FAIR AND LARGE STABLE MATCHINGS IN THE STABLE MARRIAGE AND STUDENT-PROJECT ALLOCATION PROBLEMS

FRANCES COOPER

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

SEPTEMBER 1, 2020

Abstract

In this thesis, we present new algorithmic and complexity results for specific matching problems involving preferences. In particular we study the Stable Marriage problem (SM) and the Student-Project Allocation problem (SPA) and their variants. A *matching* in these scenarios is an allocation of men to women (SM) or students to projects (SPA). Primarily we are interested in finding matchings that are *stable*. A *stable matching* is a matching that admits no *blocking pair*, which is a pair of agents (not already allocated together) who would rather deviate from the given matching and become assigned to each other. In addition to stability, other objectives may be applied. We focus on finding either fair or large stable matchings in SM and SPA.

In the Stable Marriage problem with Incomplete lists (SMI), the *rank* of a matched man or woman, with respect to a matching, is the position of their assigned partner on their preference list. The *degree* of the set of men in a matching is the rank of a worst-off man, over all matched men. A similar definition exists for the set of women. The *cost* of the set of men in a matching is sum of ranks of all matched men. Again a similar definition exists for the set of women. We introduce the following degree-based definitions of fairness in SMI. A stable matching is *regret-equal* if it minimises the difference in degree between the set of men and the set of women, over all stable matchings. Additionally, a stable matching is *min-regret sum* if it minimises the sum of the degree of men and the degree of women, over all stable matchings. We present polynomial-time algorithms to find these types of fair stable matchings, given an instance of SMI, and perform experiments to both test the performance of two algorithms to find a regret-equal stable matching, and to compare properties of several other types of fair stable matchings over both degree- and cost-based measures.

Also in SMI, we investigate fairness in the form of *profile-based* stable matchings. The *profile* of a matching is a vector of integers, where the ith vector element indicates the number of agents assigned to their ith-choice partner. A stable matching is *rank-maximal* if its profile is lexicographically maximum taken over all stable matchings. A stable matching is *generous* if its reverse profile is lexicographically minimum taken over all stable matchings. A polynomial-time algorithm exists to find a rank-maximal stable matching, using weights that are exponential in the number of men or women [32]. We adapt this algorithm to work with

polynomially-bounded weight vectors, and show using randomly-generated instances that our approach is significantly less costly in terms of space. Further experiments are carried out to compare these profile-based optimal matchings over several cost- and profile-based fairness properties. We additionally show that in the Stable Roommates problem, each of the problems of finding a rank-maximal stable matching or a generous stable matching is NP-hard.

In the Student-Project Allocation problem with lecturer preferences over Students including Ties (SPA-ST) we study the problem of finding large stable matchings. A $\frac{3}{2}$-approximation algorithm exists to find a maximum-sized stable matching in HRT, and we extend this to the SPA-ST case, developing a linear-time $\frac{3}{2}$-approximation algorithm for the problem of finding a maximum-sized stable matching in SPA-ST. We test the performance of our approximation algorithm using the implementation of a new Integer Programming (IP) model that finds a maximum-sized stable matching, and show that in practice, our approximation algorithm produces stable matchings of size far closer to optimal than the $\frac{3}{2}$ bound. Additionally, we give an example to show that this bound is tight.

Finally, we look at fairness in the context of the Student-Project Allocation problem with lecturer preferences over Students including Ties and Lecturer targets (SPA-STL), an extension to SPA-ST in which each lecturer has an associated target, indicating their preferred number of allocations (or the number of allocations preferred by the matching scheme administrator). We first investigate load balancing without the presence of stability. A *load-max-balanced* matching is a matching in which the maximum difference between a lecturer's target and their number of allocations is minimised. A *load-sum-balanced* matching is a matching in which the sum of differences between all lecturers' targets and their number of allocations is minimised. Finally, a *load-balanced* matching is a matching that is both load-max-balanced and load-sum-balanced. We provide new polynomial-time algorithms to find matchings of these types. Additionally we show that in the presence of stability, each of the problems of finding a stable matching that is load-max-balanced, load-sum-balanced or load-balanced is NP-hard. Finally, we present new IP models for finding such types of optimal stable matching.

Acknowledgements

I would like to thank my supervisor, Prof. David Manlove for his fantastic support and guidance throughout the duration of my PhD. Thank you also to my second supervisor Dr. Kitty Meeks who was always on hand to offer advice, and to my examiners Dr. Jess Enright and Prof. Prudence Wong for their detailed feedback on my thesis.

A massive thank you to all the people who I met during my PhD and who have made my time at Glasgow unforgettable. Finally, thank you to my family for their unwaivering support, to Chris for keeping me sane and to Gözel for keeping me mad.

Publications

- F. Cooper and D.F. Manlove. A 3/2-Approximation Algorithm for the Student-Project Allocation Problem. In *Proceedings of SEA 2018: the 17th International Symposium on Experimental Algorithms, volume 103 of Leibniz International Proceedings in Informatics (LIPIcs)*, article 8 pages 1-13, 2018. Available at `https://doi.org/10.4230/LIPIcs.SEA.2018.8`. This article highlights the main algorithmic and experimental results from Chapter 5 and the first half of Chapter 6. The full version, which additionally includes all proofs relating to the above, is available as Technical Report number 1804.02731, ArXiv, 2018. Available at `https://arxiv.org/abs/1804.02731`.

- F. Cooper and D.F. Manlove. Algorithms for new types of fair stable matchings. In *Proceedings of SEA 2020: the 18th International Symposium on Experimental Algorithms, volume 160 of Leibniz International Proceedings in Informatics (LIPIcs)*, article 20 pages 1-13, 2020. Avaliable at `https://doi.org/10.4230/LIPIcs.SEA.2020.20`. This article presents the main algorithmic and experimental results from Chapter 3 except Algorithm RESP. The full version, which additionally includes all proofs relating to the above, is avaliable as Technical Report number 2001.10875, ArXiv, 2020. Available at `https://arxiv.org/abs/2001.10875`.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format.

Frances Cooper

Table of Contents

List of Tables

List of Figures

List of Algorithms

Acronyms

$(3,3)$-COM-SMTI The problem of finding a maximum stable matching in SMTI, such that each man's preference list is strictly ordered and of length 3, and, each woman's preference list is either strictly ordered and of length 3, or is a tie of length 2. 176

CHA Capacitated House Allocation problem. 7

CHAT Capacitated House Allocation problem with Ties. 7

GENSR The problem of finding a generous stable matching in SR. 81

HA House Allocation problem. 4, 6, 7, 127

HAT House Allocation problem with Ties. 7

HR Hospitals/Residents problem. 1–4, 6, 9, 18–20, 57

HR-GR Hospitals/Residents problem with Grouped Residents. 57, 58

HRC Hospitals/Residents problem with Couples. 19

HRT Hospitals/Residents problem with Ties. 2, 3, 5, 19, 22, 95–98, 100, 101, 103, 110, 182, 203

HRTH Hospitals/Residents problem with Ties and Hospital targets. 182

LBS-SPA-STL The problem of finding a load-balanced stable matching in SPA-STL. 175

LMBS-SPA-STL The problem of finding a load-max-balanced stable matching in SPA-STL. 175

LSBS-SPA-STL The problem of finding a load-sum-balanced stable matching in SPA-STL. 175

MAX-HRT The problem of finding a maximum stable matching in HRT. 2, 5, 19, 95, 96

MAX-SMTI The problem of finding a maximum stable matching in SMTI. 16, 22

Chapter 1

Introduction

Matching problems arise when it is necessary to assign one or more sets of entities to each other, based on preferences on one or both sides. Practical applications include the assignment of kidney donors to patients, graduating medical students to hospitals, or applicants to campus housing accommodation.

Manlove [47] describes a decentralised allocation mechanism as a process whereby the agents involved are able to negotiate with one another directly to form allocations. These free-for-all market scenarios often lead to several problems [70], such as entities forming assignments with one another far earlier than the required deadline in order to secure an assignment, potentially leading to less informed choices. Centralised matching schemes, in which all preferences are collected by a third party and dealt with simultaneously by some automated mechanism, avoid many of these problems. Such matching schemes arise in a number of applications, including those mentioned above, and additionally in the assignment of high-school students to university places in China [75]. This scheme requires all students to sit National College Entrance Exams, the results of which are used, along with student preferences, in the allocation process. Another example is a mechanism used in Boston to assign students to schools based on parent preferences over schools and priorities for children at schools [2]. It is clear from these examples that matching problem instances may be extremely large, and as such require efficient algorithms to find suitable matchings.

In the application of assigning a set of graduating medical students to a set of hospitals, both sets may have preferences over the other. Medical students may have preferences over hospitals depending on hospital reputation or recommendation from other students, and hospitals may have preferences over students due to grades achieved in their studies. This case, where there are two disjoint sets of entities, both of which have preferences over the other, can be modelled by the *Hospitals/Residents problem* (HR). An instance of HR comprises resident and hospital agents with each resident ranking a subset of the hospitals (the resident's *acceptable hospitals*) in order of preference and vice versa. A *matching* in an HR instance may

be defined as an assignment of residents to hospitals such that each resident is assigned to at most one hospital on their preference list and each hospital is assigned zero or more residents on their preference list up to a pre-determined capacity. A *blocking pair* in an HR matching is defined as a resident-hospital pair who prefer each other to at least one of their assigned partners (if any). A matching is considered *stable* if it admits no blocking pair. As Peranson and Randlett [65] discuss, the *National Resident Matching Program* is a long-standing matching scheme in the US (estabished in 1952) which assigns graduating medical students to hospitals, with the aim of producing a stable matching in the underlying HR instance. Stability gives the assurance that there is no graduating medical student-hospital pair who would have incentive to break their current assignments and become assigned to each other. This is a highly important criteria in matching problems, as it leads to an allocation that is robust against deviations of this form [67].

There may be many different stable matchings in an instance of HR, and which matching is best will depend on the priorities of the matching scheme administrator. In order to discuss various optimality criteria that an administrator may choose, we first define two measures associated with a matching in HR. The *rank* of an assigned resident, with respect to a matching, is the position of their assigned hospital on their preference list. The *profile* of a matching is a vector that indicates the number of agents with an assignment to their first-choice partner, the number of agents with an assignment to their second-choice partner, and so on. Note that since hospitals may be multiply assigned, they may contribute more than once to the profile. Common optimality criteria in HR then include maximising the number of assigned residents, minimising the sum of ranks over all assigned residents (possibly with a penalty for unassigned residents), and various optimality criteria associated with the profile of a matching, among the set of all stable matchings.

In the case of assigning graduating medical students to hospitals, hospital preference lists may be very large. In such cases, it is not reasonable to expect hospitals to be able to rank all students in strict preference order. This motivates the study of an extention to HR in which indifference may be allowed in preference lists. The Hospitals/Residents problem with Ties (HRT) is a generalisation of HR in which hospitals may have ties in their preference lists indicating indifference between two or more residents, and vice versa. In HRT, stable matchings may be of different sizes, and so this naturally leads to MAX-HRT, the problem of finding a stable matching of maximum size (also known as a *maximum stable matching*). MAX-HRT is NP-hard [51].

For NP-hard problems such as MAX-HRT, no efficient (polynomial-time) algorithm exists unless $\mathsf{P} = \mathsf{NP}$. In such cases, one possibility is to settle for an approximation algorithm. Approximation algorithms find a solution that need not be optimal, but is often close to optimal in a precise sense, in polynomial time. The *approximation factor* of an approximation algorithm A is a fraction indicating a guarantee on how close to optimal the algorithm achieves

in the worst case. Specifically, for a maximisation (respectively minimisation) problem, this is the maximum ratio of the measure of an optimal (respectively approximate, produced by A) solution to the measure of an approximate, produced by A (respectively optimal solution), taken over all problem instances. A c-approximation algorithm is then an approximation algorithm with approximation factor c. Király [40] developed a $\frac{3}{2}$-approximation algorithm for the NP-hard problem of finding a maximum stable matching in an instance of HRT, which will produce a stable matching that is at least two-thirds of the size of the maximum stable matching. In some cases, however, we either require an optimal solution, or the NP-hard problem in question cannot be approximated to a reasonable degree in polynomial time. In such cases, an alternative is to formulate the problem using an *Integer Programming* (IP) model. IP solvers use algorithms that run in exponential time in the worst case. Typically, the more complex the problem is, the longer it will take to solve. However, IP solvers are often able to find optimal solutions reasonably quickly, especially for less complex instances.

This thesis focusses on two types of matching problem. One is a special case of HR, and the other a generalisation. A description of these matching problems now follows.

A special case of HR, known as the Stable Marriage problem with Incomplete lists (SMI) comprises men and women as the agents, in which men take the place of residents and women take the place of hospitals. The term 'Incomplete lists' refers to the fact that men need only rank a subset of women in preference order, and vice versa. Each man or woman may only be assigned a single partner in any matching. Analogous definitions of a *stable matching* and the *profile* of a matching apply to SMI as in HR. The *rank* of an assigned man (woman) is then the position of his (her) assigned partner on his (her) preference list. The *degree* of a matching is given by the largest rank over all assigned men and women. Finally, the *cost* of a matching is given by the sum of the ranks of all assigned men and women.

A generalised version of HR is known as the *Student-Project Allocation problem with lecturer preferences over Students* (SPA-S). Here, we have a set of students, a set of projects and a set of lecturers. Each project may only be supervised by a single lecturer but lecturers may supervise multiple projects. Students have preferences over projects and lecturers have preferences over students who rank at least one of their offered projects. Both projects and lecturers have upper quotas associated with them indicating the maximum number of allocations they are allowed. In this way, HR can be seen as a special case of SPA-S in which each lecturer supervises a single project, with which they also share an identical capacity. An extension to SPA-S, that allows ties in preference lists in an analogous way to HRT, is known as SPA-ST. A *stable matching* in SPA-ST is a matching in which no student-lecturer pair has reason to deviate from their allocations and assign to each other. SPA-ST is a generalisation of HRT, and therefore has the same characteristics that stable matchings may be different sizes. We define the problem of finding a maximum-sized stable matching (hereafter *maximum stable matching*) as MAX-SPA-ST. MAX-SPA-ST is NP-hard [51]. A further extension is

known as SPA-STL (SPA-ST with lecturer targets, which indicate the preferred number of allocations for each lecturer).

In this thesis, we concentrate on finding stable matchings with additional optimality criteria in both SMI and SPA-S. In SMI we examine the problem of finding "fair" stable matchings, that in some way balance the interests of both men and women. Several notions of fairness already exist in this setting involving the cost, degree and profile of a stable matching. We present new definitions of fairness in SMI, associated with the degree of a matching, and develop efficient algorithms to find such matchings. Additionally, we adapt an existing algorithm to find a profile-based fair stable matching, to work with a far reduced memory load. Fairness is also examined with respect to lecturer loads in SPA-STL. In this scenario, we study matchings that in some way balance lecturer allocation loads, with respect to their targets, and present new definitions of load balancing in this context. We develop efficient algorithms to find such matchings when stability is ignored, and show that when stability is required, such matchings are NP-hard to find. In addition to fairness we study the problem of finding large stable matchings, presenting a $\frac{3}{2}$-approximation algorithm for MAX-SPA-ST.

The remainder of this thesis is structured as follows.

- *Chapter 2: Literature review*. Chapter 2 describes several matching problems in more detail including SMI, HR, SPA-S, and their extensions. Additionally, we describe the *House Allocation problem* (HA), a special case of HR in which hospitals do not have preferences over residents, and the *Stable Roommates problem* (SR), a generalisation of SMI in which there is only one set of agents, each of whom has preferences over the other agents. Relevant literature is also surveyed.

- *Chapter 3: Degree-based stable matchings in* SMI. In Chapter 3 we explore fair stable matchings in SMI. We introduce two new notions of fairness involving the degree of a matching. Firstly, a *regret-equal stable matching* minimises the difference in the ranks between a worst-off man and a worst-off woman, among all stable matchings. Secondly, a *min-regret sum stable matching* minimises the sum of the ranks of a worst-off man and a worst-off woman, among all stable matchings. In this chapter, we present new efficient algorithms to find stable matchings of these types. Experiments to compare several types of fair optimal stable matchings were conducted and show that our new algorithm to find a regret-equal stable matching produces matchings that are competitive with respect to other existing fairness objectives. On the other hand, existing types of fair stable matchings did not provide as close an approximation to regret-equal stable matchings.

- *Chapter 4: Profile-based stable matchings in* SMI. Continuing the theme of fairness, Chapter 4 focusses on profile-based optimality in SMI. We study *rank-maximal* and

generous stable matchings. A stable matching is *rank-maximal* if the maximum number of men and women are assigned to their first-choice partner, and subject to this, their second-choice partner, and so on, among the set of all stable matchings. A stable matching is *generous* if the minimum number of men and women are assigned to the worst ranked choice, and subject to that, the second-to-worst ranked choice, and so on. Let n be the number of men or women, and m be the total length of all preference lists. In this chapter we develop an adaptation to an existing $O(nm^2 \log n)$ algorithm for finding a rank-maximal and generous stable matching, to work with a far reduced memory allocation requirement (an estimated 100-fold improvement for instances with $100,000$ men or women). The definitions of rank-maximal and generous stable matchings in SR are analogous to the SMI case. We show that the problem of finding stable matchings of these types in SR is NP-hard. Additionally we present an empirical evaluation to compare various fairness measures over profile-based and other types of fair stable matchings.

- *Chapter 5: Large stable matchings in* SPA-ST. The largest body of work is given in Chapter 5 where SPA-ST is investigated. Recall that Király [40] described a $\frac{3}{2}$-approximation algorithm for MAX-HRT. Chapter 5 extends this work with the creation of a $\frac{3}{2}$-approximation algorithm for MAX-SPA-ST. The introduction of a third type of agent (lecturers) adds significant complications compared to the HRT case, resulting in extensive correctness proofs for this algorithm.

- *Chapter 6: Experiments and further results for* SPA-ST. Finally, Chapter 6 comprises two separate contributions in the area of SPA-ST. Firstly, in Section 6.2 an IP model to find a stable matching in SPA-ST is presented. Using this IP model, the approximation algorithm developed in Chapter 5 is evaluated, investigating how close to the maximum stable matching size is achieved in practice by the approximation algorithm, using randomly-generated instances. We find that the approximation algorithm easily surpasses its $\frac{3}{2}$ bound, constructing a stable matching within 92% of optimal in all cases tested. Secondly, in Section 6.3, we study SPA-STL. We seek a matching that in some way balances student loads across lecturers according to their targets. We define the *load-max-balanced score* of a matching as the maximum absolute difference between a lecturer's target and their number of allocations. We also define the *load-sum-balanced score* as the sum of absolute differences between each lecturer target and their number of allocations. A *load-balanced matching* in SPA-STL is then a matching that minimises the load-max-balanced score and load-sum-balanced score over all matchings. We show that a load-balanced matching must exist and give an efficient algorithm to return such a matching. Additionally, we study the problem of finding a load-balanced stable matching, that is, a stable matching that simultaneously

minimises both the load-max-balanced score and load-sum-balanced score. We show that the problem of finding a load-balanced stable matching is NP-hard, and present new IP models to find such matchings.

Experimental results are presented in Chapters 3, 4 and 6. All experiments were conducted on the same machine with 32 cores, 8×64GB RAM and Dual Intel® Xeon® CPU E5-2697A v4 processors, unless otherwise stated.

Chapter 2

Literature review

2.1 The House Allocation problem (HA)

We begin this literature review by surveying bipartite matching problems with one-sided preferences. These problems arise in the case of the assignment of a set of applicants to a set of houses, such that each applicant and house may be assigned only once. Each applicant ranks a subset of houses (the applicant's *acceptable houses*) in order of preference, but the houses do not express any preference over applicants. This case, where there are two disjoint sets of entities, the members of one of which have preference over members of the other, is known as the *House Allocation problem* (HA) [27, 76, 1]. A *matching* in an HA instance may be defined as an assignment of applicants to houses such that each applicant and house is assigned at most once, and each applicant finds the house to which they are assigned acceptable. The definition of the *degree* of a matching in HA is analogous to that in the SMI case.

As with HR described in Chapter 1, each HA instance may admit many different matchings and the decision as to which matching is best will depend on the priorities of the matching scheme administrator. In contrast to SMI, HR and SPA-S, the concept of stability does not exist, since houses have no preference over applicants. However we are able to use several similar definitions that were given for SMI, to describe optimality criteria in HA. The definitions of rank-maximal and generous criteria for HA are analogous to the SMI setting. In a *rank-maximal matching*, the maximum number of applicants are assigned to their first-choice house, and subject to this, their second-choice house, and so on. In a *greedy maximum matching*, the maximum number of applicants are assigned, and, subject to this, we optimise on first, second, third choices etc, as in the case of a rank-maximal matching. Finally, in a *generous maximum matching*, the maximum number of applicants are assigned as possible and, subject to this, the generous criterion is applied. The rank and cost of an applicant can be defined in a similar way to SMI. Then, a *minimum cost maximum cardinality* matching is

a matching that first maximises the size of the matching and subject to this, minimises cost over all applicants.

Extensions to HA include the *House Allocation problem with Ties* (HAT), in which applicants may be indifferent between two or more houses on their preference list; the *Capacitated House Allocation problem* (CHA), where houses may admit more than one applicant up to some fixed upper limit; and the *Capacitated House Allocation problem with Ties* (CHAT) in which both extensions apply. The concepts of *degree* and *profile* extends naturally to these capacitated cases. There are polynomial-time algorithms for finding several profile-based optimal matchings in CHAT instances as follows. Let I be an instance of CHAT. Huang et al. [26] described an algorithm to find a rank-maximal matching M in I that runs in $\tilde{O}(dnm)$ time[1], where n is the number of applicants and houses, m is the total length of all preference lists and d is the degree of M. This algorithm may be adapted to find a greedy maximum matching and a generous maximum matching in I, and runs in the same time complexity. Finally, Gabow and Tarjan [18] described an algorithm to find a minimum cost maximum cardinality matching in I that runs in $O(\sqrt{n}m \log n)$ time.

