.

[42] J. Leskovec, L. A. Adamic, and B. A. Huberman, "The dynamics of viral marketing," *ACM Transactions on the Web*, vol. 1, no. 1, p. (Art. 5) 39, 2007.

[43] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Predicting positive and negative links in online social networks," in *Proceedings of the 19th international conference on World wide web*, pp. 641–650, ACM, 2010.

[44] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Signed networks in social media," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1361–1370, ACM, 2010.

[45] R. Albert, H. Jeong, and A. L. Barabási, "Diameter of the World-Wide Web," *Nature*, vol. 401, no. 6749, pp. 130–131, 1999.

[46] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[47] A. Clauset, C. R. Shalizi, and M. E. J. Newman, "Power-Law Distributions in Empirical Data," *SIAM Review*, vol. 51, no. 4, pp. 661–703, 2009.

[48] L. A. Adamic and B. A. Huberman, "Power-Law Distribution of the World Wide Web," *Science*, vol. 287, no. 5461, pp. 2115–2115, 2000.

[49] S. Dill, R. Kumar, K. S. McCurley, S. Rajagopalan, D. Sivakumar, and A. Tomkins, "Self-similarity in the web," *ACM Transactions on Internet Technology*, vol. 2, no. 3, pp. 205–223, 2002.

[50] P. Wang, J. Zhao, X. Zhang, Z. Li, J. Cheng, J. C. Lui, D. Towsley, J. Tao, and X. Guan, "MOSS-5: A Fast Method of Approximating Counts of 5-Node Graphlets in Large Graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 1, pp. 73–86, 2018.

[51] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield, "Efficient Graphlet Counting for Large Networks," in *Proceedings of the 2015 IEEE International Conference on Data Mining*, pp. 1–10, IEEE, 2015.

[52] N. Pržulj, D. G. Corneil, and I. Jurisica, "Efficient estimation of graphlet frequency distributions in protein–protein interaction networks," *Bioinformatics*, vol. 22, no. 8, pp. 974–980, 2006.

[53] D. E. Knuth, *Stable Marriage and its Relation to Other Combinatorial Problems: An Introduction to the Mathematical Analysis of Algorithms*, vol. 10 of *CRM Proceedings and Lecture Notes*. American Mathematical Society, 1997. Translated from the French by Martin Goldstein and revised by the author.

[54] M. Lampis, "Algorithmic Meta-Theorems for Restrictions of Treewidth," *Algorithmica*, vol. 64, no. 1, pp. 19–37, 2012.

[55] D. Gusfield and R. W. Irving, *The Stable Marriage Problem: Structure and Algorithms*. Foundations of Computing Series, MIT Press, 1989.

[56] D. F. Manlove, *Algorithmics of matching under preferences*, vol. 2 of *Series on Theoretical Computer Science*. World Scientific, 2013.

[57] T. Feder, N. Megiddo, and S. A. Plotkin, "A Sublinear Parallel Algorithm for Stable Matching," *Theoretical Computer Science*, vol. 233, no. 1-2, pp. 297–308, 2000.

[58] R. W. Irving, "Stable marriage and indifference," *Discrete Applied Mathematics*, vol. 48, no. 3, pp. 261–272, 1994.

[59] E. Ronn, "NP-Complete Stable Matching Problems," *Journal of Algorithms*, vol. 11, no. 2, pp. 285–304, 1990.

[60] A. E. Roth, "On the Allocation of Residents to Rural Hospitals: a General Property of Two-Sided Matching Markets," *Econometrica*, vol. 54, no. 2, pp. 425–427, 1986.

[61] D. Gale and M. Sotomayor, "Some remarks on the stable matching problem," *Discrete Applied Mathematics*, vol. 11, no. 3, pp. 223–232, 1985.

[62] R. W. Irving, D. F. Manlove, and G. O'Malley, "Stable marriage with ties and bounded length preference lists," *Journal of Discrete Algorithms*, vol. 7, no. 2, pp. 213–219, 2009.

[63] D. F. Manlove, "Stable marriage with ties and unacceptable partners," *Technical Report no. TR-1999-29, Department of Computing Science, University of Glasgow*, 1999.

[64] R. W. Irving, D. F. Manlove, and S. Scott, "The Hospitals/Residents Problem with Ties," in *Algorithm Theory – SWAT 2000*, vol. 1851 of *Lecture Notes in Computer Science*, pp. 259–271, Springer, 2000.

[65] R. W. Irving, D. F. Manlove, and S. Scott, "Strong Stability in the Hospitals/Residents Problem," in *STACS 2003*, vol. 2607 of *Lecture Notes in Computer Science*, pp. 439–450, Springer, 2003.

[66] R. W. Irving and D. F. Manlove, "The Stable Roommates Problem with Ties," *Journal of Algorithms. Cognition, Informatics and Logic*, vol. 43, no. 1, pp. 85–105, 2002.