Popularity, another type of optimality criterion, may also be defined in the HA context. A matching is said to be *popular* if there is no other matching preferred by the majority of applicants who have a strict preference between them. Popular matchings may be of different sizes and an algorithm to find a maximum-sized popular matching (hereafter a *maximum popular matching*) in an instance of HA, or report that none exists, is described by Abraham et al. [5]. This algorithm runs in $O(n + m)$ time where n is the sum of the number of applicants and houses, and m is the total length of all preference lists. Abraham et al. [5] describes a further $O(\sqrt{n_1}m)$ algorithm to find a maximum popular matching in an instance of HAT, where n_1 is the number of applicants. For instances of CHA and CHAT, Manlove and Sng [49] developed algorithms to find a maximum popular matching or report that none exists in $O(\sqrt{C}n_1 + m)$ and $O(\sqrt{C + n_1}m)$ respectively, where C is the sum of capacities of all houses. Finally, counting the number of popular matchings in HA can be done in linear time [55].

Pareto optimality is a further optimality criterion that may be applied to HA. Given an instance of HA or one of its extensions, a matching is *Pareto optimal* if there is no other matching in which an applicant is made better off, without making another applicant worse off. The Serial Dictatorship Mechanism (SDM), a description of which may be found in Roth and Sotomayor's book [69], finds a Pareto optimal matching in an instance of HA in $O(m)$ time, where m is the total length of all preference lists. An applicant ordering is required as input to this algorithm, with applicants nearer the top of the list having advantage. The Random Serial Dictatorship Mechanism is identical to Algorithm SDM except that a

[1] $\tilde{O}$ notation (also known as 'Soft-O') is used to represent the 'big-O' growth-rate of a function such that logarithmic factors are ignored. Thus if $f(n) = \tilde{O}(g(n))$ then $f(n) = O(g(n) \log^k g(n))$ for some constant k.

random applicant ordering is used as input. Finally, Pareto optimal matchings may be of different sizes and a maximum-sized such matching can be found in $O(\sqrt{n_1}m)$ where n_1 is the number of applicants [4].

2.2 The Stable Marriage problem (SM)

2.2.1 Introduction

We now move on to considering bipartite matching problems with two-sided preferences. The Stable Marriage problem (SM), the archetypal problem in this category, was first introduced in Gale and Shapley's seminal paper "College Admission and the Stability of Marriage" [19]. In an instance of SM we have two sets of agents, men and women (of equal number, henceforth n), such that each man ranks every woman in strict preference order, and vice versa. An extension to SM, known as the Stable Marriage problem with Incomplete lists (SMI), described in Chapter 1, allows each man (woman) to rank a *subset* of women (men). A *stable matching* in SM may be defined in an identical way to the SMI case. Chapters 3 and 4 deal with fairness over stable matchings in SMI, and we therefore present further results with respect to this SMI scenario.

2.2.2 The Stable Marriage problem with Incomplete lists (SMI)

2.2.2.1 Formal definitions

In this section we give a more formal definition of SMI. An instance I of the Stable Marriage problem with Incomplete lists (SMI) comprises two sets of n agents, men $U = \{m_1, m_2, ..., m_n\}$ and women $W = \{w_1, w_2, ..., w_n\}$. Each man (woman) ranks a subset of women (men) in strict preference order, where m denotes the total length of all preference lists. A man m_i finds a woman w_j *acceptable* if w_j appears on m_i's preference list. Similarly, a woman w_j finds a man m_i *acceptable* if m_i appears on w_j's preference list. Without loss of generality, we assume that acceptability is symmetric, i.e., given any man m_i and any women w_j, m_i finds w_j acceptable if and only if w_j finds m_i acceptable. A pair (m_i, w_j) is *acceptable* if m_i finds w_j acceptable and w_j finds m_i acceptable. A *matching* M in this context is an assignment of men to women comprising acceptable pairs such that no man or woman is assigned to more than one person. An example SMI instance I_0 with 8 men and women is taken from Gusfield and Irving's book [25, p. 69] and is given as Figure 2.1. This example has *complete* preference lists, where each man ranks every woman, and vice versa, and is therefore also an instance of SM. If m_i is assigned in M, we let $M(m_i)$ denote m_i's assigned partner. Similarly, if w_j is assigned in M, we let $M(w_j)$ denote w_j's assigned partner.

Men's preferences:	Women's preferences:
m_1: w_5 w_7 w_1 w_2 w_6 w_8 w_4 w_3	w_1: m_5 m_3 m_7 m_6 m_1 m_2 m_8 m_4
m_2: w_2 w_3 w_7 w_5 w_4 w_1 w_8 w_6	w_2: m_8 m_6 m_3 m_5 m_7 m_2 m_1 m_4
m_3: w_8 w_5 w_1 w_4 w_6 w_2 w_3 w_7	w_3: m_1 m_5 m_6 m_2 m_4 m_8 m_7 m_3
m_4: w_3 w_2 w_7 w_4 w_1 w_6 w_8 w_5	w_4: m_8 m_7 m_3 m_2 m_4 m_1 m_5 m_6
m_5: w_7 w_2 w_5 w_1 w_3 w_6 w_8 w_4	w_5: m_6 m_4 m_7 m_3 m_8 m_1 m_2 m_5
m_6: w_1 w_6 w_7 w_5 w_8 w_4 w_2 w_3	w_6: m_2 m_8 m_5 m_3 m_4 m_6 m_7 m_1
m_7: w_2 w_5 w_7 w_6 w_3 w_4 w_8 w_1	w_7: m_7 m_5 m_2 m_1 m_8 m_6 m_4 m_3
m_8: w_3 w_8 w_4 w_5 w_7 w_2 w_6 w_1	w_8: m_7 m_4 m_1 m_5 m_2 m_3 m_6 m_8

Figure 2.1: SMI instance I_0 [25, p. 69].

Matching M is *stable* if it has no *blocking pair* (m_i, w_j) which is defined as follows:

1. (m_i, w_j) is an acceptable pair, and;
2. m_i is unassigned or prefers w_j to $M(m_i)$, and;
3. w_j is unassigned or prefers m_i to $M(w_j)$.

A stable matching must exist in every instance of SMI [19]. Gale and Shapley [19] described two $O(m)$ time algorithms to find a stable matching in an instance of SMI. The Man-oriented Gale-Shapley Algorithm produces a *man-optimal* stable matching, that is, a stable matching in which each man is assigned to his most-preferred woman in any stable matching. Unfortunately, the man-optimal stable matching is also *woman-pessimal* where each woman is assigned to her least-preferred man in any stable matching [58]. Similarly the Woman-oriented Gale-Shapley Algorithm produces the woman-optimal (man-pessimal) stable matching.

As in the HR case, there may be many stable matchings in a particular instance of SMI. We denote by $\mathcal{M}_S$ the set of all stable matchings in I, which may be exponential in size [30]. By the Rural Hospitals theorem [20], the same set of men and women are assigned in all stable matchings of $\mathcal{M}_S$. Thus, for the remainder of this thesis, we assume any instance I of SMI has been pre-processed using the Man-oriented Gale-Shapley Algorithm to discard all agents unassigned in any stable matching.

Recall that the man-optimal stable matching is also the woman-pessimal stable matching, and vice versa. Clearly, it may not be ideal to have one group so heavily favoured over the other, and so it is natural to wish to find a stable matching in $\mathcal{M}_S$ that is in some sense fair for both sets of men and women. Before examining fairness in the next section, we provide several relevant definitions.

Let M be a stable matching in SMI. We denote by rank(m_i, w_j) the position of w_j in m_i's preference list, and define rank(w_j, m_i) similarly. The *rank* of m_i with respect to matching

M is defined rank$(m_i, M(m_i))$. An analogous definition holds for women. We define the *man-degree* $d_U(M)$ of matching M as the largest rank of all men in M, that is, $d_U(M) = \max\{\text{rank}(m_i, M(m_i)) : m_i \in U\}$. Similarly, the *woman-degree* $d_W(M)$ of matching M is given by $d_W(M) = \max\{\text{rank}(w_j, M(w_j)) : w_j \in W\}$. The *degree* of M, denoted $d(M)$, is given by $d(M) = \max\{d_U(M), d_W(M)\}$. Define the *degree pair* of M, denoted $d'(M) = (a, b)$ as the tuple of man and woman degrees in M, where $a = d_U(M)$ and $b = d_W(M)$. The *man-cost* $c_U(M)$ of matching M is defined as the sum of ranks of all men, that is, $c_U(M) = \sum_{m_i \in U} \text{rank}(m_i, M(m_i))$. Similarly, the *woman-cost* of matching M is given by $c_W(M) = \sum_{w_j \in W} \text{rank}(w_j, M(w_j))$. Finally, the *cost* of matching M is denoted $c(M)$ where $c(M) = c_U(M) + c_W(M)$.

2.2.2.2 Fairness

There are several optimality criteria we may wish to use when defining the notion of a fair stable matching in an instance of SMI. We present a review of these criteria here. Let I be an instance of SMI.

Firstly, we examine cost-based optimality criteria.

Given a stable matching M, define its *balanced score* to be $\max\{c_U(M), c_W(M)\}$. M is *balanced* [14] if it has minimum balanced score over all stable matchings in $\mathcal{M}_S$. Feder [14] showed that the problem of finding a balanced stable matching in SMI is NP-hard, although can be approximated within a factor of 2. This approximation factor was improved by McDermid to $2 - \frac{1}{l}$, where l is the length of the longest preference list, as noted in Manlove [47, p. 110]. Gupta et al. [22] showed that a balanced stable matching can be found in $O(f(n)8^t)$ time when parameterised by $t = k - \min\{c_U(M_0), c_W(M_z)\}$, where $f(n)$ is a function polynomial in n and k is the balanced score of a balanced stable matching. The *sex-equal score* of M is defined to be $|c_U(M) - c_W(M)|$. M is *sex-equal* [25] if it has minimum sex-equal score over all stable matchings in $\mathcal{M}_S$. Finding a sex-equal stable matching was shown to be NP-hard by Kato [38]. This result was later strengthened by McDermid and Irving [56] who showed that, even in the case when preference lists have length at most 3, the problem of deciding whether there is a stable matching with sex-equal score 0 is NP-complete. Since sex-equal scores may equal 0, it does not make sense to seek an algorithm that approximates the sex-equal score to within a certain factor. Iwama et al. [35] define a *near sex-equal stable matching* (if it exists) as a stable matching M such that $|c_U(M) - c_W(M)| \leq \epsilon\Delta$ where $\Delta = \min\{|c_U(M_0) - c_W(M_0)|, |c_U(M_z) - c_W(M_z)|\}$ and ϵ is a positive constant [35]. Additionally, they describe an $O(n^{3+\frac{1}{\epsilon}})$ algorithm to find a near sex-equal stable matching or report that none exists. A polynomial-time algorithm to find a sex-equal stable matching was described for instances in which men have preference lists of length at most 2 (women's preference lists remaining unbounded) [56].

A stable matching M is *egalitarian* [41] if $c(M)$ is minimum over all stable matchings in $\mathcal{M}_S$. A weight-based generalisation of an instance I of SMI occurs when we allow each man m_i to assign a unique positive integer *weight* to each woman w_j on his preference list, denoted $w(m_i, w_j)$. A similar definition exists for the weight of a man on a woman's preference list. Rankings on men's and women's preference lists may then be inferred from these weights. Let $w(M)$ denote the weight function of stable matching M, such that $w(M) = \sum_{(m_i,w_j)\in M} (w(m_i, w_j) + w(w_j, m_i))$. A matching M is *minimum (maximum) weight* if $w(M)$ is minimum (maximum) taken over all stable matchings in I. Thus the minimum weight function $w(M)$ is a generalisation of the *egalitarian cost function* that minimises $c(M)$ over all stable matchings. Irving et al. [32] showed that an egalitarian stable matching can be found in $O(m^2)$ time and a minimum weight stable matching in $O(m^2 \log n)$ time. Additionally, Irving et al. [32] described a simple transformation that allows the minimum weight stable matching algorithm to be used to find a maximum weight stable matching in the same time complexity. Let K be the weight of a minimum weight stable matching. Feder [15] improved on the methods above, giving an $O(m^{1.5})$ algorithm, based on weighted SAT, for finding a minimum weight stable matching when $K \leq m$ (which includes the case where we wish to find an egalitarian stable matching), and an $O(nm \log m)$ algorithm to find a minimum weight stable matching in the general case when $m < K$ and $K = O(m^c)$, where c is a constant value.

Secondly, we examine degree-based optimality criteria. A stable matching M is *minimum regret* [41] if the maximum of $d_U(M)$ and $d_W(M)$ is minimum among all stable matchings in $\mathcal{M}_S$. It is possible to find a minimum regret stable matching in $O(m)$ time [23].

Next, we examine profile-based optimality criteria. We define the *profile* of a stable matching as follows. Given a stable matching M, let the profile of M be given by the vector $p(M) = \langle p_1, p_2, ..., p_n\rangle$ where $p_k = |\{(m_i, w_j) \in M : \text{rank}(m_i, w_j) = k\}| + |\{(m_i, w_j) \in M : \text{rank}(w_j, m_i) = k\}|$ for each k $(1 \leq k \leq n)$. Thus we define a stable matching M in I to be *rank-maximal* if $p(M)$ is lexicographically maximum, taken over all stable matchings in $\mathcal{M}_S$. We define the *reverse profile* $p_r(M)$ to be the vector $p_r(M) = \langle p_k, p_{k-1}, ..., p_1\rangle$, where k is the degree of a minimum regret stable matching. A stable matching M in I is *generous* if $p_r(M)$ is lexicographically minimum, taken over all stable matchings in $\mathcal{M}_S$. Irving et al. [32] showed that a rank-maximal stable matching and a generous stable matching can be found in $O(nm^2 \log n)$ time. Feder [15] improved on this, giving an $O(n^{0.5}m^{1.5})$ algorithm for finding a rank-maximal stable matching.

We now describe a *median* stable matching, which is an additional type of fair stable matching whose definition is not based on degree, cost or profile. A *median* stable matching may be described in the following way. Let l_i denote the multiset of all women who are assigned to man m_i in the stable matchings in $\mathcal{M}_S$ (in general l_i is a multiset as m_i may have the same partner in more than one stable matching). Assume that l_i is sorted according to m_i's

ρ_0: (m_1, w_5) (m_3, w_8)
ρ_1: (m_1, w_8) (m_2, w_3) (m_4, w_6)
ρ_2: (m_3, w_5) (m_6, w_1)
ρ_3: (m_7, w_2) (m_5, w_7)
ρ_4: (m_3, w_1) (m_5, w_2)

Figure 2.2: Rotations for instance I_0.

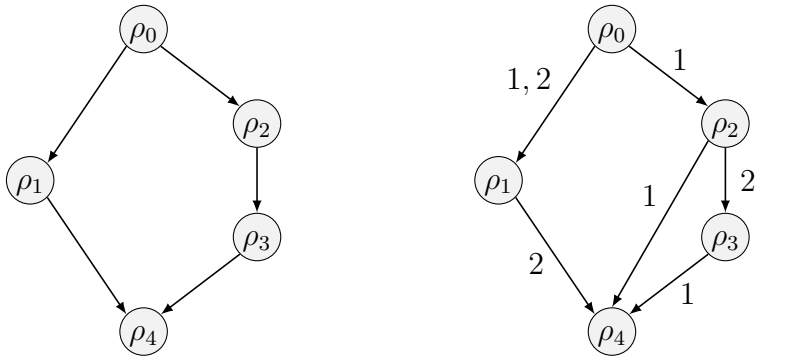
preference order (there may be repeated values) and let $l_{i,j}$ represent the jth element of this list. For each j ($1 \leq j \leq |\mathcal{M}_S|$), let M_j denote the set of pairs obtained by assigning m_i to $l_{i,j}$. Teo and Sethuraman [73] showed the surprising result that M_j is a stable matching for every j such that $1 \leq j \leq |\mathcal{M}_S|$. If $|\mathcal{M}_S|$ is odd then the unique *median* stable matching is found when $j = \left\lceil \frac{|\mathcal{M}_S|}{2} \right\rceil$. However, if $|\mathcal{M}_S|$ is even, then the *set of median* stable matchings are the stable matchings such that each man (woman) does no better (worse) than his (her) partner when $j = \frac{|\mathcal{M}_S|}{2}$ and no worse (better) than his (her) partner when $j = \frac{|\mathcal{M}_S|}{2} + 1$. For the purposes of this thesis, in particular Chapter 4, we define the *median* stable matching as the stable matching found when $j = \left\lceil \frac{|\mathcal{M}_S|}{2} \right\rceil$. Computing the set of median stable matchings is #P-hard [9].

2.2.2.3 Structure of stable matchings

In this section, we describe a structural characterisation of the stable matchings in SMI, which may be exploited to enumerate all stable matchings of a given instance.

For some stable matching M in an instance I of SMI, let $s(m_i, M)$ denote the next woman w_j on m_i's preference list (starting from $M(m_i)$) who prefers m_i to $M(w_j)$. A *rotation* ρ is then a sequence of man-woman pairs $\{(m_1, w_1), (m_2, w_2), ..., (m_q, w_q)\}$ in M, such that $m_{i+1} = M(s(m_i, M))$ for $1 \leq i \leq q$, where addition is taken modulo q [32]. We say rotation ρ is *exposed* on M if $\{(m_1, w_1), (m_2, w_2), ..., (m_q, w_q)\} \subseteq M$. If ρ is exposed on M, we may *eliminate* ρ on M, that is, remove all pairs of ρ from M and add pairs (m_i, w_{i+1}) for $1 \leq i \leq q$, where addition is taken modulo q, in order to produce another stable matching M' of I. Recall Figure 2.1 shows example instance I_0. A list of rotations of I_0 is given in Figure 2.2.

An example of a rotation being eliminated on a stable matching now follows. The man-optimal stable matching for instance I_0 is given by $M_0 = \{(m_1, w_5), (m_2, w_3), (m_3, w_8), (m_4, w_6), (m_5, w_7), (m_6, w_1), (m_7, w_2), (m_8, w_4)\}$. The only rotation exposed on M_0 from Figure 2.2, according to the above definition, is $\rho_0 = \{(m_1, w_5), (m_3, w_8)\}$. Permuting the pairs of ρ_0 involves moving m_1 to have partner w_8 and moving m_3 to have partner w_5. Hence, eliminating ρ_0 on M_0 results in the stable matching $M_0' = \{(m_1, w_8), (m_2, w_3), (m_3, w_5), (m_4, w_6), (m_5, w_7), (m_6, w_1), (m_7, w_2), (m_8, w_4)\}$.



(a) Rotation poset $R_p(I_0)$. (b) Rotation digraph $R_d(I_0)$.

Figure 2.3: Rotation poset and digraph of I_0.

The *rotation poset* $R_p(I)$ of I indicates the order in which rotations may be eliminated. Rotation ρ is said to *precede* rotation τ if τ is not exposed until ρ has been eliminated. There is a one-to-one correspondence between the set of stable matchings and the set of closed subsets of $R_p(I)$ [32, Theorem 3.1]. Gusfield and Irving [25] describe an additional structure known as the *rotation digraph* $R_d(I)$ of I which is based on $R_p(I)$ and allows for the enumeration of all stable matchings in $O(m + n|\mathcal{M}_S|)$ time. A description of the creation of a *rotation digraph* now follows. First, retain each rotation from the rotation poset as a vertex. There are two types of predecessor relationships to consider.

1. Suppose pair $(m_i, w_j) \in \rho$. We have a directed edge in our digraph from ρ' to ρ if ρ' is the unique rotation that moves m_i to w_j. In this case we say that ρ' is a *type* 1 *predecessor* of ρ.

2. Let ρ be the rotation that moves m_i below w_j and $\rho' \neq \rho$ be the rotation that moves w_j above m_i. Then we add a directed edge from ρ' to ρ and say ρ' is a *type* 2 *predecessor* of ρ.

The rotation poset and rotation digraph for instance I_0, denoted $R_p(I_0)$ and $R_d(I_0)$ respectively, are shown in Figure 2.3.

Using the rotation digraph structure, Gusfield and Irving [25] showed how to enumerate all stable matchings in $O(m + n|\mathcal{M}_S|)$ time, where $\mathcal{M}_S$ is the set of all stable matchings. All stable matchings of instance I_0 are listed in Figure 2.4.

2.2.3 The Stable Marriage problem with Ties and Incomplete lists (SMTI)

The Stable Marriage problem with Ties and Incomplete lists (SMTI) is an extension to SMI in which men (women) may have ties in his (her) preference lists, indicating an indifference

$$M_0 = \{(m_1, w_5), (m_2, w_3), (m_3, w_8), (m_4, w_6), (m_5, w_7), (m_6, w_1), (m_7, w_2), (m_8, w_4)\}$$
$$M_1 = \{(m_1, w_8), (m_2, w_3), (m_3, w_5), (m_4, w_6), (m_5, w_7), (m_6, w_1), (m_7, w_2), (m_8, w_4)\}$$
$$M_2 = \{(m_1, w_3), (m_2, w_6), (m_3, w_5), (m_4, w_8), (m_5, w_7), (m_6, w_1), (m_7, w_2), (m_8, w_4)\}$$
$$M_3 = \{(m_1, w_8), (m_2, w_3), (m_3, w_1), (m_4, w_6), (m_5, w_7), (m_6, w_5), (m_7, w_2), (m_8, w_4)\}$$
$$M_4 = \{(m_1, w_3), (m_2, w_6), (m_3, w_1), (m_4, w_8), (m_5, w_7), (m_6, w_5), (m_7, w_2), (m_8, w_4)\}$$
$$M_5 = \{(m_1, w_8), (m_2, w_3), (m_3, w_1), (m_4, w_6), (m_5, w_2), (m_6, w_5), (m_7, w_7), (m_8, w_4)\}$$
$$M_6 = \{(m_1, w_3), (m_2, w_6), (m_3, w_1), (m_4, w_8), (m_5, w_2), (m_6, w_5), (m_7, w_7), (m_8, w_4)\}$$
$$M_7 = \{(m_1, w_3), (m_2, w_6), (m_3, w_2), (m_4, w_8), (m_5, w_1), (m_6, w_5), (m_7, w_7), (m_8, w_4)\}$$

Figure 2.4: Stable matchings for instance I_0.