[67] S. Scott, *A Study of Stable Marriage Problems with Ties*. PhD thesis, University of Glasgow, 2005.

[68] A. Kunysz, "The Strongly Stable Roommates Problem," in *24th Annual European Symposium on Algorithms*, vol. 57 of *Leibniz International Proceedings in Informatics*, p. (Art. 60) 15, Schloss Dagstuhl, 2016.

[69] T. Kavitha, K. Mehlhorn, D. Michail, and K. E. Paluch, "Strongly stable matchings in time $O(nm)$ and extension to the hospitals-residents problem," *ACM Transactions on Algorithms*, vol. 3, no. 2, p. (Art. 15) 18, 2007.

[70] M. M. Halldórsson, K. Iwama, S. Miyazaki, and H. Yanagisawa, "Improved Approximation Results for the Stable Marriage Problem," *ACM Transactions on Algorithms*, vol. 3, no. 3, p. (Art. 30) 18, 2007.

[71] E. McDermid, "A $3/2$-Approximation Algorithm for General Stable Marriage," in *ICALP 2009: Automata, Languages and Programming*, vol. 5555 of *Lecture Notes in Computer Science*, pp. 689–700, Springer, 2009.

[72] Z. Király, "Linear Time Local Approximation Algorithm for Maximum Stable Marriage," *Algorithms*, vol. 6, no. 3, pp. 471–484, 2013.

[73] D. Adil, S. Gupta, S. Roy, S. Saurabh, and M. Zehavi, "Parameterized algorithms for stable matching with ties and incomplete lists," *Theoretical Computer Science*, vol. 723, pp. 1–10, 2018.

[74] N. Robertson and P. D. Seymour, "Graph minors. III. Planar tree-width," *Journal of Combinatorial Theory, Series B*, vol. 36, no. 1, pp. 49–64, 1984.

[75] D. Marx and I. Schlotter, "Parameterized Complexity and Local Search Approaches for the Stable Marriage Problem with Ties," *Algorithmica*, vol. 58, no. 1, pp. 170–187, 2010.

[76] R. Bredereck, K. Heeger, D. Knop, and R. Niedermeier, "Parameterized complexity of stable roommates with ties and incomplete lists through the lens of graph parameters," *Information and Computation*, vol. 289, Part A, p. (Art. 104943) 41, 2022.

[77] K. Kawarabayashi and B. Rossman, "A polynomial excluded-minor approximation of treedepth," *Journal of the European Mathematical Society*, vol. 24, no. 4, pp. 1449–1470, 2022.

[78] P. Wollan, "The structure of graphs not admitting a fixed immersion," *Journal of Combinatorial Theory, Series B*, vol. 110, pp. 47–66, 2015.

[79] S. Gupta, S. Saurabh, and M. Zehavi, "On Treewidth and Stable Marriage: Parameterized Algorithms and Hardness Results (Complete Characterization)," *SIAM Journal on Discrete Mathematics*, vol. 36, no. 1, pp. 596–681, 2022.

[80] R. W. Irving, D. F. Manlove, and S. Scott, "The Stable Marriage Problem with Master Preference Lists," *Discrete Applied Mathematics*, vol. 156, no. 15, pp. 2959–2977, 2008.

[81] P. Chebolu, L. A. Goldberg, and R. Martin, "The complexity of approximately counting stable matchings," *Theoretical Computer Science*, vol. 437, pp. 35–68, 2012.

[82] M. Dyer, L. A. Goldberg, C. Greenhill, and M. Jerrum, "The Relative Complexity of Approximate Counting Problems," *Algorithmica*, vol. 38, no. 3, pp. 471–500, 2004.

[83] N. Bhatnagar, S. Greenberg, and D. Randall, "Sampling Stable Marriages: Why Spouse-Swapping Won't Work," in *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1223–1232, ACM, 2008.

[84] P. Chebolu, L. A. Goldberg, and R. Martin, "The complexity of approximately counting stable roommate assignments," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1579–1605, 2012.

[85] A. I. Barvinok, "Two Algorithmic Results for the Travelling Salesman Problem," *Mathematics of Operations Research*, vol. 21, no. 1, pp. 65–84, 1996.

[86] T. Muir, "On a Class of Permanent Symmetric Functions," *Proceedings of the Royal Society of Edinburgh*, vol. 11, pp. 409–418, 1882.

[87] G. Meurant, *Computer Solution of Large Linear Systems*, vol. 28 of *Studies in Mathematics and its Applications*. Elsevier, 1999.

[88] R. Piziak and P. L. Odell, "Full Rank Factorization of Matrices," *Mathematics Magazine*, vol. 72, no. 3, pp. 193–201, 1999.