Men's preferences:	Women's preferences:
m_1: w_1	w_1: m_2 m_1
m_2: (w_1 w_2)	w_2: (m_2 m_3)
m_3: w_2 w_3	w_3: m_3
m_4: (w_4 w_5)	w_4: (m_4 m_5)
m_5: (w_4 w_6)	w_5: (m_4 m_6)
m_6: (w_5 w_6)	w_6: (m_5 m_6)

Figure 2.5: SMTI instance I_1 [47, p. 152].

between two or more women (men). The definitions of an *acceptable* man-woman pair, a *matching* M and the notation $M(m_i)$ and $M(w_j)$ for man m_i and woman w_j, follow from the SMI case. An example instance I_1 of SMTI is shown in Figure 2.5, where brackets indicate ties in preference lists. The introduction of ties has the effect of changing what it means for a matching to be stable. We present the following stability definitions for SMTI.

- *Weakly stable:* A matching M in SMTI is *weakly stable* if it has no *blocking pair* (m_i, w_j) such that the following conditions hold:
  1. (m_i, w_j) is acceptable, and;
  2. m_i is unassigned or prefers w_j to $M(m_i)$, and;
  3. w_j is unassigned or prefers m_i to $M(w_j)$.
- *Strongly stable:* A matching M in SMTI is *strongly stable* if it has no *blocking pair* (m_i, w_j) such that either of the following holds:
  - 1. (m_i, w_j) is acceptable, and;
    2. m_i is unassigned or prefers w_j to $M(m_i)$, and;
    3. w_j is unassigned or prefers m_i to $M(w_j)$ or is indifferent between them.

    or
  - 1. (m_i, w_j) is acceptable, and;
    2. m_i is unassigned or prefers w_j to $M(m_i)$ or is indifferent between them, and;

3. w_j is unassigned or prefers m_i to $M(w_j)$.

- *Super stable:* A matching M in SMTI is *super-stable* if it has no *blocking pair* (m_i, w_j) such that the following conditions hold:

  1. (m_i, w_j) is acceptable, and;
  2. m_i is unassigned or prefers w_j to $M(m_i)$ or is indifferent between them, and;
  3. w_j is unassigned or prefers m_i to $M(w_j)$ or is indifferent between them.

The definition of weak-stability in SMTI is identical to stability in the SMI case. It is clear from the stability definitions above that all strongly stable and super-stable matchings are also weakly stable. Instance I_1 has four weakly stable matchings as shown in Figure 2.6. Here, M_1 and M_2 are also strongly stable matchings. This must be true since 1) each man and woman in both M_1 and M_2 are assigned their first-choice partners, 2) unassigned man m_1 would rather be assigned to w_1, however w_1 prefers her assigned partner m_2, and 3) unassigned woman w_3 would rather be assigned to m_3, however m_3 prefers his assigned partner w_2. Finally since all matchings contain either the pair (m_5, w_6) or (m_6, w_6), both of which are blocking pairs under super-stability, there is no super-stable matching in instance I_1.

As shown above, a super-stable matching need not exist. The same result also holds for strong stability as shown by instance I_2 in Figure 2.7, where the matching $\{(m_1, w_1), (m_2, w_2)\}$ is blocked by pair (m_2, w_1) and the matching $\{(m_1, w_2), (m_2, w_1)\}$ is blocked by pair (m_2, w_1) under the strong stability definition. In an instance I of SMTI, a matching is *super-stable* if and only if it is weakly stable in every instance that is derived by breaking ties of I [33]. Thus we may deduce that if a weakly stable matching M in an instance of SMTI exists, such that the associated preference list elements of pairs in M are not involved in ties, then this is a super-stable matching. An example therefore, of a super-stable matching, is given by $\{(m_1, w_1), (m_2, w_2), (m_3, w_3)\}$ in an instance of size $n = 3$, where m_1, m_2 and m_3 rank respectively w_1, w_2 and w_3 as their first choice (with no ties), and vice versa, regardless of the choice of other preference list elements (which may include ties for second choices).

It is possible to find a weakly stable matching in SMTI in linear time, by breaking ties of the instance randomly to form an SMI instance [51], and then using either the Man-optimal

$$
\begin{aligned}
M_0 &= \{(m_2, w_1), (m_3, w_2), (m_4, w_4), (m_6, w_6)\} \\
M_1 &= \{(m_2, w_1), (m_3, w_2), (m_4, w_4), (m_5, w_6), (m_6, w_5)\} \\
M_2 &= \{(m_2, w_1), (m_3, w_2), (m_4, w_5), (m_5, w_4), (m_6, w_6)\} \\
M_3 &= \{(m_1, w_1), (m_2, w_2), (m_3, w_3), (m_4, w_4), (m_5, w_6), (m_6, w_5)\}
\end{aligned}
$$

Figure 2.6: Stable matchings for instance I_1.

Men's preferences:	Women's preferences:
m_1: w_1 w_2	w_1: (m_1 m_2)
m_2: w_1 w_2	w_2: m_1 m_2

Figure 2.7: SMTI instance I_2 [47, p. 150].

or Woman-optimal Gale-Shapley algorithm to find a stable matching. Kavitha et al. [39] developed an algorithm for the problem of finding a strongly stable matching or reporting than none exists, given an instance I of SMTI, that runs in $O(nm)$ time. Manlove [46] describes an algorithm that find a super-stable matching or reporting than none exists, in an instance I of SMTI, that runs in $O(m)$ time.

From this point on, weak stability in SMTI may also be referred to as *stability*. Given an instance of SMTI, stable matchings may be of different sizes. The problem of finding a maximum stable matching in SMTI is denoted MAX-SMTI, and was shown to be NP-hard [51]. This NP-hardness result remains even in the case that each man's preference list is strictly ordered and of length 3, and, each woman's preference list is either strictly ordered and of length 3, or is a tie of length 2 [57]. Yanagisawa [74] showed that it is not possible to approximate the problem of finding a maximum stable matching within a factor of $\frac{33}{29}$ unless P = NP. McDermid [53] developed a $\frac{3}{2}$-approximation algorithm for this problem, and a simpler approximation algorithm was later developed by Király [40] with the same approximation factor.

2.3 The Stable Roommates problem (SR)

A further type of matching problem exists when we have a single set of entities, who match to other members of the same set. This is a non-bipartite generalisation of SM, known as the *Stable Roommates problem* (SR). An instance of SR consists of a single set of n agents (roommates), $A = \{a_1, a_2, ..., a_n\}$, each of whom ranks other members of the set in strict order of preference. A matching in this context is an assignment of pairs of agents such that each agent is assigned exactly once. The *Stable Roommates problem with Incomplete lists* (SRI) is an extension of SR in which each agent ranks a subset of the others. Let m be the total length of all preference lists. An agent a_i finds another agent a_j *acceptable* if a_j appears on a_i's preference list. As in the SMI case, we assume that acceptability is symmetric, i.e., given any pair of agents a_i and a_j, a_i finds a_j acceptable if and only if a_j finds a_i acceptable. A pair $\{a_i, a_j\}$ of agents is defined as *acceptable* if a_i finds a_j acceptable and a_j finds a_i acceptable. Figure 2.8 shows an example SRI instance I_3. A *matching* M in SRI is then an assignment of acceptable pairs of agents such that each agent is assigned at most once. If a_i is assigned in a matching M, we let $M(a_i)$ denote a_i's assigned partner.

Agents's preferences:
a_1: a_3 a_2 a_5 a_4
a_2: a_1 a_5 a_4
a_3: a_1 a_5
a_4: a_5 a_1 a_2
a_5: a_1 a_2 a_4 a_3

Figure 2.8: SRI instance I_3.

The notion of stability also exists in this setting. Given an instance I of SRI, a pair $\{a_i, a_j\}$ is a *blocking pair* if:

1. $\{a_i, a_j\}$ is an acceptable pair, and;
2. a_i is unassigned or prefers a_j to $M(a_i)$, and;
3. a_j is unassigned or prefers a_i to $M(a_j)$

A matching is considered *stable* if it admits no blocking pair. Using a counterexample, Gale and Shapley [19] showed that a stable matching in SR (and therefore SRI) need not exist in all instances. Irving [29] gave an $O(n^2)$ algorithm to find a stable matching in SR or report that no such matching exists. This algorithm may be easily extended to the SRI case, and runs in $O(m)$ time [25]. We define a *solvable* instance as an instance that admits at least one stable matching. Let I be an instance of SRI. Then, if I is solvable, the set of stable matchings of I must all involve the same set of agents [25]. As in the SMI case, when we henceforth study an SRI instance I, we assume that Irving's algorithm to find a stable matching has been used to pre-process I, discarding all agents unassigned in any stable matching.

As with SMI, a structural characterisation of stable matchings exists in solvable instances of SRI, which involves the *rotation poset*. This structure (described in more detail in Gusfield and Irving's book [25]) can be exploited to give the following algorithmic results. A *stable pair* is a pair of agents that belong to some stable matching. Gusfield [24] described an algorithm to find the set of all stable pairs in $O(nm \log n)$ time. This was later improved by Feder [15] to run in $O(nm)$ time. Secondly, the set of stable matchings can be listed in $O(m)$ time per matching, after a pre-processing time of $O(nm \log n)$ [24, 25]. Again, this was later improved by Feder [16] to run in $O(n)$ time per matching, after a pre-processing time of $O(m)$.

Let M be a stable matching in SRI. For any two agents a_i and a_j, we denote by rank(a_i, a_j) the position of a_j in a_i's preference list and define the *rank* of a_i with respect to matching M as rank$(a_i, M(a_i))$. The *degree* of M is the largest rank of all agents in M. A stable matching M is *minimum regret* if the degree of M is minimum over all stable matchings.

Gusfield and Irving [25] describe an algorithm to find a minimum regret stable matching, in a solvable instance I of SRI, in $O(m)$ time. As in the SMI case, the *profile* of M is given by the vector $p(M) = \langle p_1, p_2, ..., p_n \rangle$ where $p_k = |\{a_i \in A : \text{rank}(a_i, M(a_i)) = k\}|$ for each k $(1 \leq k \leq n)$. A stable matching M is then *rank-maximal* if $p(M)$ is lexicographically maximum, taken over all stable matchings. Finally, a stable matching M is *generous* if the *reverse profile* $p_r(M) = \langle p_n, p_{n-1}, ..., p_1 \rangle$ is lexicographically minimum, taken over all stable matchings.

The *Stable Roommates problem with Ties and Incomplete lists* (SRTI) is an extension of SRI in which an agent may be indifferent between two or more other agents on their preference list. The definition of a *matching* carries over from the SRI case. As in SMTI, the definition of a stable matching in SRTI can be expanded. A *weakly stable*, *strongly stable* and *super-stable* matching in SRTI may be defined in an analogous way to SRTI. For a solvable instance I of SRTI, and any matching M of I, M is weakly stable in I if and only if it is also stable in some instance of SRI created from breaking ties in I [31]. However, determining whether an instance of SRTI admits a weakly stable matching is NP-complete [66]. Finally, Kunysz [42] described an $O(nm)$ algorithm to find a strongly stable matching or report than none exists, and Irving and Manlove [31] described an $O(m)$ algorithm to find a super-stable matching or report than none exists.

2.4 The Hospitals/Residents problem (HR)

2.4.1 Introduction

An extension to SMI is known as Hospitals/Residents problem (HR) and was introduced in Chapter 1. Formally, an instance of HR has two sets of agents, namely residents and hospitals. Residents rank a subset of hospitals (their *acceptable hospitals*) in strict preference order, and vice versa. As in SMI, without loss of generality we assume that acceptability is symmetric, i.e., given a resident r_i and a hospital h_j, r_i finds h_j acceptable if and only if h_j finds r_i acceptable. A resident-hospital pair (r_i, h_j) is *acceptable* if r_i is acceptable to h_j and h_j is acceptable to r_i. Denote by c_j the capacity of hospital h_j. A *matching* in this scenario is defined as an allocation of residents to hospitals such that each resident-hospital pair in M is acceptable, each resident may be assigned at most one hospital, and each hospital h_j may be assigned at most c_j residents. If r_i is assigned in matching M, we let $M(r_i)$ denote r_i's assigned hospital, otherwise if r_i is unassigned, we define $M(r_i) = \perp$ (undefined). Denote by $M(h_j)$ the set of residents assigned to h_j in M. We describe a hospital h_j as *undersubscribed* if $|M(h_j)| < c_j$, and *full* if $|M(h_j)| = c_j$.

A pair (r_i, h_j) of M is *blocking* if the following conditions hold:

1. (r_i, h_j) is acceptable, and;
2. r_i is unassigned or prefers h_j to $M(r_i)$, and;
3. Either h_j is undersubscribed, or is full and prefers r_i to its worst ranked resident in $M(h_j)$.

A matching M is *stable* if it admits no blocking pair.

As with the SMI case, Gale and Shapley [19] showed that a stable matching always exists in an instance of HR and can be found in linear time using the Resident-oriented Gale-Shapley Algorithm or the Hospital-oriented Gale-Shapley Algorithm [19]. There may be many stable matchings in a particular instance of HR.

The *"Rural Hospitals" Theorem* [68, 67, 20] identifies several properties that hold for all instances of HR, and is given as Theorem 2.4.1.

Theorem 2.4.1 ([68, 67, 20])**.** *Let I be and instance of* HR*. Then, the following properties hold in I:*

- *The same set of residents are assigned in all stable matchings;*
- *Each hospital is assigned the same number of residents in all stable matchings;*
- *Any hospital that is undersubscribed in one stable matching is assigned exactly the same set of residents in all stable matchings.*

2.4.2 Variants of HR

An extension to HR, known as the *Hospitals/Residents problem with Ties* (HRT) [46], allows residents to be indifferent between two or more hospitals on their preference list, and vice versa. HRT is an extension of SMTI, and there are counterpart definitions for *weak stability*, *strong stability* and *super-stability* in HRT. From this point on, we shorten weak stability to *stability* in HRT. A stable matching in HRT has an analogous definition to the HR case. Since HRT is an extension of SMTI, it also has the characteristic that stable matchings may be of different sizes. Recall that MAX-HRT is the NP-hard problem of finding a maximum stable matching in HRT. A $\frac{3}{2}$-approximation algorithm was developed by Király [40] for this problem. Yanagisawa [74] showed that there is no approximation algorithm for MAX-HRT with approximation factor $\frac{33}{29}$ unless $\mathsf{P} = \mathsf{NP}$.

The *Hospitals/Residents problem with Couples* (HRC) [67] extends HR to allow some residents to apply jointly to hospitals in couples (for example couples may wish to be assigned

to hospitals that are close to each other). This is achieved by allowing resident couples to submit a joint preference list over pairs of hospitals. The definition of a *stable matching* in HR may be extended to the HRC setting [47, Section 5.3]. In this context a stable matching may not exist, however it is possible to find a matching that is *almost-stable* (a matching with the minimum number of blocking pairs). Manlove et al. [52] showed that this was an NP-hard problem.

2.5 The Student-Project Allocation problem (SPA)

2.5.1 Introduction

The Student-Project Allocation problem (SPA) is an extension to HR in which students take the place of residents, projects take the place of hospitals and a third set of agents, lecturers, are introduced. Each project is then offered by a single lecturer, and each lecturer has an associated capacity that indicates the maximum number of students who may be allocated to projects they offer. There are two sub-cases that we consider for this problem. In SPA-S lecturers have preferences over students, and in SPA-P lecturers have preferences over projects. A review of each of these cases is now given.

2.5.2 The Student-Project Allocation problem with lecturer preferences over Students (SPA-S)

2.5.2.1 Formal definitions

An informal definition of the SPA-S problem was given in Chapter 1. We now give a more formal definition.

An instance I of SPA-S comprises a set $S = \{s_1, s_2, ..., s_{n_1}\}$ of *students*, a set $P = \{p_1, p_2, ..., p_{n_2}\}$ of *projects*, and a set $L = \{l_1, l_2, ..., l_{n_3}\}$ of *lecturers*, where $|S| = n_1$, $|P| = n_2$ and $|L| = n_3$. Each project is *offered* by a single lecturer, and each lecturer l_k *offers* a set of projects $P_k \subseteq P$, where $P_1, \ldots, P_k$ partitions P. Each project $p_j \in P$ has a *capacity* c_j, indicating the maximum number of students that may be assigned to it. Similarly, each lecturer $l_k \in L$ has a *capacity* d_k, indicating the maximum number of students who may be assigned to projects they offer. We assume without loss of generality that d_k is no greater than the sum of the capacities of the projects in P_k. Each student $s_i \in S$ has a set $A_i \subseteq P$ of *acceptable* projects that they rank in strict order of preference. Each lecturer $l_k \in L$ ranks the students s_i for which $A_i \cap P_k \neq \emptyset$ (i.e., s_i finds acceptable at least one project offered

Student preferences:	Project details:	Lecturer preferences:	
s_1: p_2 p_3 p_1	p_1: lecturer l_1, $c_1 = 1$	l_1: s_1 s_2 s_4 s_3	$d_1 = 2$
s_2: p_2 p_1	p_2: lecturer l_1, $c_2 = 2$	l_2: s_1 s_3	$d_2 = 2$
s_3: p_1 p_2 p_3	p_3: lecturer l_2, $c_3 = 2$		
s_4: p_1			

Figure 2.9: SPA-S instance I_4.

by l_k) in strict preference order. Let m denote the total length of all student preference lists. Figure 2.9 shows an example SPA-S instance I_4.

An *assignment* M in I is a subset of $S \times P$ such that, for each pair $(s_i, p_j) \in M$, $p_j \in A_i$, that is, s_i finds p_j acceptable. Let $M(s_i)$ denote the set of projects assigned to a student $s_i \in S$, let $M(p_j)$ denote the set of students assigned to a project $p_j \in P$, and let $M(l_k)$ denote the set of students assigned to projects in P_k for a given lecturer $l_k \in L$. A *matching* M is an assignment such that $|M(s_i)| \leq 1$ for all $s_i \in S$, $|M(p_j)| \leq c_j$ for all $p_j \in P$ and $|M(l_k)| \leq d_k$ for all $l_k \in L$. If $s_i \in S$ is assigned in a matching M, we let $M(s_i)$ denote s_i's assigned project, otherwise if s_i is unassigned, we define $M(s_i) = \perp$ (undefined). We describe a project p_j as *undersubscribed* or *full* if $|M(p_j)| < c_j$ or $|M(p_j)| = c_j$, respectively. Similarly, we describe a lecturer l_k as *undersubscribed* or *full* if $|M(l_k)| < d_k$ or $|M(l_k)| = d_k$, respectively.

Given a matching M in I, let $(s_i, p_j) \in (S \times P) \backslash M$ be a student-project pair, where p_j is offered by lecturer l_k. Then (s_i, p_j) is a *blocking pair* of M [6] if 1, 2 and 3 hold as follows:

1. s_i finds p_j acceptable;
2. s_i either prefers p_j to $M(s_i)$ or is unassigned in M;
3. Either (a), (b), or (c) holds as follows:
   (a) p_j is undersubscribed and l_k is undersubscribed;
   (b) p_j is undersubscribed, l_k is full and either $s_i \in M(l_k)$ or l_k prefers s_i to the worst student in $M(l_k)$;
   (c) p_j is full and l_k prefers s_i to the worst student in $M(p_j)$.

A matching M in an instance I of SPA-S is *stable* if it admits no blocking pair.

Abraham et al. [6] developed two $O(m)$ time algorithms to find a stable matching in an instance of SPA-S. One algorithm finds the *student-optimal stable matching*, in which each student is assigned their best project in any stable matching of the instance. The other algorithm finds the *lecturer-optimal stable matching* in which a similar optimality property holds for all the lecturers. It was thus proved that all SPA-S instances have a stable matching [6].

Abraham et al. [6] also described several properties that hold for a given SPA-S instance, known as the *"Unpopular Projects"* Theorem. This is given as Theorem 2.5.1.

Theorem 2.5.1 ([6]). *Let I be and instance of* SPA-S. *Then, the following properties hold in I:*

- *Each lecturer has the same number of assigned students in all stable matchings;*
- *The same set of students are assigned in all stable matchings;*
- *Any project offered by an undersubscribed lecturer has the same number of students in all stable matchings.*

2.5.2.2 The Student-Project Allocation problem with lecturer preferences over Students including Ties (SPA-ST)

The Student-Project Allocation problem with lecturer preferences over Students including Ties (SPA-ST) is an extension of SPA-S, in which students (lecturers) may have ties in their preference lists, indicating indifference between two or more projects (students). The *rank* of project p_j on student s_i's list, denoted rank(s_i, p_j), is defined as 1 plus the number of projects that s_i strictly prefers to p_j. An analogous definition exists for the rank of a student on a lecturer's list, denoted rank(l_k, s_i). In figures involving SPA-ST instances, ties are indicated using brackets, as in the SMTI case. Analogous definitions of a *matching* M and the notation $M(s_i)$, $M(p_j)$ and $M(l_k)$, for student s_i, project p_j and lecturer l_k, follow from the SPA-S case.

When ties exist in preference lists it is possible to extend the definition of stability, as in the SMTI and HRT cases to include *weak stability*, *strong stability* [61] and *super-stability* [60]. In this thesis we consider only weak stability in SPA-ST, which we refer to from this point on as *stability*. *Stability* in SPA-ST has the same definition as in SPA-S and therefore we bring forward the definition of a *blocking pair* from SPA-S to this setting. As in the SPA-S case, a stable matching must exist in every instance of SPA-ST. Such a matching can be found in $O(m)$ time by breaking ties randomly in the SPA-ST instance (to form an SPA-S instance) and then using the algorithm described in the previous section to find a student-optimal or lecturer-optimal stable matching [6]. However, in contrast with the SPA-S case, stable matchings may be of different sizes. Let $\mathcal{M}_S$ denote the set of all stable matchings. Recall that MAX-SPA-ST is the problem of finding a maximum stable matching in SPA-ST. We note that SMTI is the special case of SPA-ST in which there are an equal number of projects and lecturers, each lecturer offers a unique project and the capacity of each project and lecturer is 1. Thus, since MAX-SMTI is NP-hard [51], it follows that the more general MAX-SPA-ST

problem is also NP-hard. The result of Yanagisawa [74], which showed that it is not possible to approximate MAX-SMTI within a factor of $\frac{33}{29}$ unless P = NP, also carries over to this case.

2.5.2.3 The Student-Project Allocation problem with lecturer preferences over Students including Ties and Lecturer targets (SPA-STL)

The *Student-Project Allocation problem with lecturer preferences over Students including Ties and Lecturer targets* (SPA-STL) is an extension of SPA-ST in which lecturers have targets indicating a preferred number of allocations (set by the matching administrator).

Formally, we define an instance of the SPA-STL in the same way as SPA-ST with the following addition. For a given lecturer l_k, their target is denoted t_k and is in the range $0 \leq t_k \leq d_k$ (where d_k is the capacity for lecturer l_k). Analogous definitions to those in SPA-ST exist for the concepts of a *matching*, a *blocking pair*, *stability*, and the *rank* of a project (student) on a student's (lecturer's) preference list. Additionally, in the SPA-STL setting, for a given matching M, we adopt the notation $M(s_i)$, $M(p_j)$ and $M(l_k)$, for student s_i, project p_j and lecturer l_k, as defined in the SPA-ST case. We let $\mathcal{M}_S$ denote the set of all stable matchings in SPA-STL, and $\mathcal{M}$ denote the set of all matchings in I. There is currently no existing literature on this topic. Targets are typically defined by a matching scheme administrator, and reflect the ideal number of assignees for each lecturer as decided by the Head of School. It is therefore desirable to find a matching that assigns each lecturer a number of students that is as close to their target as possible.

2.5.3 The Student-Project Allocation problem with lecturer preferences over Projects (SPA-P)

An instance of the Student Project Allocation problem with lecturer preferences over Projects (SPA-P) is defined in an identical way to SPA-S with the following amendment. Each lecturer $l_k \in L$ does not rank students, but instead ranks all projects in P_k in strict preference order. The definition of a *matching* in SPA-P carries over from the SPA-S case.

Clearly, in this new setting the definition of a *blocking pair* changes.

Let M be a matching in an instance I of SPA-P. Let $(s_i, p_j) \in (S \times P) \backslash M$ be a student-project pair, where p_j is offered by lecturer l_k. Then (s_i, p_j) is a *blocking pair* of M if properties 1, 2 and 3 hold as follows:

1. s_i finds p_j acceptable;
2. s_i either prefers p_j to $M(s_i)$ or is unassigned in M;

3. p_j is undersubscribed and either (a), (b) or (c) holds as follows:

   (a) $s_i \in M(l_k)$ and l_k prefers p_j to $M(s_i)$;

   (b) $s_i \notin M(l_k)$ and l_k is undersubscribed;

   (c) $s_i \notin M(l_k)$ and l_k is full and l_k prefers p_j to their worst project p_r satisfying $M(p_r) \neq \emptyset$;

   where l_k is the lecturer who offers p_j.

In the SPA-P setting, a *blocking coalition* may also be formed in which there is a set of students $\{s_{i_0}, ..., s_{i_{a-1}}\}$ assigned in M, such that each student s_{i_b} would rather be assigned to $M(s_{i_{b+1}})$ than $M(s_{i_b})$, where addition is taken modulo a, $a \geq 2$ and $0 \leq b \leq a-1$.

Then, a matching is *stable* if it admits no blocking pair or blocking coalition. Manlove and O'Malley [48] showed that a stable matching must exist in an instance of SPA-P and can be found in $O(m)$ time. In contrast to the case for SPA-S, stable matchings may be different sizes, and we let MAX-SPA-P denote the problem of finding a maximum-sized stable matching in an instance of SPA-P. MAX-SPA-P is NP-hard [48]. Iwama et al. [37] described a $\frac{3}{2}$-approximation algorithm for MAX-SPA-P and showed that this problem is not approximable within a factor of $\frac{21}{19}$.

Henceforth, the reader is invited to refer to the glossary for a reminder of acronym definitions.

Chapter 3

Degree-based stable matchings in SMI

3.1 Introduction

3.1.1 Background

In this chapter we focus on fair stable matchings in SMI. The definition of SMI was presented in Section 2.2.2. Recall the linear-time Man-oriented Gale-Shapley Algorithm [19] produces the *man-optimal stable matching*, in which each man is assigned his best partner in any stable matching. Unfortunately, the man-optimal stable matching is also the *woman-pessimal stable matching*, in which each woman is assigned her worst partner in any stable matching. Similarly, the Woman-oriented Gale-Shapley Algorithm produces a *woman-optimal* (and *man-pessimal*) *stable matching*. This motivates the problem of finding a stable matching that in some way balances the interests of both men and women. Several definitions of fairness were introduced in Section 2.2.2.2 and are summarised in Table 3.1.

In Table 3.1 there are two new natural definitions of fairness that can be studied. Let I be an instance of SMI, with set of stable matchings M_S, and let M denote a stable matching $M \in M_S$.

- We define the *regret-equality score* $r(M)$ as $|d_U(M) - d_W(M)|$ for a given stable matching M. M is *regret-equal* if $r(M)$ is minimum, taken over all stable matchings in $\mathcal{M}_S$. Note that in general we will prefer a regret-equal stable matching M such that $d_U(M) + d_W(M)$ is minimised (e.g. $d'(M) = (3, 3)$ rather than $d'(M) = (10, 10)$).
- We define the *regret sum* as $d_U(M) + d_W(M)$ for a given stable matching M. M is *min-regret sum* if $d_U(M) + d_W(M)$ is minimum taken over all stable matchings in $\mathcal{M}_S$.

	Cost	Degree
Minimising the maximum	$\min_{M \in \mathcal{M}_S} \max\{c_U(M), c_W(M)\}$	$\min_{M \in \mathcal{M}_S} \max\{d_U(M), d_W(M)\}$
	Balanced stable matching [14]	Minimum regret stable matching [41]
Minimising the absolute difference	$\min_{M \in \mathcal{M}_S} \lvert c_U(M) - c_W(M) \rvert$	$\min_{M \in \mathcal{M}_S} \lvert d_U(M) - d_W(M) \rvert$
	Sex-equal stable matching [25]	Regret-equal stable matching *
Minimising the sum	$\min_{M \in \mathcal{M}_S} (c_U(M) + c_W(M))$	$\min_{M \in \mathcal{M}_S} (d_U(M) + d_W(M))$
	Egalitarian stable matching [41]	Min-regret sum stable matching *

Table 3.1: Commonly used definitions of fair stable matchings in SMI. Our contributions are labelled with an *.

3.1.2 Motivation

Let mentees take the place of men and mentors take the place of women. Thus, mentees (mentors) rank a subset of mentors (mentees) and may only be allocated one mentor (mentee) in any matching. If we used the (renamed) Mentee-Oriented Gale-Shapley Algorithm [19] to find a stable matching of mentees to mentors, then we would find a mentee-optimal stable matching M. However, as previously discussed, this would also be a mentor-pessimal stable matching. A similar but reversed situation happens using the (also renamed) Mentor-Oriented Gale-Shapley Algorithm [19]. Therefore we may wish to find a stable matching that is in some sense fair between mentees and mentors using some of the fairness criteria described in the previous section. All the types of fair stable matchings described in Table 3.1 are viable candidates. However, as described in Section 2.2.2.2, each of the problems of finding a balanced stable matching or a sex-equal stable matching is NP-hard, and there are existing polynomial time algorithms in the literature to find only two types of fair stable matchings, namely an egalitarian stable matching (in $O(m^{1.5})$ time) [14] and a minimum regret stable matching (in $O(m)$ time) [23]. Therefore, additional definitions of new, fair stable matchings and polynomial-time algorithms to calculate them provide additional choice for a matching scheme administrator.

Moreover, we may be interested in finding a measure that gives the worst-off mentee a partner of rank as close as possible to that of the worst-off mentor. However, from our experimental work in Section 3.7, we found that there was no other type of optimal stable matching from Table 3.1, that closely approximates the regret-equality score of the regret-equal stable matching. Our results do show, however, that there exist regret-equal stable matchings with balanced score, cost and degree that are close to that of a balanced stable matching, an egalitarian stable matching and a minimum regret stable matching, respectively. This motivates

the search for efficient algorithms to produce a regret-equal stable matching that has "good" measure relative to other types of fair stable matching.

Whilst the practical motivation for studying min-regret sum stable matchings may not be as strong as in the regret-equality case, theoretical motivation comes from completing the study of the algorithmic complexity of computing all types of fair stable matchings relative to cost and degree, as shown in Table 3.1.

3.1.3 Contribution

In this chapter, we present two algorithms to find a regret-equal stable matching in an instance I of SMI. Recall M_0 and M_z are the man-optimal and woman-optimal stable matchings in I. First we present the *Regret-Equal Degree Iteration Algorithm* (REDI), to find a regret-equal stable matching in an instance I of SMI, with time complexity $O(d_0nm)$, where $d_0 = |d_U(M_0) - d_W(M_0)|$. Second we present the *Regret-Equal Stable Pair Algorithm* (RESP), to find a regret-equal stable matching in an instance I of SMI, with time complexity $O(n^4)$. Additionally, we present the *Min-Regret Sum Algorithm* (MRS), to find a min-regret sum stable matching in an instance I of SMI, with time complexity $O(d_sm)$, where $d_s = d_U(M_z) - d_U(M_0)$. In addition to this theoretical work, Algorithms REDI and RESP were implemented and their performance was compared against Gusfield's [23] algorithm to enumerate all stable matchings in $O(m + n|\mathcal{M}_S|)$ time. We found that Algorithm REDI was significantly faster than Algorithm RESP and the enumeration algorithm. Finally, experiments were conducted to compare six different types of optimal stable matchings (balanced, sex-equal, egalitarian, min-regret, regret-equal, min-regret sum), and outputs from Algorithms REDI and RESP, over a range of measures (including balanced score, sex-equal score, cost, degree, regret-equality score, regret sum). In addition to the observations already discussed in Section 3.1.2, we found a large variation in sex-equal scores and regret-equality scores among the six different types of optimal stable matching, and, a far smaller variation for the balanced score, cost, degree and regret sum measures. This smaller variation also includes outputs of Algorithms REDI and RESP, indicating that we are able to find a regret-equal stable matching in polynomial time with a likely good balanced score, cost and degree using these algorithms. Indeed, we find in practice that Algorithm REDI approximates these types of optimal stable matchings at an average of 9.0%, 1.1% and 3.0% over their respective optimal values, for randomly-generated instances with $n = 1000$.

3.1.4 Structure of the chapter

The rest of the chapter is structured as follows. In Section 3.2 we give some preliminary definitions for the chapter. Sections 3.3, 3.4 and 3.6 describe Algorithm REDI, Algorithm

RESP and Algorithm MRS respectively, giving in each case pseudocode, correctness proofs and time complexity calculations. The experimental evaluation is presented in Section 3.7. Finally, conclusions and future work are given in Section 3.8.

3.2 Preliminary definitions

Throughout this chapter we will use the notation and terminology relating to structural aspects of stable matchings first introduced in Section 2.2.2.3. Additionally we also present the following definitions. Let R be the set of rotations of I. Then $R_j(M)$ is the set of rotations that contain a women of rank j in M, that is, $R_j(M) = \{\rho \in R : (m, w) \in \rho \wedge \text{rank}(w, M(w)) = j)\}$. Let M_z be the woman-optimal stable matching [19]. For any stable pair $(m_i, w_j) \notin M_z$, let $\phi(m_i, w_j)$ denote the unique rotation containing pair (m_i, w_j). Let ρ be a rotation in the rotation digraph $R_d(I)$. Denote by $c(\rho)$ the closure of rotation ρ (with respect to the precedence ordering on rotations defined in Section 2.2.2.3). Let $c(R')$ denote $\bigcup_{\rho \in R'} c(\rho)$ where R' is a set of rotations. We say that the closure of an undefined rotation or an empty set of rotations is the empty set.

3.3 Regret-Equal Degree Iteration Algorithm to find a regret-equal stable matching

3.3.1 Description of the Algorithm

Algorithm REDI, which finds a regret-equal stable matching in a given instance I of SMI, is presented as Algorithm 3.1. For an instance I of SMI, Algorithm REDI begins with operations to find the man-optimal and woman-optimal stable matchings, M_0 and M_z, found using the Man-oriented and Women-oriented Gale-Shapley Algorithm [19]. The set of rotations R is also found using the Minimal Differences Algorithm [32].

Let $d'(M_0) = (a_0, b_0)$. If $a_0 = b_0$ then we must have an optimal stable matching and so we output M_0 on Line 5. If $a_0 > b_0$ then any other matching M', where $d'(M') = (a', b')$, must have $a' \geq a_0$ and $b' \leq b_0$ since any rotation (or combination of rotations) eliminated on the man-optimal matching M_0 will make men no better off and women no worse off. Therefore M_0 is optimal and so it is returned on Line 5. Now suppose $a_0 < b_0$. Throughout the algorithm we save the best matching found so far to the variable M_{opt} starting with M_0. We know that a matching exists with $d_0 = b_0 - a_0$ and so we try to improve on this, by finding a matching M with $r(M) < d_0$.

We create several 'columns' of possible degree pairs of a regret-equal matching as follows. The top-most pairs for columns $k \geq 1$ are given by the sequence

$$\big((a_0, b_0), (a_0 + 1, b_0), (a_0 + 2, b_0), ..., (\min\{n, 2b_0 - a_0 - 1\}, b_0)\big).$$

The sequence of pairs for column k ($1 \leq k \leq \min\{2d_0, n - a_0 + 1\}$) from top to bottom is given by

$$\big((a_0+k-1, b_0), (a_0+k-1, b_0-1), (a_0+k-1, b_0-2), ..., (a_0+k-1, \max\{a_0-d_0+k, 1\})\big).$$

At this point as long as the size n of the instance satisfies $n \geq 2b_0-a_0-1$ and $a_0-d_0+1 \geq 1$, the possible degree pairs of a regret-equal matching are shown in Figure 3.1. We know this accounts for all possible degree pairs since, as above, if M' is any matching not equal to M_0, where $d'(M') = (a', b')$, it must be that $a' \geq a_0$ and $b' \leq b_0$. Setting $b' = b_0$, the largest a' could be is given by b_0 added to the maximum possible improved difference $d_0 - 1$, that is, $a' = b_0 + d_0 - 1 = 2b_0 - a_0 - 1$. If $n < 2b_0 - a_0 - 1$ then we only consider the first $n - a_0 + 1$ columns in Figure 3.1. The $a_0 - d_0 + k$ value is obtained by noting that if x is the final value of women's degree for the column sequence above then $a_0 + k - 1 - x = d_0 - 1$ and so $x = a_0 + k - d_0$. Figure 3.2 shows an example of the possible regret-equal degree pairs when $d'(M_0) = (2, 6)$ and $n \geq 9$.

The column operation (Algorithm 3.2) works as follows. Let local variable M hold the current matching for this column, and let local variable Q be the set of rotations corresponding to M. Iteratively we first test if $r(M) < r(M_{opt})$ setting M_{opt} to M if so. We now check whether $d_U(M) \geq d_W(M)$. If it is, then any further rotation for this column will only make $r(M)$ larger, and so we stop iterating for this column, returning M_{opt}. Next, we find the set of rotations Q' in the closure of $R_b(M) \subseteq R$ that are not already eliminated to reach M. If eliminating these rotations would either increase the men's degree or not decrease the women's degree, then we return M_{opt}. Otherwise, set M to be the matching found when eliminating these rotations. If after the column operation, $d_U(M_{opt}) = d_W(M_{opt})$, then we have a regret-equal matching and it is immediately returned on Lines 10 or 24 of Algorithm 3.1.

The column operation described above is called first from the man-optimal stable matching M_0 on Line 8, to iterate down the first column. Then for each man m_i we do the following. Let M be set to M_0. Iteratively we eliminate $(m_i, M(m_i))$ from M by eliminating rotation ρ and its predecessors (not already eliminated to reach M) such that $(m_i, M(m_i)) \in \rho$. We continue doing this until both the men's degree increases and rank$(m_i, M(m_i)) = d_U(M)$ (in the same operation). This has the effect of jumping our focus from some column of possible degree pairs, to another column further to the right with m_i being one of the lowest

ranked men in M. Once we have moved to a new column we perform the column operation described above. If either m_i has the same partner in M as in M_z (hence there are no rotations left that move m_i) or $d_U(M) > d_W(M)$ (further rotations will only increase the regret-equality score), then we stop iterating for m_i. In this case we restart this process for the next man, or return M_{opt} if we have completed this process for all men. Note that since at the end of a while loop iteration, if $r(M) = 0$ then M_{opt} is returned, it is not possible for the condition $d_U(M) = d_W(M)$ to ever be satisfied in the while loop clause.

3.3.2 Correctness Proof

In Theorem 3.3.2 we present the correctness proof for Algorithm REDI.

Proposition 3.3.1. *Let I be an instance of* SMI *and let M and M' be stable matchings in I where for each man $m_i \in U$, rank$(m_i, M(m_i)) \leq$ rank$(m_i, M'(m_i))$ and $d_U(M) = d_U(M')$. Let Q and Q' denote the set of rotations eliminated from M_0 to reach M and M' respectively. Then stable matching M'' with $d'(M'') = d'(M')$ may be found by ensuring all rotations in $R_d = c(\{\rho \in R : \exists (m, w) \in \rho$ where $d_W(M) \geq$ rank$(w, m) > d_W(M')\} \backslash Q)$ are eliminated from M. Figure 3.3 shows a summary of degree pairs for M, M' and M'' in the general case.*

Proof. First, since rank$(m_i, M(m_i)) \leq$ rank$(m_i, M'(m_i))$ and each rotation must make some man worse off, we know that each rotation in Q must be eliminated to reach M' and therefore $Q \subseteq Q'$. Second, we also know that any rotation containing a woman at rank larger than $d_W(M')$ must be eliminated from M in order to reach M' (additional rotations may also have been eliminated). But these are precisely the rotations in R_d, hence $R_d \subseteq Q'$. Let M'' be the stable matching found when eliminating R_d on M and let $Q'' = Q \cup R_d$ denote the unique set of rotations corresponding to M''. Then $Q'' \subseteq Q'$ since $Q \subseteq Q'$ and $R_d \subseteq Q'$.

We observe the following.

- Since $Q \subseteq Q''$, it must be that $d_U(M) \leq d_U(M'')$, and as $d_U(M) = d_U(M')$, it follows that $d_U(M') \leq d_U(M'')$;
- $Q \cup R_d$ is the set of all rotations with pairs containing women of rank larger than $d_W(M')$. Since $Q'' = Q \cup R_d$ and all rotations in Q'' are eliminated on M_0 to reach M'', it must be the case that $d_W(M'') \leq d_W(M')$;
- Since $Q'' \subseteq Q'$, it must be that $d_U(M'') \leq d_U(M')$ and $d_W(M'') \geq d_W(M')$.

Hence $d'(M'') = d'(M')$ as required. □

$r(M)$	Degree pairs $(d_U(M), d_W(M))$								
k	1	2	...	$b_0 - a_0$	$b_0 - a_0 + 1$	$b_0 - a_0 + 2$	...	$2b_0 - 2a_0 - 1$	$2b_0 - 2a_0$
$d_0 = b_0 - a_0$	(a_0, b_0)								
$b_0 - a_0 - 1$	$(a_0, b_0 - 1)$	$(a_0 + 1, b_0)$							
...	...	...							
1	$(a_0, a_0 + 1)$	$(a_0 + 1, a_0 + 2)$	...	$(b_0 - 1, b_0)$					
0	(a_0, a_0)	$(a_0 + 1, a_0 + 1)$	...	...	(b_0, b_0)				
1	$(a_0, a_0 - 1)$	$(a_0 + 1, a_0)$	...	...	...	$(b_0 + 1, b_0)$			
...	...	...	...	...	...	...			
$b_0 - a_0 - 2$	$(a_0, a_0 - d_0 + 2)$	$(a_0 + 1, a_0 - d_0 + 3)$	...	...	...	...	...	$(2b_0 - a_0 - 2, b_0)$	
$b_0 - a_0 - 1$	$(a_0, a_0 - d_0 + 1)$	$(a_0 + 1, a_0 - d_0 + 2)$	...	...	...	...	...	...	$(2b_0 - a_0 - 1, b_0)$

Figure 3.1: Possible regret-equal degree pairs when $d'(M_0) = (a_0, b_0)$, $n \geq 2b_0 - a_0 - 1$ and $a_0 - d_0 + 1 \geq 1$.

$r(M)$	Degree pairs $(d_U(M), d_W(M))$							
k	1	2	3	4	5	6	7	8
$d_0 = 4$	$(2, 6)$							
3	$(2, 5)$	$(3, 6)$						
2	$(2, 4)$	$(3, 5)$	$(4, 6)$					
1	$(2, 3)$	$(3, 4)$	$(4, 5)$	$(5, 6)$				
0	$(2, 2)$	$(3, 3)$	$(4, 4)$	$(5, 5)$	$(6, 6)$			
1	$(2, 1)$	$(3, 2)$	$(4, 3)$	$(5, 4)$	$(6, 5)$	$(7, 6)$		
2		$(3, 1)$	$(4, 2)$	$(5, 3)$	$(6, 4)$	$(7, 5)$	$(8, 6)$	
3			$(4, 1)$	$(5, 2)$	$(6, 3)$	$(7, 4)$	$(8, 5)$	$(9, 6)$

Figure 3.2: Possible regret-equal degree pairs when $d'(M_0) = (2, 6)$ and $n \geq 9$.