[89] P. B. Borwein, "On the Complexity of Calculating Factorials," *Journal of Algorithms*, vol. 6, no. 3, pp. 376–380, 1985.

[90] M. Jerrum, A. Sinclair, and E. Vigoda, "A Polynomial-Time Approximation Algorithm for the Permanent of a Matrix with Nonnegative Entries," *Journal of the ACM*, vol. 51, no. 4, pp. 671–697, 2004.

[91] A. Björklund, B. Gupt, and N. Quesada, "A faster hafnian formula for complex matrices and its benchmarking on a supercomputer," *ACM Journal of Experimental Algorithmics*, vol. 24, p. (Art. 1.11) 17, 2019.

[92] A. Barvinok, "Polynomial Time Algorithms to Approximate Permanents and Mixed Discriminants Within a Simply Exponential Factor," *Random Structures & Algorithms*, vol. 14, no. 1, pp. 29–61, 1999.

[93] R. M. Karp, M. Luby, and N. Madras, "Monte-Carlo Approximation Algorithms for Enumeration Problems," *Journal of Algorithms*, vol. 10, no. 3, pp. 429–448, 1989.

[94] E. Mendelson, *Introduction to mathematical logic*. Chapman & Hall, fourth ed., 1997.

[95] D. R. Karger, "A Randomized Fully Polynomial Time Approximation Scheme for the All-Terminal Network Reliability Problem," *SIAM Journal on Computing*, vol. 29, no. 2, pp. 492–514, 1999.

[96] Y. Liu, J. Chen, and J. Wang, "On counting 3-D matchings of size k," *Algorithmica*, vol. 54, no. 4, pp. 530–543, 2009.

[97] M. Jerrum and K. Meeks, "The parameterised complexity of counting connected subgraphs and graph motifs," *Journal of Computer and System Sciences*, vol. 81, no. 4, pp. 702–716, 2015.

[98] Y. Liu, S. Wang, and J. Wang, "Parameterized counting matching and packing: a family of hard problems that admit FPTRAS," *Theoretical Computer Science*, vol. 734, pp. 83–93, 2018.

[99] M. Fellows, D. Hermelin, and F. Rosamond, "On the Fixed-Parameter Intractability and Tractability of Multiple-Interval Graph Properties," *Theoretical Computer Science*, vol. 410, pp. 53–61, 2009.

[100] D. Štefankovič, E. Vigoda, and J. Wilmes, "On Counting Perfect Matchings in General Graphs," in *LATIN 2018: Theoretical Informatics*, vol. 10807 of *Lecture Notes in Computer Science*, pp. 873–885, Springer, 2018.

[101] T. Luczak, V. Rödl, and E. Szemerédi, "Partitioning Two-Coloured Complete Graphs into Two Monochromatic Cycles," *Combinatorics, Probability and Computing*, vol. 7, no. 4, pp. 423–436, 1998.

[102] P. Allen, "Covering two-edge-coloured complete graphs with two disjoint monochromatic cycles," *Combinatorics, Probability and Computing*, vol. 17, no. 4, pp. 471–486, 2008.

[103] S. Bessy and S. Thomassé, "Partitioning a graph into a cycle and an anticycle, a proof of Lehel's conjecture," *Journal of Combinatorial Theory, Series B*, vol. 100, no. 2, pp. 176–180, 2010.

[104] R. J. Duffin, "Topology of Series-Parallel Networks," *Journal of Mathematical Analysis and Applications*, vol. 10, no. 2, pp. 303–318, 1965.

[105] T. Kloks, *Treewidth: Computations and Approximations*, vol. 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[106] A. Gyárfás, "Vertex covers by monochromatic pieces - A survey of results and problems," *Discrete Mathematics*, vol. 339, no. 7, pp. 1970–1977, 2016.

[107] L. Gerencsér and A. Gyárfás, "On Ramsey-Type Problems," *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae. Sectio Mathematica*, vol. 10, pp. 167–170, 1967.

[108] A. Gyárfás, "Covering Complete Graphs by Monochromatic Paths," in *Irregularities of partitions*, vol. 8 of *Algorithms and Combinatorics*, pp. 89–91, Springer, 1989.

[109] A. Pokrovskiy, "Partitioning edge-coloured complete graphs into monochromatic cycles and paths," *Journal of Combinatorial Theory, Series B*, vol. 106, pp. 70–97, 2014.

[110] P. Erdős, A. Gyárfás, and L. Pyber, "Vertex coverings by Monochromatic Cycles and Trees," *Journal of Combinatorial Theory, Series B*, vol. 51, no. 1, pp. 90–95, 1991.

[111] S. Letzter, "Monochromatic cycle partitions of 3-coloured complete graphs." *In preparation*, 2018.