Algorithm 3.1 REDI(I), returns a regret-equal stable matching for an instance I of SMI.

Require: An instance I of SMI.
Ensure: Return a regret-equal stable matching M_{opt}.

1: $M_0 \leftarrow$ MGS(I) ▷ M_0 is the man-optimal stable matching found using the Man-oriented Gale-Shapley Algorithm (MGS) [19].
2: $M_z \leftarrow$ WGS(I) ▷ M_z is the woman-optimal stable matching found using the Woman-oriented Gale-Shapley Algorithm (WGS) [19].
3: $R \leftarrow$ Min-Diff(I) ▷ R is the set of rotations found using the Minimal Differences Algorithm (Min-Diff) [32].
4: **if** $d_U(M_0) \geq d_W(M_0)$ **then**
5: **return** M_0
6: **end if**
7: $M_{opt} \leftarrow M_0$ ▷ M_{opt} is the best stable matching found so far.
8: $M_{opt} \leftarrow$REDI-Col($I, M_0, \emptyset, M_{opt}$) ▷ Find the best matching for the first column.
9: **if** $r(M_{opt}) = 0$ **then**
10: **return** M_{opt}
11: **end if**
12: **for each** $m_i \in U$ **do** ▷ For each man.
13: $M \leftarrow M_0$ ▷ M is the matching we start from for m_i at the beginning of each column.
14: $Q \leftarrow \emptyset$ ▷ Q is the set of rotations corresponding to M.
15: **while** $(m_i, M(m_i)) \notin M_z$ **and** $d_U(M) < d_W(M)$ **do**
16: $\rho = \phi(m_i, M(m_i))$
17: $a \leftarrow d_U(M)$
18: $Q' \leftarrow c(\rho) \backslash Q$
19: $M \leftarrow M/Q'$ ▷ Rotations in Q' are eliminated in order defined by the rotation poset of I.
20: $Q \leftarrow Q \cup Q'$
21: **if** $d_U(M) > a$ **and** rank$(m_i, M(m_i)) = d_U(M)$ **then** ▷ The men's degree has increased and m_i is a worst ranked man in M.
22: $M_{opt} \leftarrow$REDI-Col(I, M, Q, M_{opt}) ▷ Find the best matching for this column.
23: **if** $r(M_{opt}) = 0$ **then**
24: **return** M_{opt}
25: **end if**
26: **end if**
27: **end while**
28: **end for**
29: **return** M_{opt}

Algorithm 3.2 REDI-Col(I, M, Q, M_{opt}), subroutine for Algorithm 3.1. Column operation for the current column $d_U(M)$. Returns M_{opt}, the best stable matching found so far (according to the regret-equality score).

Require: An instance I of SMI, stable matching M, the closure of M, Q and M_{opt} the best stable matching found so far (according to the regret-equality score).

Ensure: Finds the best stable matching (according to the regret-equality score) found when incrementally eliminating women of worst rank from the current matching, without increasing the men's degree. If an improvement is made then M_{opt} is updated. M_{opt} is returned. All variables used within Algorithm 3.2 are understood to be local.

1: $a \leftarrow d_U(M)$
2: **while true do**
3: **if** $r(M) < r(M_{opt})$ **then**
4: $M_{opt} \leftarrow M$
5: **end if**
6: **if** $d_U(M) \geq d_W(M)$ **then** ▷ Further rotations for this column would only increase the difference in degree of men and women.
7: **return** M_{opt}
8: **end if**
9: $b \leftarrow d_W(M)$
10: $Q' \leftarrow c(R_b(M)) \backslash Q$
11: **if** $d_U(M/Q') > a \vee d_W(M/Q') = b$ **then**
12: **return** M_{opt}
13: **else**
14: $M \leftarrow M/Q'$ ▷ Rotations in Q' are eliminated in order defined by the rotation poset of I.
15: $Q \leftarrow Q \cup Q'$
16: **end if**
17: **end while**

$$
\begin{aligned}
&(a_0 + k - 1, b_0)\\
&(a_0 + k - 1, b_0 - 1)\\
&(a_0 + k - 1, b_0 - 2)\\
&\ldots\\
&d'(M)\\
&\ldots\\
&d'(M') = d'(M'')\\
&\ldots\\
&(a_0 + k - 1, \max\{a_0 - d_0 + k, 1\})
\end{aligned}
$$

Figure 3.3: Degree pairs in column $k = d_U(M)$ for instance I, built as per the description in Section 3.3.1.

Theorem 3.3.2. *Let I be an instance of* SMI. *Any matching produced by Algorithm* REDI *is a regret-equal stable matching of I.*

Proof. There are four points in Algorithm 3.1's execution where we return a matching, namely Lines 5, 10, 24 and 29. First we show that if M is a matching returned at any of these points then M is stable. Next we look at each of these points where a matching may be returned and show they are regret-equal stable matchings.

Let M be the matching returned at any of the four points above. Then M is stable since it is found by iteratively eliminating sets of rotations that form closed subsets of the rotation poset of I (on Line 19 of Algorithm 3.1 and Line 14 of Algorithm 3.2) starting from the man-optimal stable matching (created on Line 1). Since there is a 1-1 correspondence between closed subsets of the rotation poset and the set of all stable matchings [32, Theorem 3.1], M is a stable matching.

Let M_1 be a matching that is returned on Line 5. Since M_1 has been returned on Line 5 it must be that $d_U(M_1) \geq d_W(M_1)$ and therefore by the same reasoning given in the second paragraph of Section 3.3.1, M_1 is a regret-equal stable matching.

Let M_2 be a matching that is returned by either Line 10 or Line 24. To be returned at these points $r(M_2) = 0$ and therefore M_2 is a regret-equal stable matching.

Let M_3 be a matching that is returned on Line 29 and let M' be a regret-equal stable matching such that $d_U(M')$ is minimum over all regret-equal stable matchings. We will prove that $r(M_3) = r(M')$ by showing it will not have been possible for us to miss a stable matching with regret-equality score equal to $r(M')$ during the algorithm's execution. First we show that the column operation (Algorithm 3.2) must be executed for column $d_U(M')$. Then, we show that during this column operation M_{opt} will be updated such that $r(M_{opt}) = r(M')$.

Since M' is not returned on Line 5, $M' \neq M_0$. If $d_U(M') = d_U(M_0)$ then clearly the column operation is executed on Line 8 for column $d_U(M')$. Assume therefore that $d_U(M') \neq d_U(M_0)$. Without loss of generality let $\{m_1, m_2..., m_k\}$ be the set of men who are at rank $d_U(M')$ in M'. We enter the while loop on Line 15 for each man $m_j \in \{m_1, m_2..., m_k\}$. M is initialised to M_0. Successively, the algorithm eliminates all rotations in $c(\phi(m_j, M(m_j)))$ that are not yet eliminated until $M(m_j) = M'(m_j)$. This must be possible since $(m_j, M'(m_j)) \in M'$ and M' is a stable matching in I. During the while loop iteration that rotates m_j down to his $M'(m_j)$ partner, it is not necessarily the case that the current matching M updates its man-degree to $d_U(M')$ at this point. This is because although $d_U(M) = d_U(M')$, an earlier movement of m_j in a previous while loop iteration may have brought some other man in $\{m_1, m_2..., m_k\}$ down to his partner in M' already. Note that it is not possible for a man to overshoot column $d_U(M')$ when moving m_j down to his $M'(m_j)$ partner since the set of rotations we have eliminated to bring m_i down to $M'(m_i)$ is a subset of the rotations cor-

responding to M'. However, for at least one of these men $m_i \in \{m_1, m_2..., m_k\}$, the while loop iteration that moves m_i down to $M'(m_i)$ will also lead to $d_U(M)$ increasing to the same value as $d_U(M')$. Assume we are beginning the while loop iteration on Line 15 for man m_i. Let $M = M_0$. We continue eliminating rotations that have not yet been eliminated in $c(\phi(m_i, M(m_i)))$ until $M(m_i) = M'(m_i)$. Our choice of m_i ensures that the movement of m_i to his $M'(m_i)$ partner occurs at the same time as $d_U(M)$ increases to $d_U(M')$ and so we satisfy the conditions on Line 21 to perform the column operation (Algorithm 3.2) on M for column $d_U(M')$.

From above we know that the column operation (Algorithm 3.2) will be executed for column $d_U(M')$. Either we start this column operation with M_0, or we have only eliminated the minimum number of rotations necessary to take m_i down to $M'(m_i)$ from M_0. In either case we know that rank$(m_l, M(m_l)) \leq$ rank$(m_l, M'(m_l))$ for all $m_l \in U$. For this column a regret-equal stable matching may be found with degree pairs that are either $(d_U(M'), d_U(M') + r(M'))$ or $(d_U(M'), d_U(M') - r(M'))$. The degree pair of matching M' is given by one of the above pairs, but it may be possible for regret-equal stable matchings to exist in I with both of the above degree pairs. Assume first that $d'(M') = (d_U(M'), d_U(M') + r(M'))$. The algorithm will attempt to successively eliminate from M rotations containing women of rank $d_W(M)$. By Proposition 3.3.1, since rank$(m_i, M(m_i)) \leq$ rank$(m_i, M'(m_i))$ for all $m_i \in U$ and $d_U(M) = d_U(M')$, we know the algorithm will continue this process until M is updated to a regret-equal stable matching with $d'(M) = d'(M')$ and so M_{opt} will be set to M with $r(M_{opt}) = r(M')$. Assume then that $d'(M') = (d_U(M'), d_U(M') - r(M'))$, where a regret-equal stable matching may or may not exist with degree pair $(d_U(M'), d_U(M') + r(M'))$. Then similar to before, the algorithm successively eliminates from M rotations containing women of rank $d_W(M)$. This may result in a regret-equal stable matching M being found with $d'(M) = (d_U(M'), d_U(M') + r(M'))$ in which case M_{opt} will be set to M with $r(M_{opt}) = r(M')$. Assume this is not the case. Then, by Proposition 3.3.1, since rank$(m_i, M(m_i)) \leq$ rank$(m_i, M'(m_i))$ for all $m_i \in U$ and $d_U(M) = d_U(M')$, we know the algorithm will continue eliminating rotations containing women of rank $d_W(M)$ until M is updated to a regret-equal stable matching with $d'(M) = d'(M')$ and so M_{opt} will be set to M with $r(M_{opt}) = r(M')$ as above.

Therefore any matching returned by Algorithm REDI is a regret-equal stable matching. □

3.3.3 Time complexity

Theorem 3.3.3. *Let I be an instance of* SMI. *Algorithm* REDI *always terminates within $O(d_0 nm)$ time, where $d_0 = |d_U(M_0) - d_W(M_0)|$, n is the number of men or women in I, m is the total length of all preference lists and M_0 is the man-optimal stable matching.*

Proof. Algorithm 3.1 begins by calculating the man-optimal stable matching and woman-optimal stable matching using the Gale-Shapley Algorithms in $O(m)$ time [19], and the set of all rotations using the Minimal Differences Algorithm in $O(m)$ time [32]. The for loop on Line 12 of Algorithm 3.1 iterates over all men, n times, where n is the number of men or women. During the nested while loop on Line 15, each man m_i may be rotated down his preference list on Line 19, $2d_0 - 1$ times (the maximum possible number of columns from Figure 3.1 minus 1). Rotations are eliminated on Line 19 of Algorithm 3.1 and Line 14 of Algorithm 3.2 successively, beginning at the man-optimal stable matching M_0, meaning $O(m)$ rotations are eliminated in total for each while loop iteration at Line 15 of Algorithm 3.1. This may also be viewed as $O(m)$ man-woman pair changes since the number of possible man-woman pairs in I is $O(m)$ and each pair existing in the set of rotations is unique. Therefore Algorithm REDI runs in $O(d_0nm)$ time and since preference lists of men and women are finite, the algorithm terminates. □

3.4 Regret-Equal Stable Pair Algorithm to find a regret-equal stable matching

3.4.1 Description of the Algorithm

For an instance I of SMI, Algorithm 3.3 begins by finding the man-optimal and woman-optimal stable matchings, M_0 and M_z, using the Man-oriented and Women-oriented Gale-Shapley Algorithms [19]. Also, we find the set of all rotations R of I using the Minimal Differences Algorithm [32], the rotation digraph $R_d(I)$ and the set of stable pairs S of I. The transitive closure of $R_d(I)$, denoted Π, is created by finding all rotations reachable in $R_d(I)$ from each rotation $\rho \in R$ [25, p. 115]. For each pair $(m_i, w_j) \in S \backslash M_0$, we construct $\tau(m_i, w_j)$ such that $\tau(m_i, w_j)$ denotes the rotation that moves man m_i to woman w_j. Similarly, for each pair $(m_i, w_j) \in S \backslash M_z$, we construct $\phi(m_i, w_j)$ such that $\phi(m_i, w_j)$ denotes the rotation that moves m_i from w_j.

The possible degree pairs of a regret-equal matching in I are given by

$$P = \{d_U(M_0), d_U(M_0) + 1, ..., d_U(M_z)\} \times \{d_W(M_z), d_W(M_z) + 1, ..., d_W(M_0)\}.$$

We define the following order $\prec$ over P.

Definition 3.4.1. *Let P be the set of degree pairs as defined above. Then the relation $\prec$ over*

P is given by the following.

$$(a,b) \prec (a',b') \in P \quad \textit{iff} \quad |a-b| < |a'-b'|$$
$$\textit{or} \quad |a-b| = |a'-b'| \quad \textit{and} \quad a+b < a'+b'$$

Note that an increasing ordering of elements over the $\prec$ relation is consistent with an increasing regret-equality score.

During the main for loop of Algorithm 3.3 on Line 11, we iterate through the possible degree pairs of P' in order, where P' is given by the pairs in P sorted in increasing order according to the $\prec$ relation above. For any pair $(a,b) \in P'$ that is considered according to this order, we immediately return any stable matching M_{opt} such that particular conditions (described below) hold with $d_U(M_{opt}) = a$ and $d_W(M_{opt}) = b$. Therefore as P' is sorted with respect to an increasing regret-equality score, we ensure that a regret-equal stable matching is returned. The minimisation of $a + b$ is not necessary when finding a regret-equal stable matching, however it seems reasonable to assume that a stable matching M' with degree pair $d'(M') = (1,2)$, for example, would be preferred to a stable matching M'' with degree pair $d'(M'') = (9,10)$, despite both having the same regret-equality score. As an example, the following provides the order of iteration over degree pairs for a complete instance of size 3 where $d_U(M_0) = d_W(M_z) = 1$ and $d_U(M_z) = d_W(M_0) = 3$.

$$(1,1),(2,2),(3,3),(1,2),(2,1),(2,3),(3,2),(1,3),(3,1)$$

For each of these degree pairs we first truncate instance I at the given degree pair (a,b), resulting in truncated instance I_T. The procedure for this is given in Algorithm 3.4 and comprises the truncation of all men's and women's preference lists below rank a and b respectively. The preference lists are then made consistent by removing any woman w_j from man m_i's list where w_j does not rank m_i and vice versa. Truncation is performed, since we know that in a stable matching with degree pair (a,b), no man (respectively woman) can be assigned a partner lower down their preference list than rank a (respectively b). Therefore, when seeking a stable matching with degree pair (a,b), truncation will not have an effect on whether or not a stable matching of this type can be found, but gives the added benefit of reducing the size of the set of stable matchings compared to the original instance. Next the man-optimal stable matching M_0^T of I_T is found. If $|M_0^T| < n$ then we end this iteration of the main for loop since there can be no stable matching of I_T that is also a stable matching of I. However, if $|M_0^T| = n$ then we do the following. First, find the woman-optimal stable matching M_z^T of I_T. Next, we find all rotations R_F in R that contain a pair with the man's rank no better than that of his partner in M_0^T and better than that of his partner in M_z^T. In Proposition 3.4.3 we show that if a rotation contains one pair within this range, then all pairs

are within this range. The stable pairs S_F are then defined to be the union of all pairs in R_F and all pairs in M_z^T.

Let R_T and S_T denote the set of all rotations and stable pairs of I_T, respectively. Later we will prove than $S_F = S_T$ and so S_F is also the set of all stable pairs in I_T. For each pair of (possibly the same) stable pairs $(m_1, w_1), (m_2, w_2) \in S_F$, where rank$(m_1, w_1) = a$ and rank$(w_2, m_2) = b$ (with rank calculated according to I), we test whether (m_1, w_1) and (m_2, w_2) exist in the same stable matching of I by checking that $\phi(m_2, w_2)$ does not precede $\tau(m_1, w_1)$, (assuming both rotations are defined). Note that $\tau(m_1, w_1)$ may not be defined if $(m_1, w_1) \in M_0^T$ and $\phi(m_2, w_2)$ may not be defined if $(m_2, w_2) \in M_z^T$. Since $(m_1, w_1) \in S_T$, there is a stable matching $M_1 \in I_T$ with $(m_1, w_1) \in M_1$. We know rank$(m_2, w_2) = b$ and $d_W(M_1) \leq b$, so $\tau(m_2, w_2)$ must have been eliminated to reach M_1, assuming $\tau(m_2, w_2)$ is defined. But $(m_1, w_1) \in M_1$, and hence it is not possible for $\phi(m_1, w_1)$ to precede $\tau(m_2, w_2)$, assuming both these rotations are defined. Thus this condition, described in Gusfield and Irving's book [25, p. 115], is not tested. If the condition of Line 21 is not satisfied, then there does not exist a stable matching containing pairs (m_1, w_1) and (m_2, w_2) in I, and so we move on to the next pair of stable pairs. If a stable matching does exist that satisfies this condition, then we find the rotation (if it is defined) that creates pair (m_1, w_1), namely $\tau(m_1, w_1)$, and the set of rotations R_G that contains any pair with a woman of rank $> b$. Then, the regret-equal stable matching M_{opt} is found by eliminating $c(\tau(m_1, w_1)) \cup c(R_G)$ on M_0.

3.4.2 Correctness Proof

In the proofs of this section we will use the following notation and terminology. Let I be an instance of SMI. Let I_T be the truncated instance of I where men are truncated below rank a and women below rank b according to Algorithm 3.4, let $\mathcal{M}_S^T$ be the set of stable matchings in I_T, and let S_T be the set of stable pairs in I_T. In all cases, the ranks of men on women's preference lists (and vice versa) are all calculated with respect to instance I. Recall that $\mathcal{M}_S$ is the set of stable matchings in I. Then, let

$$\text{reduced}(\mathcal{M}_S) = \{M \in \mathcal{M}_S : \forall (m_i, w_j) \in M, \text{rank}(m_i, w_j) \leq a \wedge \text{rank}(w_j, m_i) \leq b\}.$$

In this section we begin by showing that the set of stable pairs in rotations of the truncated instance I_T, and the set of stable pairs in the filtered rotations of I created on Line 16 of Algorithm 3.3, are equal. We then show that if two of these stable pairs (m_1, w_1) and (m_2, w_2), where rank$(m_1, w_1) = a$ and rank$(w_2, m_2) = b$, can coexist in a stable matching, then it must be the case that they can coexist in a stable matching with degree pair (a, b). Finally, we prove that when a matching is returned it must be a regret-equal stable matching.

Algorithm 3.3 RESP(I), returns a regret-equal stable matching for an instance I of SMI.

Require: An instance I of SMI.
Ensure: Return a regret-equal stable matching M_{opt}.

$M_0 \leftarrow$ MGS(I) ▷ M_0 is the man-optimal stable matching found using the Man-oriented Gale-Shapley Algorithm (MGS) [19].
$M_z \leftarrow$ WGS(I) ▷ M_z is the woman-optimal stable matching found using the Woman-oriented Gale-Shapley Algorithm (WGS) [19].
$R \leftarrow$ Min-Diff(I) ▷ R is the set of rotations found using the Minimal Differences Algorithm (Min-Diff) [32].
$S \leftarrow \{(m_i, w_j) : (m_i, w_j) \in M_z \vee (m_i, w_j) \in R\}$
Construct the rotation digraph $R_d(I)$ of I
Construct transitive closure Π of $R_d(I)$, so that $\Pi(\rho_i, \rho_j) = 1$ iff ρ_i precedes ρ_j in $R_d(I)$ ▷ [25, p. 115]
Construct rotation $\tau(m_i, w_j)$ for each $(m_i, w_j) \in S \backslash M_0$ ▷ [25, p. 115]
Construct rotation $\phi(m_i, w_j)$ for each $(m_i, w_j) \in S \backslash M_z$ ▷ [25, p. 115]
$P \leftarrow \{d_U(M_0), d_U(M_0) + 1, ..., d_U(M_z)\} \times \{d_W(M_z), d_W(M_z) + 1, ..., d_W(M_0)\}$
$P' \leftarrow$ sort$(P, \prec)$
for each $(a, b) \in P'$ in order **do**
    $I_T \leftarrow$RESP-Truncate$(I, (a, b))$
    $M_0^T \leftarrow$ MGS(I_T) ▷ M_0^T is the man-optimal stable matching found using the Man-oriented Gale-Shapley Algorithm (MGS) [19].
    if $|M_0^T| = n$ **then**
        $M_z^T \leftarrow$ WGS(I_T) ▷ M_z^T is the woman-optimal stable matching found using the Woman-oriented Gale-Shapley Algorithm (WGS) [19].
        $R_F \leftarrow$ RESP-Filter(I, R, M_0^T, M_z^T)
        $S_F \leftarrow \{(m_i, w_j) : (m_i, w_j) \in M_z^T \vee (m_i, w_j) \in R_F\}$
        for each $\{\{(m_1, w_1), (m_2, w_2)\} \subseteq S_F$: rank$(m_1, w_1) = a$, rank$(w_2, m_2) = b\}$ **do** ▷ Ranks are calculated with respect to I.
            $\tau_1 \leftarrow \tau(m_1, w_1)$
            $\phi_2 \leftarrow \phi(m_2, w_2)$
            if τ_1 is undefined **or** ϕ_2 is undefined **or** $\Pi(\phi_2, \tau_1) \neq 1$ **then**
                Let R_G be the set of rotations in I containing any pair with a woman of rank $> b$.
                $Q \leftarrow c(\tau(m_1, w_1)) \cup c(R_G)$
                $M_{opt} \leftarrow M_0/Q$ ▷ Rotations in Q are eliminated in order defined by the rotation poset of I_T.
                return M_{opt}
            end if
        end for
    end if
end for

Algorithm 3.4 RESP-Truncate$(I, (a, b))$, subroutine for Algorithm 3.3. Truncates the given instance I at the given degrees a and b for men and women respectively. Returns truncated instance I.

Require: An instance I of SMI and integers a and b such that $1 \leq a \leq n$ and $1 \leq b \leq n$.
Ensure: Return truncated instance I.

for $i \in \{1, 2, ..., n\}$ **do**
  pref$_T(m_i) \leftarrow$ copy of first $\min\{a, |\text{pref}_T(m_i)|\}$ elements of pref(m_i) ▷ pref(m_i) is the preference list of man m_i
  pref$_T(w_i) \leftarrow$ copy of first $\min\{b, |\text{pref}_T(w_i)|\}$ elements of pref(w_i) ▷ pref(w_i) is the preference list of woman w_i
end for
for $i \in \{1, 2, ..., n\}$ **do**
  for $w_j \in \text{pref}_T(m_i)$ **do**
    if m_i is not in pref$_T(w_j)$ **then**
      Remove w_j from pref$_T(m_i)$ ▷ Size of pref$_T(m_i)$ is reduced by 1
    end if
  end for
  for $m_j \in \text{pref}_T(w_i)$ **do**
    if w_i is not in pref$_T(m_j)$ **then**
      Remove m_j from pref$_T(w_i)$ ▷ Size of pref$_T(w_i)$ is reduced by 1
    end if
  end for
end for
return I

Algorithm 3.5 RESP-Filter(I, R, M_0^T, M_z^T), subroutine for Algorithm 3.3. Filters rotations in R, removing rotations which contain pairs with men's rank better than in M_0^T or create pairs with men's rank worse than in M_z^T. Returns R_F, the filtered rotations.

Require: An instance I of SMI, the set of all rotations R, and stable matchings M_0^T and M_z^T such that for all $m_i \in U$, rank$(m_i, M_0^T(m_i)) \leq$ rank$(m_i, M_z^T(m_i))$.
Ensure: Return the filtered rotations R_F.

$R_F \leftarrow \emptyset$
for $\rho \in R$ **do**
  if $\exists (m_i, w_j) \in \rho$, rank$(m_i, M_0^T(m_i)) \leq$ rank$(m_i, w_j) <$ rank$(m_i, M_z^T(m_i))$ **then**
    $R_F \leftarrow R_F \cup \{\rho\}$
  end if
end for
return R_F

First, in Lemma 3.4.2, we show that the set of stable matchings in I_T is equal to reduced($\mathcal{M}_S$), as defined above.

Lemma 3.4.2. *$\mathcal{M}_S^T$ = reduced($\mathcal{M}_S$).*

Proof. Let stable matching M' of size n exist in $\mathcal{M}_S^T$. If we transform I_T to I then we are adding preference list pairs (m_i, w_j) where either rank$(m_i, w_j) >$ rank$(m_i, M'(m_i))$ or rank$(w_j, m_i) >$ rank$(w_j, M'(w_j))$, or both. In all cases (m_i, w_j) cannot constitute a blocking pair and so M' must be stable in I with $M' \in \mathcal{M}_S$. Also, since M' is in I_T it must be the case that rank$(m_i, w_j) \leq a$ and rank$(w_j, m_i) \leq b$. Hence $\mathcal{M}_S^T \subseteq$ reduced$(\mathcal{M}_S)$.

Let stable matching M'' exist in reduced$(\mathcal{M}_S)$. By the definition of reduced$(\mathcal{M}_S)$ and I_T all pairs in M'' must exist in preference lists of the truncated instance I_T. If we transform I to I_T we will only be removing some pairs from preference lists that do not exist in M''. Therefore since we are only removing pairs, it is not possible to introduce pairs into I_T that would block M'' and so M'' is stable in I_T. Hence $\mathcal{M}_S^T \supseteq$ reduced$(\mathcal{M}_S)$.

Therefore $\mathcal{M}_S^T =$ reduced$(\mathcal{M}_S)$, as required. □

Next, Proposition 3.4.3 proves that it is not possible for a rotation to contain a pair of stable pairs, such that for a given stable matching M, the man of one stable pair prefers his partner in the rotation to his partner in M and the man of the other stable pair does not. Essentially, this shows that stable pairs of rotations are bounded by stable pairs of stable matchings.

Proposition 3.4.3. *Let M be any stable matching in I and let R be the set of rotations in I. Then it is not possible for any rotation $\rho \in R$ to contain pairs (m_1, w_1) and (m_2, w_2) such that rank$(m_1, w_1) <$ rank$(m_1, M(m_1))$ and rank$(m_2, w_2) \geq$ rank$(m_2, M(m_2))$.*

Proof. Assume for contradiction that ρ contains the pairs (m_1, w_1) and (m_2, w_2) described in the statement of this proposition. Since ρ contains pair (m_1, w_1) where rank$(m_1, w_1) <$ rank$(m_1, M(m_1))$, we know ρ must exist in the set of rotations corresponding to M. Since ρ must be eliminated to reach M and contains pair (m_2, w_2) with rank$(m_2, w_2) \geq$ rank$(m_2, M(m_2))$, we know that $(m_2, M(m_2))$ cannot exist in M, a contradiction. □

In Lemma 3.4.4, we show that S_F, created on Line 16 of Algorithm 3.3, is equal to the set of stable pairs of I_T.

Lemma 3.4.4. *Let S_{R_T} be the set of all stable pairs in rotations R_T of I_T. Let R_F denote the set of filtered rotations created on Line 16 of Algorithm 3.3, and let S_{R_F} denote the set of stable pairs in rotations of R_F. Then, $S_{R_T} = S_{R_F}$.*

Furthermore $S_T = S_F$ where S_T is the set of all stable pairs in I_T and S_F is the set of stable pairs created on Line 17 of Algorithm 3.3.

Proof. Let R denote all rotations of I. Let M_0^T and M_z^T denote the man-optimal and woman-optimal stable matchings of I_T respectively. Then M_0^T and M_z^T are also in reduced($\mathcal{M}_S$), by Lemma 3.4.2. Let R_0^T and R_z^T be the set of rotations associated with M_0^T and M_z^T in I respectively. Since M_0^T is the man-optimal stable matchings in I_T, we know that for any other stable matching $M \in \mathcal{M}_S^T \backslash M_0^T$, rank$(m_i, M_0^T(m_i)) \leq$ rank$(m_i, M(m_i))$ for each m_i in I_T. By Lemma 3.4.2, $\mathcal{M}_S^T =$ reduced($\mathcal{M}_S$) and so we also know that in I, for any other stable matching $M \in$ reduced($\mathcal{M}_S$)$\backslash M_0^T$, rank$(m_i, M_0^T(m_i)) \leq$ rank$(m_i, M(m_i))$ for each m_i. It follows then that in I, at least every rotation in R_0^T must have been eliminated on M_0 to reach any other stable matching in reduced($\mathcal{M}_S$)$\backslash M_0^T$.

We first show that $S_{R_T} \subseteq S_{R_F}$. From above we know there must exist a minimum set of rotations $R' \subseteq R$, subsets of which take us from M_0^T to any other stable matchings in reduced($\mathcal{M}_S$)$\backslash M_0^T$. Let S' be the set of stable pairs in R'. By definition, S_{R_T} is the set of stable pairs involved in rotations R_T of I_T. It is therefore also equal to the set of stable pairs involved in stable matchings of $\mathcal{M}_S^T \backslash M_z^T$. Hence S_{R_T} is the minimum set of stable pairs of rotations that can take us from M_0^T to all other stable matchings in $\mathcal{M}_S^T \backslash M_0^T$, but then as $\mathcal{M}_S^T =$ reduced($\mathcal{M}_S$), each of the pairs in S_{R_T} must also exist in the set of rotations R' and so we have $S_{R_T} \subseteq S'$. Since R' was defined to be a minimum set of rotations, we know that every rotation in R' contains at least one stable pair (m_i', w_j') with rank$(m_i', M_0^T(m_i')) \leq$ rank$(m_i', w_j') <$ rank$(m_i', M_z^T(m_i'))$. But then, by Proposition 3.4.3, any rotation of this type can only contain other stable pairs also within this range. Since S_{R_F} is precisely the set of stable pairs of all rotations where some stable pair exists in the range described above, $S' \subseteq S_{R_F}$. Finally, as $S_{R_T} \subseteq S'$ we have $S_{R_T} \subseteq S_{R_F}$.

Now we show that $S_{R_F} \subseteq S_{R_T}$. Assume that there is some stable pair (m_k, w_l) in S_{R_F}. We aim to prove that $(m_k, w_l) \in S_{R_T}$. Since $(m_k, w_l) \in S_{R_F}$, $\phi(m_k, w_l)$, the rotation that moves m_k from w_l, must exist in R_F. Thus $\phi(m_k, w_l)$ contains at least one stable pair $(m_i', w_j') \in \phi(m_k, w_l)$ such that rank$(m_i', M_0^T(m_i')) \leq$ rank$(m_i', w_j') <$ rank$(m_i', M_z^T(m_i'))$. Then by Proposition 3.4.3, all stable pairs of $\phi(m_k, w_l)$ are within this range, and so we have rank$(m_k, M_0^T(m_k)) \leq$ rank$(m_k, w_l) <$ rank$(m_k, M_z^T(m_k))$. Since (m_k, w_l) is a stable pair and rank$(m_k, w_l) <$ rank$(m_k, M_z^T(m_k))$, it must be the case that $c(\tau(m_k, w_l)) \subseteq R_z^T$, where rotation $\tau(m_k, w_l)$ may or may not be defined in I. From the first paragraph in this proof, we also know that in I, $R_0^T \subseteq R_z^T$, and so $R_0^T \cup c(\tau(m_k, w_l)) \subseteq R_z^T$. Let M' be the matching found when $R_0^T \cup c(\tau(m_k, w_l))$ is eliminated on M_0. Then for all men m_i, rank$(m_i, M'(m_i)) \leq$ rank$(m_i, M_z^T(m_i))$ meaning $d_U(M') \leq a$. Also, since $R_0^T \subseteq R_0^T \cup c(\tau(m_k, w_l))$, for all women w_j, rank$(w_j, M'(w_j)) \leq$ rank$(w_j, M_0^T(w_j))$ meaning $d_W(M') \leq b$. Therefore, we have $M' \in$ reduced($\mathcal{M}_S$), and so by Lemma 3.4.2, $M' \in \mathcal{M}_S^T$. Since we also know from above that rank$(m_k, w_l) <$ rank$(m_k, M_z^T(m_k))$, it must be that $(m_k, w_l) \notin M_z^T$ and so $(m_k, w_l) \in S_{R_T}$. Hence $S_{R_F} \subseteq S_{R_T}$.

Therefore $S_{R_T} = S_{R_F}$, as required.

Recall S_F is the set of stable pairs created on Line 17 of Algorithm 3.3, and comprises the union of all stable pairs in S_{R_F} and M_z^T. Since $S_{R_T} = S_{R_F}$ from above, and M_z^T exists in both I_T and I, it follows that $S_F = S_T$. □

In Lemma 3.4.5, we prove that if two stable pairs $\{(m_1, w_1), (m_2, w_2)\}$ exist in S_F, with rank$(m_1, w_1) = a$ and rank$(w_2, m_2) = b$, then there must exist a stable matching in the original instance I with degree pair (a, b).

Lemma 3.4.5. *Let M_{opt} be the matching returned on Line 25 of Algorithm 3.3 and let I_T be the truncated instance of I created on Line 12 during the final iteration of the for loop on Line 11. Finally, let (m_1, w_1) and (m_2, w_2) be the stable pairs in S_F with rank$(m_1, w_1) = a$ and rank$(w_2, m_2) = b$, iterated over in the final iteration of the for loop on Line 18. Then, $M_{opt} \in \mathcal{M}_S$ with $d'(M_{opt}) = (a, b)$.*

Proof. Since $(m_1, w_1) \in S_F$ and $(m_2, w_2) \in S_F$, by Lemma 3.4.4, $(m_1, w_1) \in S_T$ and $(m_2, w_2) \in S_T$ and so there are stable matchings M_1 and M_2 in $\mathcal{M}_S^T$ with $(m_1, w_1) \in M_1$ and $(m_2, w_2) \in M_2$. Then M_1 and M_2 are also stable matchings in I, by Lemma 3.4.2. Let R_1 and R_2 be the rotations associated with M_1 and M_2 respectively in I.

M_{opt} is constructed by eliminating all rotations in $R_{opt} = c(\tau(m_1, w_1)) \cup R'$ where $R' = c(\{\rho \in R : \exists (m_i, w_j) \in \rho$ with rank$(w_j, m_i) > b\})$. We now look at a specific two-step construction order for M_{opt} to show $d'(M_{opt}) = (a, b)$.

- As a first step towards M_{opt}, construct matching $M' = M_0/R'$. It must be that w_2 has a rank $\leq b$ in M'. Since $d_W(M_2) = b$, we know at least all rotations in R' must also be eliminated on M_0 to reach M_2, and so $R' \subseteq R_2$. Then, as $(m_2, w_2) \in M_2$, $\phi(m_2, w_2)$, if it exists, cannot be in R_2 and so is also not in R'. Therefore since w_2 has a rank $\leq b$ in M' and cannot have moved from her rank b partner, it follows that $(m_2, w_2) \in M'$.

- The second step involves eliminating $c(\tau(m_1, w_1))\backslash R'$ on M'. Similar to the above, since $d_W(M_1) \leq b$, we know at least all rotations in R' must also be eliminated on M_0 to reach M_1, and so $R' \subseteq R_1$. As $R' \subseteq R_1$ we also know that $d_U(M') \leq a$. Additionally, it must be possible for us to eliminate $c(\tau(m_1, w_1))\backslash R'$ on M' without increasing any man's rank beyond a, as $M_1 \in \mathcal{M}_S^T$. This elimination may cause some women to improve their rank, however, since $\phi(m_2, w_2)$ (if it is defined) cannot precede $\tau(m_1, w_1)$ (if it is defined), it is not a requirement that w_2 is moved from m_2 in order to create pair (m_1, w_1). Therefore, as $c(\tau(m_1, w_1))\backslash R'$ involves only the minimum rotations necessary to ensure pair (m_1, w_1) exists in M_{opt}, both (m_1, w_1) and (m_2, w_2) must exist in M_{opt} with $d'(M_{opt}) = (a, b)$. Since $d'(M_{opt}) = (a, b)$, $M_{opt} \in$ reduced$(\mathcal{M}_S)$ and it follows that $M_{opt} \in \mathcal{M}_S$.

The above argument assumes a specific construction order of M_{opt}, through eliminating first R' and then $c(\tau(m_1, w_1))$ on M_0. However, since the closed subsets of the rotation poset $R_p(I)$ are in one-to-one correspondence with the set of stable matchings [32], as long as the order of rotation elimination in $R_d(I)$ is respected, the same final stable matching will be produced. Therefore, Line 25 of Algorithm 3.3 will always produce M_{opt} such that $d'(M_{opt}) = (a, b)$ and $M_{opt} \in \mathcal{M}_S$. □

Finally, Proposition 3.4.6 shows that a matching is always returned by Algorithm 3.3 and Theorem 3.4.7 proves that any matching returned must be a regret-equal stable matching.

Proposition 3.4.6. *Algorithm 3.3 always terminates by returning a matching on Line 25.*

Proof. We know $\mathcal{M}_S$ must contain at least one stable matching [19]. Let M be any stable matching of I and let $d_U(M) = a$ and $d_W(M) = b$. Then there exist stable pairs (m_1, w_1) and (m_2, w_2) in M (where (m_1, w_1) and (m_2, w_2) may be the same pair) such that rank$(m_1, w_1) = a$ and rank$(m_2, w_2) = b$.

Assume for contradiction that no matching is returned by Algorithm 3.3. At some point during the algorithm's execution we iterate over the for loop on Line 11 with degree pair (a, b) and create the truncated instance I_T. By Lemma 3.4.2, as $M \in \mathcal{M}_S$, M must also exist in $\mathcal{M}_S^T$. By Lemma 3.4.4, the set of stable pairs S_F we iterate over on Line 17, is equal to the set of all stable pairs S_T in I_T. Since $M \in \mathcal{M}_S^T$ from above, it must be that (m_1, w_1) and (m_2, w_2) exist in S_F. As rank$(m_1, w_1) = a$ and rank$(m_2, w_2) = b$, we must enter the for loop on Line 18 and deduce that $\phi(m_2, w_2)$ does not precede $\tau(m_1, w_1)$ in the rotation poset. This cannot be the case since both (m_1, w_1) and (m_2, w_2) exist in some matching M of I. Hence, we return a matching on Line 25, a contradiction.

Therefore Algorithm 3.3 terminates by returning a matching on Line 25. □

Theorem 3.4.7. *Let I be an instance of* SMI. *Algorithm 3.3 always produces a regret-equal stable matching of I.*

Proof. By Proposition 3.4.6, Algorithm 3.3 must return a matching M_{opt} on Line 25. Let (a, b) be the degree pairs during the final iteration of the main for loop on Line 11, and let (m_1, w_1) and (m_2, w_2) be the (possibly same) stable pairs in M_{opt} such that rank$(m_1, w_1) = a$ and rank$(w_2, m_2) = b$ in the final iteration of the for loop on Line 18. By Lemma 3.4.5, $M_{opt} \in \mathcal{M}_S$ with $d'(M_{opt}) = (a, b)$.

Suppose for contradiction that M_{opt} is not a regret-equal stable matching. Then there exists a stable matching $M' \in \mathcal{M}_S$ such that $d'(M') = (a', b')$ and $|a' - b'| < |a - b|$. During an earlier iteration of the for loop on Line 11, we would have iterated over the degree pairs (a', b'). Using similar reasoning to the proof of Proposition 3.4.6, a matching M'' would have

$r(M)$	Degree pairs in P			
0	$(4,4)$	$(5,5)$		
1	$(3,4)$	$(4,5)$	$(5,4)$	$(5,6)$
2	$(2,4)$	$(3,5)$	$(4,6)$	$(5,7)$
3	$(2,5)$	$(3,6)$	$(4,7)$	
4	$(2,6)$	$(3,7)$		
5	$(2,7)$			

Figure 3.4: Degree pairs in $P = \{2,3,4,5\} \times \{4,5,6,7\}$, sorted according to regret-equality score.

been returned on Line 25 with $d'(M'') = (a', b')$. But then M_{opt} cannot be returned where $d'(M_{opt}) = (a, b)$, a contradiction. Hence, M_{opt} is a regret-equal stable matching. □

3.4.3 Time complexity

Theorem 3.4.8. *Algorithm 3.3 always terminates within $O(n^4)$ time, where n is the number of men or women.*

Proof. The construction of the transitive closure on Line 6 dominates all processes before Line 10, taking $O(m^2)$ time [25, p. 115]. There are $(d_U(M_z) - d_U(M_0) + 1) * (d_W(M_0) - d_W(M_z) + 1)$ pairs in $P = \{d_U(M_0), d_U(M_0) + 1, ..., d_U(M_z)\} \times \{d_W(M_z), d_W(M_z) + 1, ..., d_W(M_0)\}$ on Line 9. With the aid of suitable data structures, the stable pairs of P may be sorted on Line 10 (according to the $\prec$ relation) in a time linear in the size of P. This may be achieved by the following process. Maintain a linked list for each possible regret-equality score. Next, iterate over each man-degree a and each woman-degree b, adding degree pair (a, b) to the end of the appropriate linked list. The sorted order of P may then be found by iterating over each of the linked lists in increasing regret-equality score order. An example of this may be seen in Figure 3.4, where $P = \{2,3,4,5\} \times \{4,5,6,7\}$.

The for loop on Line 11 iterates over degree pairs in P, sorted as described above (bounded by n^2). The truncation of instance I on Line 12 is described in Algorithm 3.4, and comprises two linear-time passes through the preference lists of all men and women, taking $O(m)$ time (in reality this could be implemented with one linear time pass through each preference list). The operation to filter rotations on Line 16, described in Algorithm 3.5, requires us to look at only one pair in each rotation in R to determine whether the man in that pair has rank no better than that of his partner in M_0^T and better than that of his partner in M_z^T. This is because, by Proposition 3.4.3, if a rotation contains one pair within this range, then all pairs are within this range. Since we only check one pair per rotation in R, the time complexity for this operation is linear in $|R|$ and so bounded by $O(m)$. We are then able to find and iterate over all pairs of stable pairs in S_F on Line 18 in $O(n^2)$ time, since there are

a maximum of n pairs in S_F with man's rank a and a maximum of n pairs with woman's rank b. A simple pre-processing step would find these potential candidates in $O(n^2)$ time. For each of these iterations, there is an $O(1)$ time operation to check whether the current pair of stable pairs belong together in a stable matching [25, p. 115]. So far we have the time complexity of $O(n^4)$ within the for loop on Line 11. The elimination of rotations to form M_{opt} would appear to increase the time complexity by a factor of $O(n^2)$, however, since M_{opt} is immediately returned after creation, this only occurs once.

Hence for Algorithm 3.3, we have an overall time complexity of $O(n^4)$. □

3.5 Regret-equal stable matchings with minimum cost

We seek a regret-equal stable matching with minimum cost over all regret-equal stable matchings. This may be achieved in $O(nm^{2.5})$ time using the following process.