[112] D. Korándi, R. Lang, S. Letzter, and A. Pokrovskiy, "Minimum degree conditions for monochromatic cycle partitioning," *Journal of Combinatorial Theory, Series B*, vol. 146, pp. 96–123, 2021.

[113] R. Lang and A. Lo, "Monochromatic cycle partitions in random graphs," *Combinatorics, Probability and Computing*, vol. 30, no. 1, pp. 136–152, 2021.

[114] A. Gyárfás, M. Ruszinkó, G. Sárközy, and E. Szemerédi, "An improved bound for the monochromatic cycle partition number," *Journal of Combinatorial Theory, Series B*, vol. 96, no. 6, pp. 855–873, 2006.

[115] Z. Tuza, "Some special cases of Ryser's conjecture," 1979. *unpublished manuscript*.

[116] P. E. Haxell and Y. Kohayakawa, "Partitioning by Monochromatic Trees," *Journal of Combinatorial Theory, Series B*, vol. 68, no. 2, pp. 218–222, 1996.

[117] P. Haxell, "Partitioning Complete Bipartite Graphs by Monochromatic Cycles," *Journal of Combinatorial Theory, Series B*, vol. 69, no. 2, pp. 210–218, 1997.

[118] R. Lang, O. Schaudt, and M. Stein, "Partitioning 3-edge-coloured complete bipartite graphs into monochromatic cycles," *Electronic Notes in Discrete Mathematics*, vol. 49, pp. 787–794, 2015.

[119] Y. Peng, V. Rödl, and A. Ruciński, "Holes in Graphs," *Electronic Journal of Combinatorics*, vol. 9, no. 1, p. (Art. 1) 18, 2002.

[120] G. Chen, S. Fujita, A. Gyárfás, J. Lehel, and A. Tóth, "Around a biclique cover conjecture," *arXiv:1212.6861*, (Preprint) 2012.

[121] J. Balogh, J. Barát, D. Gerbner, A. Gyárfás, and G. N. Sárközy, "Partitioning 2-edge-colored graphs by monochromatic paths and cycles," *Combinatorica*, vol. 34, no. 5, pp. 507–526, 2014.

[122] A. Pokrovskiy, "Partitioning a graph into a cycle and a sparse graph," *Discrete Mathematics*, vol. 346, no. 1, p. (Art. 113161) 21, 2023.

[123] P. Allen, J. Böttcher, R. Lang, J. Skokan, and M. Stein, "Partitioning a 2-edge-coloured graph of minimum degree $2n/3 + o(n)$ into three monochromatic cycles," *arXiv:2204.00496*, (Preprint) 2022.

[124] G. Sárközy, "Monochromatic cycle partitions of edge-colored graphs," *Journal of Graph Theory*, vol. 66, no. 1, pp. 57–64, 2011.

[125] D. Conlon and M. Stein, "Monochromatic cycle partitions in local edge colorings," *Journal of Graph Theory*, vol. 81, no. 2, pp. 134–145, 2016.

[126] R. Lang and M. Stein, "Local colourings and monochromatic partitions in complete bipartite graphs," *European Journal of Combinatorics*, vol. 60, pp. 42–54, 2017.

[127] G. N. Sárközy, "Monochromatic Partitions in Local Edge Colorings," *Acta Mathematica Hungarica*, vol. 161, no. 2, pp. 412–426, 2020.

[128] Z. Jin and X. Li, "The complexity for partitioning graphs by monochromatic trees, cycles and paths," *International Journal of Computer Mathematics*, vol. 81, no. 11, pp. 1357–1362, 2004.

[129] Z. Jin and X. Li, "Vertex partitions of r-edge-colored graphs," *Applied Mathematics - A Journal of Chinese Universities*, vol. 23, pp. 120–126, 2008.

[130] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. Computer Science and Applied Mathematics, Academic Press, 1980.

[131] B. Courcelle and J. Engelfriet, *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Encyclopedia of Mathematics and its Applications, Cambridge University Press, 2012.

[132] J. Kneis and A. Langer, "A Practical Approach to Courcelle's Theorem," *Electronic Notes in Theoretical Computer Science*, vol. 251, pp. 65–81, 2009.

[133] H. L. Bodlaender, "A linear time algorithm for finding tree-decompositions of small treewidth," *SIAM Journal on Computing*, vol. 25, no. 6, pp. 1305–1317, 1996.

[134] H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof, "Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth," *Information and Computation*, vol. 243, pp. 86–111, 2015.

[135] N. Robertson and P. D. Seymour, "Graph minors. I. Excluding a forest," *Journal of Combinatorial Theory, Series B*, vol. 35, no. 1, pp. 39–61, 1983.

[136] F. R. K. Chung, "On the Cutwidth and the Topological Bandwidth of a Tree," *SIAM Journal on Algebraic Discrete Methods*, vol. 6, no. 2, pp. 268–277, 1985.