For a given SMI instance I, first find the regret-equality score r of the regret-equal stable matching using Algorithm REDI in $O(d_0nm)$ time. Then, iterate over all possible man-woman degree pairs (a, b) such that $|a - b| = r$ (there are $O(n)$ such pairs). For each such degree pair (a, b), truncate men's preference lists at a and women's preference lists at b in $O(m)$ time using Algorithm RESP-Truncate from the previous section, creating instance I'. Then, for each of the $O(m)$ man-woman pairs (m_i, w_j) in I', fix m_i with his ath-choice partner and w_j with her bth-choice partner (where ranks are taken with respect to instance I), if possible. If this is not possible then continue to the next degree pair. We define the *deletion* of pair (m_i, w_j) as the removal of w_j from m_i's preference list and the removal of m_i from w_j's preference list. Assume that w'_j is m_i's ath-choice partner, and m'_i is w_j's bth-choice partner. In I', we now delete pairs (m''_i, w''_j) for any w''_j such that m_i prefers w''_j to w'_j and w''_j prefers m_i to m''_i. Also delete the pair (m''_i, w''_j) for any m''_i such that w_j prefers m''_i to m'_i and m''_i prefers w_j to w''_j. Next we delete all remaining preference list elements of m_i except w'_j and all remaining preference list elements of w_j except m'_i. The Gale-Shapley Algorithm is run to check that a stable matching of size n exists in I'. If no such stable matching exists then we move on to the next degree pair. Feder's Algorithm may then be used to find an egalitarian stable matching in the reduced SMI instance I' in $O(m^{1.5})$ time (using the original ranks in I as costs). This makes a total of $O(nm^{2.5})$ time to find a regret-equal stable matching with minimum cost.

A theoretical bound may exist for the cost of a regret-equal stable matching with minimum cost in relation to the cost of an egalitarian stable matching. However, as we show in the following example, it is possible for a regret-equal stable matching with minimum cost to have a cost greater than twice that of an egalitarian stable matching. Let I_0 be the example SM instance shown in Table 3.2. Instance I_0 has four stable matchings, also shown in this

Agent	Preference lists												M_1	M_2	M_3	M_4
m_1:	w_5	w_3	w_4	w_1	w_6	w_2	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_1	w_2	w_2	w_1
m_2:	w_2	w_1	w_4	w_5	w_6	w_3	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_2	w_3	w_3	w_2
m_3:	w_3	w_1	w_2	w_5	w_6	w_4	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_3	w_4	w_4	w_3
m_4:	w_4	w_1	w_2	w_3	w_6	w_5	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_4	w_5	w_5	w_4
m_5:	w_5	w_1	w_2	w_3	w_4	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_5	w_6	w_6	w_5
m_6:	w_6	w_2	w_3	w_4	w_5	w_1	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_6	w_1	w_1	w_6
m_7:	w_{12}	w_7	w_8	w_9	w_{10}	w_{11}	w_1	w_2	w_3	w_4	w_5	w_6	w_{12}	w_{12}	w_7	w_7
m_8:	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_7	w_8	w_8
m_9:	w_8	w_9	w_{10}	w_{11}	w_{12}	w_7	w_1	w_2	w_3	w_4	w_5	w_6	w_8	w_8	w_9	w_9
m_{10}:	w_9	w_{10}	w_7	w_8	w_{11}	w_{12}	w_1	w_2	w_3	w_4	w_5	w_6	w_9	w_9	w_{10}	w_{10}
m_{11}:	w_{10}	w_{11}	w_7	w_8	w_9	w_{12}	w_1	w_2	w_3	w_4	w_5	w_6	w_{10}	w_{10}	w_{11}	w_{11}
m_{12}:	w_{11}	w_{12}	w_7	w_8	w_9	w_{10}	w_1	w_2	w_3	w_4	w_5	w_6	w_{11}	w_{11}	w_{12}	w_{12}
w_1:	m_6	m_1	m_2	m_3	m_4	m_5	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_1	m_6	m_6	m_1
w_2:	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_2	m_1	m_1	m_2
w_3:	m_2	m_3	m_4	m_5	m_6	m_1	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_3	m_2	m_2	m_3
w_4:	m_3	m_4	m_1	m_2	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_4	m_3	m_3	m_4
w_5:	m_4	m_5	m_1	m_2	m_3	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_5	m_4	m_4	m_5
w_6:	m_5	m_6	m_1	m_2	m_3	m_4	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_6	m_5	m_5	m_6
w_7:	m_7	m_9	m_{10}	m_{11}	m_{12}	m_8	m_1	m_2	m_3	m_4	m_5	m_6	m_8	m_8	m_7	m_7
w_8:	m_8	m_7	m_{10}	m_{11}	m_{12}	m_9	m_1	m_2	m_3	m_4	m_5	m_6	m_9	m_9	m_8	m_8
w_9:	m_9	m_7	m_8	m_{11}	m_{12}	m_{10}	m_1	m_2	m_3	m_4	m_5	m_6	m_{10}	m_{10}	m_9	m_9
w_{10}:	m_{10}	m_7	m_8	m_9	m_{12}	m_{11}	m_1	m_2	m_3	m_4	m_5	m_6	m_{11}	m_{11}	m_{10}	m_{10}
w_{11}:	m_{11}	m_7	m_8	m_9	m_{10}	m_{12}	m_1	m_2	m_3	m_4	m_5	m_6	m_{12}	m_{12}	m_{11}	m_{11}
w_{12}:	m_{12}	m_8	m_9	m_{10}	m_{11}	m_7	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_7	m_{12}	m_{12}
												$r(M)$	2	0	5	2
												$c(M)$	63	84	60	39

Table 3.2: SM instance I_0 with stable matchings M_1, M_2, M_3 and M_4.

table. Matching M_2 is the only regret-equal stable matching in I_0, and is therefore also the unique regret-equal stable matching with minimum cost. Matching M_4 is the unique egalitarian stable matching, and since $84 = c(M_2) > 2 * c(M_4) = 78$, any theoretical bound for the cost of a regret-equal stable matching with minimum cost must be more than twice that of an egalitarian stable matching.

3.6 Algorithm to find a min-regret sum stable matching

Algorithm MRS, which finds a min-regret sum stable matching, given an instance of SMI, is presented as Algorithm 3.6. First, the man-optimal and woman-optimal stable matchings, M_0 and M_z, are found using the Man-oriented and Women-oriented Gale-Shapley Algo-

rithms [19]. The best matching found so far, denoted M_{opt} is initialised to M_0. We then iterate over each possible man-degree a between $d_U(M_0)$ and $d_U(M_z)$ inclusive, where an improvement of M_{opt}, according to the regret sum, is still possible. As an example, suppose M_{opt} has a regret sum of 5 with $d_U(M_{opt}) = 2$ and $d_W(M_{opt}) = 3$. Then, it is not worth iterating over any man-degree greater than 3 since it will not be possible to improve on the regret sum of 5 by doing so. For each iteration of the while loop, we truncate the men's preference lists at a and find the woman-optimal stable matching M_z^T for this truncated instance. If the regret sum of M_z^T is smaller than that of M_{opt}, we update M_{opt} to M_z^T. After all iterations over possible men's degrees are completed, M_{opt} is returned.

Algorithm 3.6 MRS(I), returns a min-regret sum stable matching for an instance I of SMI.

Require: An instance I of SMI.
Ensure: Return a min-regret sum stable matching M_{opt}.

1: $M_0 \leftarrow$ MGS(I) ▷ M_0 is the man-optimal stable matching found using the Man-oriented Gale-Shapley Algorithm (MGS) [19].
2: $M_z \leftarrow$ WGS(I) ▷ M_z is the woman-optimal stable matching found using the Woman-oriented Gale-Shapley Algorithm (WGS) [19].
3: $M_{opt} \leftarrow M_0$
4: $a \leftarrow d_U(M_0)$
5: **while** $a \leq d_U(M_z)$ **and** $a + 1 < d_U(M_{opt}) + d_W(M_{opt})$ **do**
6: $\quad I_T \leftarrow$RESP-Truncate$(I, (a, n))$
7: $\quad M_z^T \leftarrow$ WGS(I_T)
8: $\quad$ **if** $d_U(M_z^T) + d_W(M_z^T) < d_U(M_{opt}) + d_W(M_{opt})$ **then**
9: $\quad\quad M_{opt} \leftarrow M_z^T$
10: $\quad$ **end if**
11: $\quad a \leftarrow a + 1$
12: **end while**
13: **return** M_{opt}

Let d_s denote the difference between the degree of men in the woman-optimal stable matching M_z, and in the man-optimal stable matching M_0, that is $d_s = d_U(M_z) - d_U(M_0)$. In Theorem 3.6.1, we show that Algorithm MRS produces a min-regret sum stable matching in $O(d_s m)$ time.

Theorem 3.6.1. *Let I be an instance of* SMI. *Algorithm* MRS *produces a min-regret sum stable matching in $O(d_s m)$ time, where $d_s = d_U(M_z) - d_U(M_0)$, m is the total length of all preference lists, and M_0 and M_z are the man-optimal and woman-optimal stable matchings respectively.*

Proof. We first redefine several definitions introduced at the beginning of Section 3.4.2. Let I_T be the truncated instance of I created on Line 6 where men are truncated at a and women are truncated at n (effectively not truncated), and let $\mathcal{M}_S^T$ be the set of stable matchings in

I_T. Finally, let

$$\text{reduced}(\mathcal{M}_S) = \{M \in \mathcal{M}_S : \forall (m_i, w_j) \in M, \text{rank}(m_i, w_j) \leq a \ \wedge \text{rank}(m_i, w_j) \leq n\}.$$

Then by Lemma 3.4.2, with $b = n$, we have $\mathcal{M}_S^T = \text{reduced}(\mathcal{M}_S)$.

Let M be a min-regret sum stable matching in I with $d_U(M)$ minimum among all min-regret sum stable matchings. We show that it is not possible to miss a stable matching M' with $d'(M') = d'(M)$, during the execution of Algorithm MRS. On Line 5 of Algorithm 3.6, we iterate over all possible men's degrees that may correspond to a min-regret sum stable matching, and will enter the while loop for degree value $a = d_U(M)$. Since $a = d_U(M)$, we know that $M \in \text{reduced}(\mathcal{M}_S)$ and so from above, $M \in \mathcal{M}_S^T$. We also know by Lemma 3.4.2, that $M_z^T \in \mathcal{M}_S$. Let R' and R_z^T be the set of rotations associated with M and M_z^T in I. On Line 7 of the algorithm we find the woman-optimal stable matching M_z^T in $\mathcal{M}_S^T$ and so we must have $\text{rank}(w_j, M_z^T(w_j)) \leq \text{rank}(w_j, M(w_j))$, for all women w_j, since $M \in \mathcal{M}_S^T$. From this inequality, we know $R' \subseteq R_z^T$, and since $d_U(M) = a$ it must be that $d_U(M_z^T) = a$. Additionally, this inequality implies $d_W(M_z^T) \leq d_W(M)$. As M is a min-regret sum stable matching, there cannot be a stable matching with man-degree a and woman-degree $< d_W(M)$, but as $d_W(M_z^T) \leq d_W(M)$, we must have $d_W(M_z^T) = d_W(M)$. Hence, $d'(M_z^T) = d'(M)$. Note that due to the choice of M, with $d_U(M)$ minimum among all min-regret sum stable matchings, it is not possible for M_{opt} to be updated to a min-regret stable matching prior to this point in the algorithm. Therefore, M_{opt} is updated to M_z^T on Line 9. Additionally, as M_{opt} is now a min-regret stable matching it will not be possible for it to be updated to another stable matching after this point, and so M_{opt} is returned on Line 13, as required.

Truncation of our instance requires two linear passes through preference lists of men and women and is therefore an $O(m)$ operation. As stated in a previous proof, in reality this could be implemented with one linear time pass through each preference list. The man-optimal and woman-optimal stable matchings can found in $O(m)$ time both within and outwith the while loop. Since the number of potential men's degrees that we iterate over is bounded by $d_s = d_U(M_z) - d_U(M_0)$, we have an overall time complexity of $O(d_s m)$ for Algorithm MRS. □

3.7 Experiments

3.7.1 Methodology

Recall that an Enumeration Algorithm exists to find the set of all stable matchings of an instance I of SMI in $O(m + n|\mathcal{M}_S|)$ time [23]. For brevity, we denote this enumera-

tion algorithm as Algorithm ENUM for the remainder of this chapter. Within this time complexity, it is possible to output a regret-equal stable matching from this set of stable matchings, by keeping track of the best stable matching found so far (according to the regret-equality score) as they are created. We randomly generated instances of SM, in order to compare the performance of Algorithms REDI, RESP and ENUM. Using output from Algorithm ENUM, we also investigated the effect of varying instance sizes, for six different types of optimal stable matchings (balanced, sex-equal, egalitarian, min-regret, regret-equal, min-regret sum), and also outputs from Algorithms REDI and RESP, over a range of measures (including balanced score, sex-equal score, cost, degree, regret-equality score, regret sum). Tests were run over 19 different instance types with varying instance size ($n \in \{10, 20, ..., 100, 200, ..., 1000\}$). Preliminary experimentation showed that, in general, complete preference lists generated according to a uniform distribution produced a larger number of stable matchings than using incomplete lists or linear distributions in which the most popular man was p times more popular than the least popular man (similar for women). Therefore since we wished to compare properties of different stable matchings, all instances have complete, uniformly distributed preference lists. Experiments were run over 500 instances of each instance type.

Each instance was run over the three algorithms described above with a timeout time of 1 hour for each algorithm. Experiments were conducted on the machine described in Chapter 1, running Ubuntu version 18.04. Instance generation, correctness and statistics summarisation programs, and plot and LaTeX table generation were all written in Python and run on Python version 3.6.1. All other code was written in Java and compiled using Java version 1.8.0. Each instance was run on a single thread with 16 instances run in parallel using GNU Parallel [72]. Serial Java garbage collection was used with a maximum heap size of 2GB distributed to each thread. Code and data repositories for these experiments can be found at `https://zenodo.org/record/3630383` and `https://zenodo.org/record/3630349` respectively.

Correctness tests were run in the following way. In addition to the above, a further two instance types were generated where $n \in \{6, 8\}$, with 5000 instances for each type. Over all instances of the 21 instance types, each matching output by an algorithm (one each for Algorithms REDI and RESP, and multiple for Algorithm ENUM), was tested for *(1) capacity:* each man (woman) may only be assigned to one woman (man) respectively; and *(2) stability:* no blocking pair exists. Additionally, the regret-equal score of the stable matchings output by each of the three algorithms were compared against each other (if they had not timed out) to ensure they were identical values. These tests were were written in Java and compiled using Java version 1.8.0. Finally, for all instances types where $n \leq 50$, further correctness testing was conducted on Algorithm ENUM to ensure that the correct number of stable matchings was produced. This was done using an IP model built using the IP modelling framework

PuLP (version 1.6.9) [59] running CPLEX (version 12.8.0) [28] with Python version 2.7.15. Similar to above, all instances were run on a single thread with 16 instances run in parallel using GNU Parallel [72]. A timeout time of 30 minutes was applied to each instance for the PuLP program, and all instances completed within the time limit. All correctness tests passed successfully.

3.7.2 Experimental results summary

A summary of generated instance information and algorithm timeouts may be seen in Table 3.3. Instance types are labelled according to n, e.g., S100 is the instance type containing instances where $n = 100$. Columns 3 and 4 show the mean number of stable matchings $|\mathcal{M}_S|_{av}$ and mean number of rotations $|R|_{av}$, respectively. Then, in columns 5, 6 and 7 we give the number of instances that timed out (after 1 hour) for Algorithms REDI, RESP and ENUM.

Figure 3.5 shows a comparison of the time taken to execute the three algorithms over increasing values of n. Precise data for this plot can be seen in Table A.1 of Appendix A which gives the mean, median, 5th percentile and 95th percentile durations for Algorithms REDI, RESP and ENUM. In Figure 3.5, the median values of time taken for each algorithm are plotted and a 90% confidence interval is displayed using the 5th and 95th percentile measurements.

Figure 3.6 shows comparisons of six different types of optimal stable matchings (balanced, sex-equal, egalitarian, min-regret, regret-equal, min-regret sum), and outputs from Algorithms REDI and RESP, over a range of measures (including balanced score, sex-equal score, cost, degree, regret-equality score, regret sum), as n increases. Optimal stable matching statistics involving a measure determined by cost (respectively degree) are given a green (respectively blue) colour. For a particular fairness objective A and a particular fairness measure B, there may be a set of several stable matchings that are optimal with respect to A. In this case we choose a matching from this set that has best possible measure with respect to B. For example, if we are looking at the regret-equality score, for a particular instance, we find a sex-equal stable matching that has smallest regret-equality score (over the set of all sex-equal stable matchings) and use this value to plot the regret-equal score for this type of optimal stable matching. This process is replicated for the other types of optimal stable matching. In each case the mean measure value is plotted for the given type of optimal stable matching. Data for these plots may be found in Tables A.2, A.3, A.4, A.5, A.6 and A.7 of Appendix A.

Figure 3.7 (associated with Table A.8 in Appendix A) shows a bar chart of the mean number of stable matchings occurring for the six different types of optimal stable matching described above, with increasing n. Finally, Figure 3.8 (associated with Table A.9 in Appendix A)

shows a bar chart of the mean number of stable matchings that satisfy different numbers of optimal stable matching criteria, with increasing n. Both of these bar charts show a reduced selection of n with $n \in \{100, 400, 700, 1000\}$ (the tables show the full data).

The main results of these experiments are:

- *Time taken:* It is clear from Figure 3.5 that Algorithm REDI is the fastest algorithm in practice among the three, taking approximately 2s to solve an instance of size $n = 1000$ with very little variation. The Algorithm ENUM takes around 8s for an instance of size $n = 1000$ with a far larger variation. Finally Algorithm RESP is the slowest taking around 2000s to solve an instance of the same size. It may seem surprising that the $O(n^4)$ Algorithm RESP appears to perform worse than the $O(m + n|\mathcal{M}_S|)$ Algorithm ENUM, which is exponential in the worst case. However, Algorithm ENUM is polynomial in the number of stable matchings $|\mathcal{M}_S|$ and since $|\mathcal{M}_S|$ tends to the order of $n \log n$ with larger n [44], this algorithm may behave more like a lower order polynomial in the general case. We can also see from Table 3.3 that the poorer performance of Algorithm RESP results in some timeouts for instance types with higher values of n. For all time-based calculations, instances that timed out were deemed to have taken the 1 hour timeout time. This means that the 95th percentile values reported for Algorithm RESP are slightly underestimated for instance types with $n \geq 600$, however the 5th percentile and median values remain accurate.

- *Balanced and sex-equal stable matchings:* In all plots of Figure 3.6, balanced and sex-equal stable matchings have remarkably similar mean scores over all instance sizes and all measures. This may be due to the similar nature of these optimality measures, where both measures involve a calculation over the cost of matchings (recall that the balanced objective involves minimising the maximum of the total cost for the men and the total cost for the women, whilst the sex-equal objective involves minimising the absolute value of the difference between the total cost for the men and the total cost for the women). This similarity was previously noted by Manlove [47, p. 110], who references work undertaken by Eric McDermid to find an instance of SMI in which no balanced stable matching is also a sex-equal stable matching.

- *Sex-equal score:* A wide variation in sex-equal score over the six optimal matchings can be seen in Figure 3.6b (and Table A.3). Sex-equal and balanced stable matchings are extremely closely aligned giving a mean sex-equal score of 265.0 and 284.0 respectively for the instance type with $n = 1000$. Min-sum regret stable matchings, on the other hand, performed the least well with a mean sex-equal score of 12400.0 for the same instance type.

- *Regret-equality score:* Similar to the previous point we see a wide variation in regret-equality score over the six optimal stable matchings in Figure 3.6e (and Table A.6). For the instance type with $n = 1000$, this ranges from a mean regret-equality score of 14.2 for the regret-equal stable matching to 84.6 for the minimum regret stable matching. It is interesting to note that the type of optimal stable matching (out of the six optimal stable matchings tested) whose regret-equality score tends to be furthest away from that of a regret-equal stable matching is the min-regret sum stable matching. This may be due to the fact that minimising the sum of two measures does not necessarily force the two measures to be close together.

- *Outputs from Algorithms* REDI *and* RESP*:* Over all measures, outputs from Algorithms REDI and RESP were remarkably similar and so in this regard, neither algorithm is preferred over the other. Note that timeouts occurred for Algorithm RESP for the five largest instance types, meaning that mean results are somewhat skewed for this algorithm beyond $n = 600$ since the means are only computed over instances that terminated before timeout. Figure 3.6e and Table A.6 even appear to show that output from Algorithm RESP has a smaller regret-equality score than that of a regret-equal stable matching, which of course is not the case. Clearly, for the instances that timed out, the regret-equality score of a regret-equal stable matching must have been larger in general than for instances that completed before the timeout. This is explained by the fact that Algorithm RESP iterates through degree pairs in non-decreasing regret-equality score order. Due to the wide variation of regret-equality scores among different types of optimal stable matchings (as described above) it is clear that no other optimal stable matching is able to closely approximate a regret-equal stable matching, which highlights the importance of Algorithms REDI and RESP that are designed specifically for optimising this measure. Interestingly, Algorithms REDI and RESP are also competitive in terms of balanced score, cost and degree. Indeed, we can see from Tables A.2, A.4 and A.5, that Algorithm REDI approximates these types of optimal stable matchings at an average of 9.0%, 1.1% and 3.0% over their respective optimal values, for instances with $n = 1000$. Over all instance sizes, these values are within ranges $[4.0\%, 10.9\%]$, $[1.1\%, 3.4\%]$ and $[1.3\%, 3.7\%]$, respectively. This gives a good indication of the high-quality of output from this algorithm even on seemingly unrelated measures. Algorithm RESP performed slightly better than Algorithm REDI for instances with $n = 1000$ over the above measurements, however this is likely to be due to the more complex cases (with larger measure values) timing out.

- *Frequency of different types of optimal stable matchings:* From Figure 3.7, we can see a clear ordering for the three most frequent types of degree-based optimal stable matching. From most frequent to least frequent they are minimum regret, regret-equal

and min-regret sum stable matchings. The minimum-regret stable matching may be most frequent because this optimality criterion is somewhat less constrained than the other two, as it is based only on the worst performing agent. In contrast, the regret-equal and min-regret sum stable matchings are based on the worst performing man and worst performing woman. Additionally, cost-based optimal stable matchings are likely to be more constrained than degree-based ones (due to the number of different costs and degrees possible for stable matchings of any n), which may account for the very low average number of stable matchings for these types.

- *The number of optimality criteria that stable matchings satisfy:* The bar chart in Figure 3.8 shows a clear pattern of fewer stable matchings satisfying higher numbers of optimality criteria. For smaller instances there is a trend for this to level out somewhat, which can be seen more clearly for the instance types with lowest n in Table A.9. Taken to extreme, if there is only one stable matching in an instance, it will necessarily satisfy all optimality criteria. As n increases, so too does the number of stable matchings. This increase in number of stable matchings has the effect of increasing the number of possible values of any particular measure, such as cost, over the set of all stable matchings. It is therefore less likely that any particular one stable matching will obtain the optimal value associated with this measure and so, in turn, a particular stable matching is likely to be optimal for fewer optimality criteria.

Case	n	$\lvert\mathcal{M}_S\rvert_{av}$	$\lvert R\rvert_{av}$	Timeout REDI	Timeout RESP	Timeout ENUM
S10	10	3.0	1.8	0	0	0
S20	20	6.7	4.2	0	0	0
S30	30	10.3	6.3	0	0	0
S40	40	16.1	8.9	0	0	0
S50	50	21.0	11.2	0	0	0
S60	60	27.7	13.7	0	0	0
S70	70	33.1	15.8	0	0	0
S80	80	40.8	18.1	0	0	0
S90	90	48.0	20.4	0	0	0
S100	100	54.8	22.7	0	0	0
S200	200	139.2	42.4	0	0	0
S300	300	219.1	58.9	0	0	0
S400	400	348.4	76.2	0	0	0
S500	500	442.5	90.3	0	0	0
S600	600	546.2	105.7	0	1	0
S700	700	670.5	118.8	0	2	0
S800	800	815.2	132.5	0	41	0
S900	900	977.0	144.0	0	118	0
S1000	1000	1077.5	156.7	0	167	0

Table 3.3: General instance and algorithm timeout results.

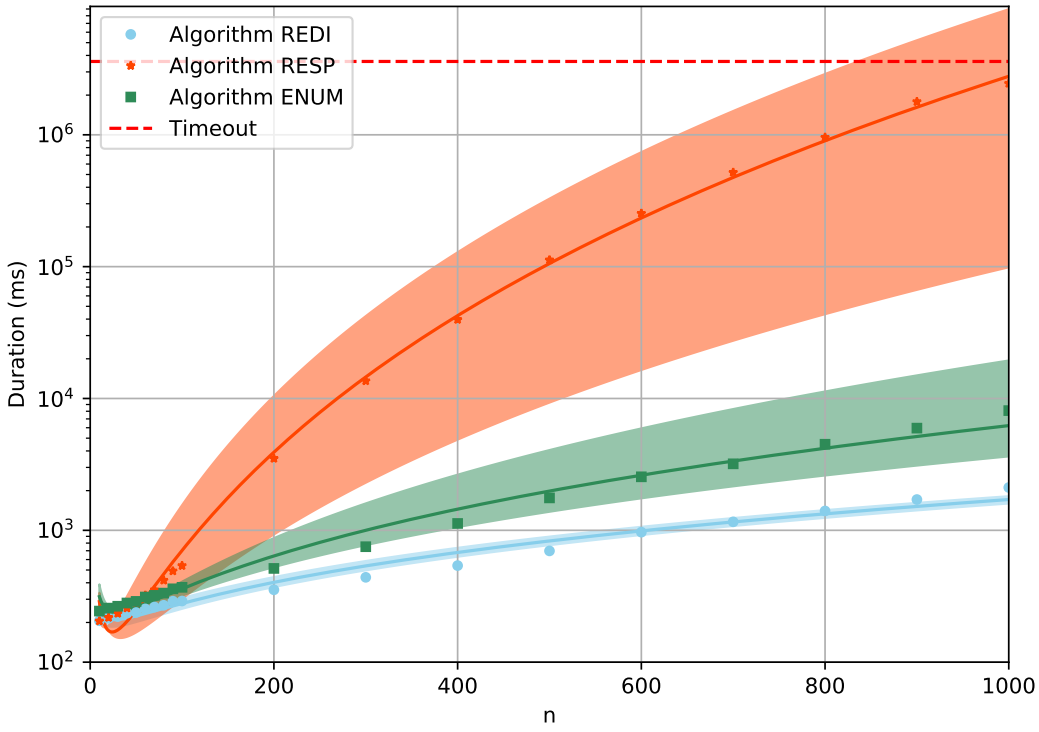
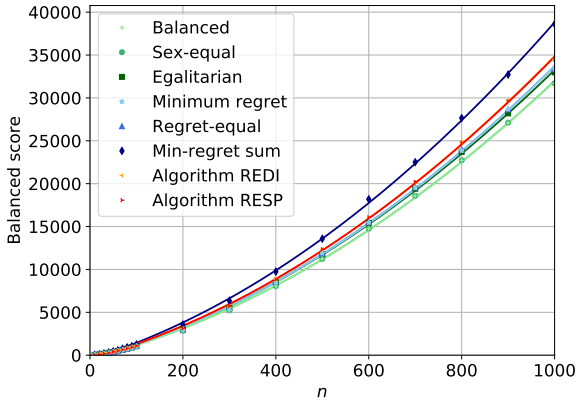
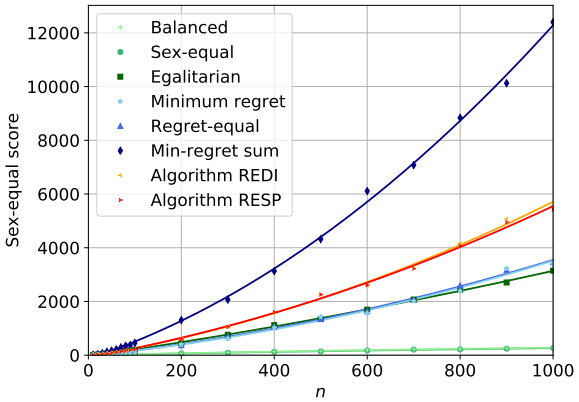


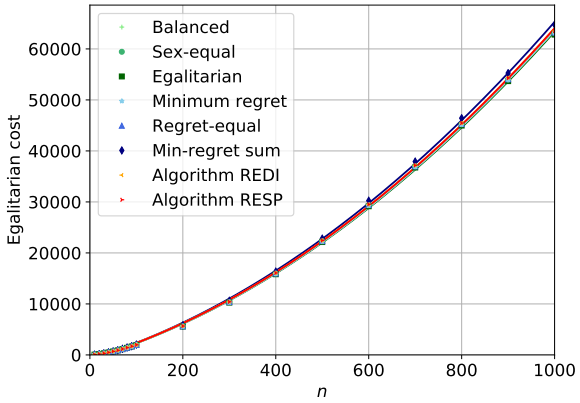
Figure 3.5: A log plot of the time taken to execute Algorithms REDI, RESP and ENUM, where n is the number of men or women. A second order polynomial model has been assumed for all best-fit lines.



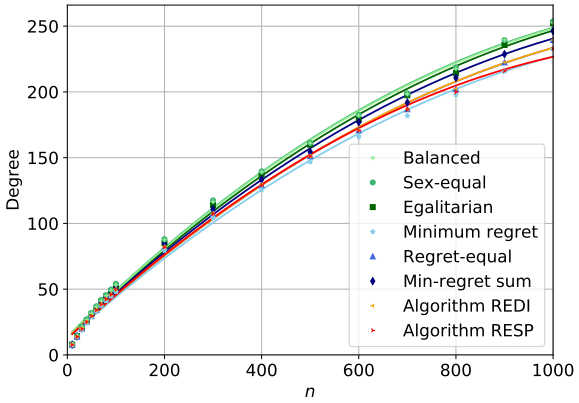
(a) Plot of balanced score.



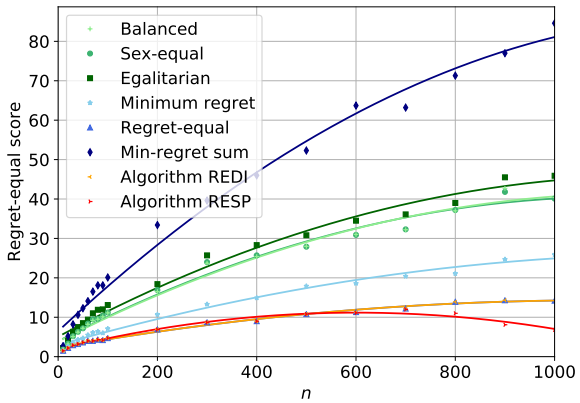
(b) Plot of sex-equal score.



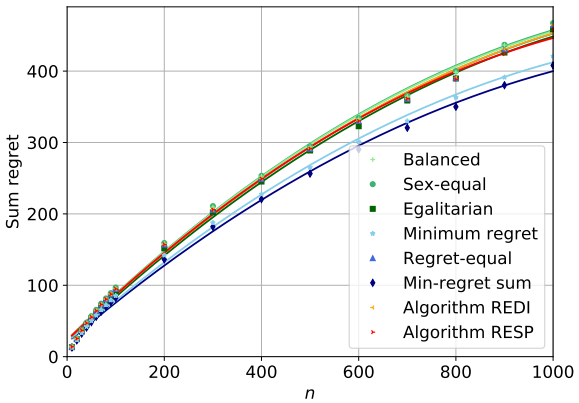
(c) Plot of cost.



(d) Plot of degree.



(e) Plot of regret-equality score.



(f) Plot of regret sum.

Figure 3.6: Plots of experiments to compare six different optimal stable matchings (balanced, sex-equal, egalitarian, min-regret, regret-equal, min-regret sum), and outputs from Algorithms REDI and RESP, over a range of measures (including balanced score, sex-equal score, cost, degree, regret-equality score, regret sum), where n is the number of men or women. A second order polynomial model has been assumed for all best-fit lines.

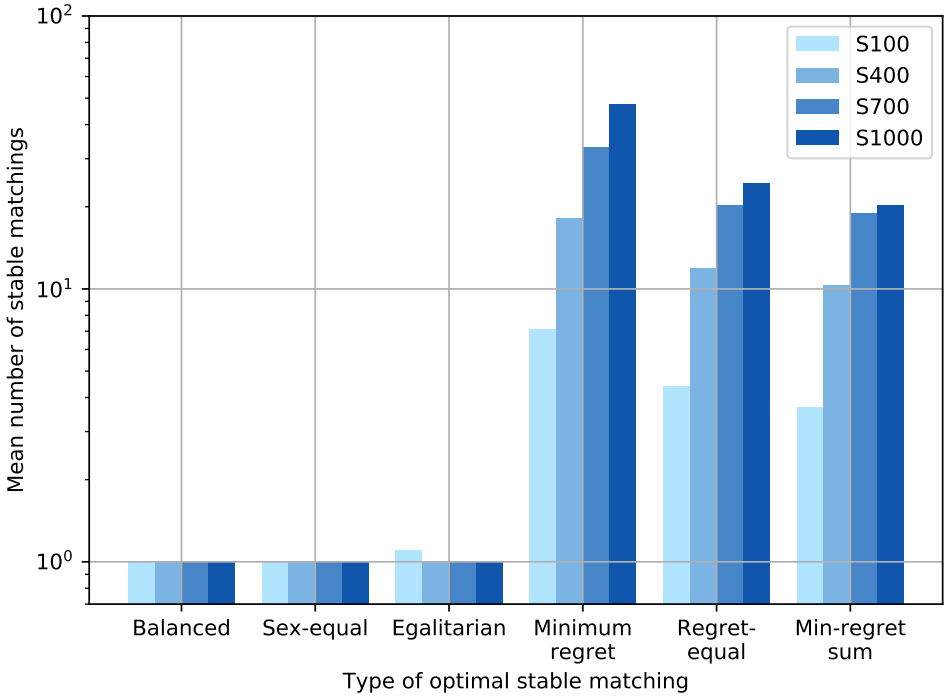


Figure 3.7: Bar chart of the mean number of stable matchings for different types of optimal matchings, for $n \in \{100, 400, 700, 1000\}$.

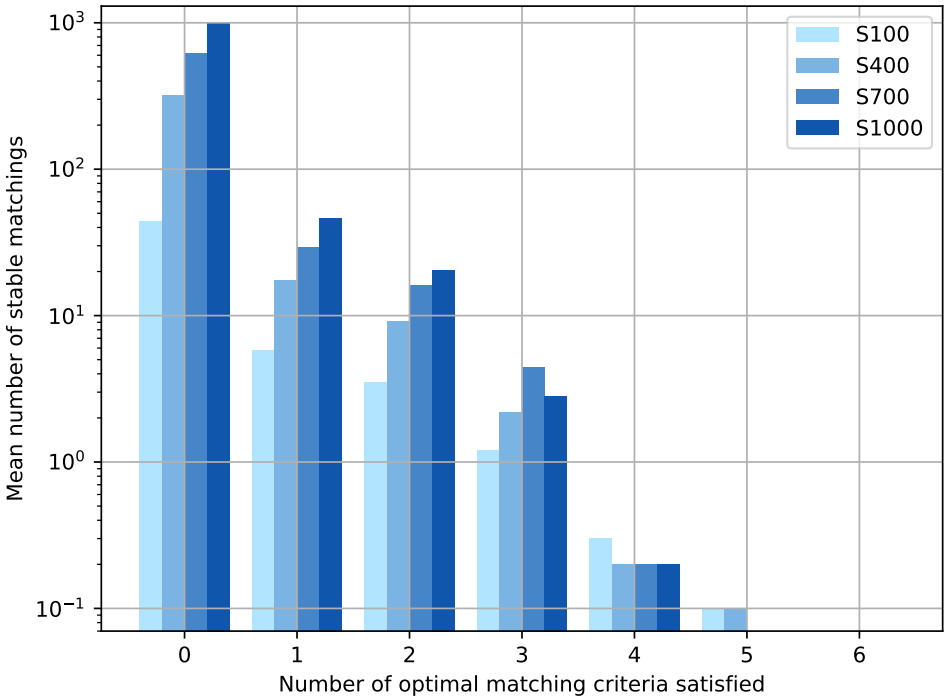


Figure 3.8: Bar chart of the mean number of stable matchings which satisfy different numbers of optimal stable matching criteria, for $n \in \{100, 400, 700, 1000\}$.

3.8 Conclusions and future work

We introduced two new notions of fair stable matchings for SMI, namely, the regret-equal stable matching and the min-regret sum stable matching. We presented algorithms that are able to compute matchings of these types in polynomial time: $O(d_0nm)$ time for the regret-equal stable matching, where $d_0 = |d_U(M_0) - d_W(M_0)|$; and $O(d_sm)$ time for the min-regret sum stable matching, where $d_s = d_U(M_z) - d_U(M_0)$. It remains open as to whether these time complexities can be improved.

We now define new variants of SMI and HR in which groups are formed on one side of the market, and look at open problems relating to stable matchings that are in some sense fair among these groups.

Let the *Stable Marriage problem with Incomplete lists and Grouped Women* (SMI-GW) be the extension of SMI in which the set of women is partitioned into two disjoint sets W_1 and W_2. This scenario may arise, for example, in a practical situation where each group of women is a group of workers with particular characteristics and each man is a job within a company. The aim then is to distribute jobs fairly among different groups of workers. With respect to a stable matching M, let $d_{W_1}(M)$ and $d_{W_2}(M)$ denote the rank of the worst-ranked woman in W_1 and W_2 respectively, and let the *regret-equality score* be given by $r'(M) = |d_{W_1}(M) - d_{W_2}(M)|$. Finally, let a stable matching M in SMI-GW be *regret-equal* if $r'(M)$ is minimum among all stable matchings in $\mathcal{M}_S$. In the worker-job scenario, a regret-equal stable matching would aim to give the worst-off worker from one group a job that they rank as close as possible to the worst-off worker from the other group. Let $d_{t_i} = d_{W_i}(M_0) - d_{W_i}(M_z)$ for $i \in \{1, 2\}$. A simple $O(d_{t_1}d_{t_2}m)$ algorithm exists to find the regret-equal stable matching in SMI-GW and executes as follows. Calculate the man-optimal stable matching M_0 and the woman-optimal stable matching M_z using the Man-oriented and Woman-oriented Gale-Shapley Algorithms [19] respectively. Iterate over all possible degree pairs $(\{d_{W_1}(M_z), ..., d_{W_1}(M_0)\} \times \{d_{W_2}(M_z), ..., d_{W_2}(M_0)\})$ of groups W_1 and W_2 in order of non-decreasing regret-equality score. For degree pair (a, b), eliminate all rotations from M_0 containing a woman from group W_1 with rank greater than a (if any), and all rotations containing a woman from group W_2 with rank greater than b (if any). If the resultant matching M has $d_{W_1}(M) = a$ and $d_{W_2}(M) = b$ then return M as optimal.

Let the *Hospitals/Residents problem with Grouped Residents* (HR-GR) be the extension of HR in which each resident belongs to one of two groups R_1 and R_2. An extended concept of a rotation (defined in SMI), known as a *meta-rotation* [7], exists in instances of HR. As with a rotation in the SMI case, informally, a *meta-rotation* is a list of resident-hospital pairs, such that when permuted on a stable matching (on which they are exposed), give rise to another stable matching. As in the SMI case, there is a one-to-one correspondence between the set of stable matchings and the closed subsets of the *meta-rotation poset* [7]. We may then use a

similar algorithm to that described above, to find a regret-equal stable matching M such that $r''(M) = |d_{R_1}(M) - d_{R_2}(M)|$ is minimum over all stable matchings. Within the algorithm above, residents take the place of women and hospitals take the place of men. It is easy to see that this algorithm may be extended to instances with any constant number of groups and still be run in polynomial time.

It is an open problem as to whether a stable matching, analogous to a sex-equal stable matching in SMI, may be found in polynomial time for either SMI-GW or HR-GR. Other fairness criteria in Table 3.1 are trivial for the HR-GR case since a resident-optimal stable matching would naturally satisfy analogous balanced, min-regret, egalitarian and min-regret sum criteria.

In Section 2.2.2.2 we saw that a balanced stable matching may be approximated within a factor of $2 - \frac{1}{l}$, where l is the length of the longest preference list [47, p. 110], and that it is possible to find a near sex-equal stable matching in polynomial time [35]. A further consideration is to look at whether one type of optimal stable matching is able to closely approximate another. In Section 3.5 we showed that a minimum cost regret-equal stable matching does not in general approximate an egalitarian stable matching within a factor of 2. Of the six types of optimal stable matching studied, two are NP-hard to find (balanced and sex-equal stable matchings) and so it is reasonable to ask whether it is possible to approximate either of these stable matchings using the other types of optimal stable matchings that can be found in polynomial time. It is clear from Figure 3.6b that no type of optimal stable matching (apart from the balanced stable matching) is in general able to closely approximate a sex-equal stable matching. Our empirical work showed that, in practice, the types of optimal stable matching that can be found in polynomial time were able to reasonably closely approximate a balanced stable matching. The best current bound on the performance guarantee of any approximation algorithm for the problem of finding a balanced stable matching is 2. It remains open as to whether any of the types of polynomial-time computable optimal stable matchings considered in this chapter can approximate a balanced stable matching to within a factor less than 2 in the worst case.

Chapter 4

Profile-based stable matchings in SMI

4.1 Introduction

4.1.1 Background

In this chapter we continue our study of fairness in SMI by examining the problem of finding profile-based optimal stable matchings. Both SMI and profile-based optimality were introduced in Section 2.2.2. In particular we study the problems of finding a rank-maximal stable matching and a generous stable matching in SMI. Profile-based optimality such as rank-maximality or the generous criteria provide guarantees that do not exist with other optimality criteria giving a distinct advantage to these approaches in certain scenarios.

In addition to investigating profile-based optimality in SMI, we also examine the complexity of the problem of finding profile-based optimal stable matchings in SR (the generalisation of SMI introduced in Section 2.3).

Recall that in an instance of SMI, n is the number of men or women, and m is the total length of all preference lists. Irving et al. [32] describe the use of weights that are exponential in n (henceforth *exponential weights*) in order to find a rank-maximal stable matching using a maximum weight approach. This requires an additional factor of $O(n)$ time complexity to take into account calculations over exponential weights, giving an overall time complexity of $O(nm^2 \log n)$ [1]. Irving et al.'s approach (described in more detail in Section 4.3) requires a Max Flow algorithm to be used. Irving et al. [32] stated that the strongly polynomial $O(m^2 \log n)$ Sleator-Tarjan algorithm [71] was the best option (at the time of writing). The Sleator-Tarjan algorithm [71] is an adapted version of Dinic's algorithm [11] and finds a maximum flow in a network in $O(|V||E| \log |V|)$ time. Since $|V| \leq m$, $|E| \leq m$

[1] Irving et al. [32] actually state a time complexity of $O(nm^2 \log n \log n)$, however, we believe that this time complexity bound is somewhat pessimistic and that a bound of $O(nm^2 \log n)$ applies to this approach.

and $O(\log m)$ is equivalent to $O(\log n)$ [25], this translates to $O(m^2 \log n)$ for the maximum weight stable matching problem and an overall time complexity of $O(nm^2 \log n)$ for the rank-maximal stable matching problem. However in 2013 Orlin [64] described an improved strongly polynomial Max Flow algorithm with an $O(|V||E|)$ (translating to $O(m^2)$) time complexity, giving a total overall time complexity for finding a rank-maximal stable matching of $O(nm^2)$. Feder's weighted SAT approach [15] has an overall $O(n^{0.5}m^{1.5})$ time complexity for finding a rank-maximal stable matching. Neither Irving et al. [32] nor Feder [15] considered generous stable matchings, however, a generous stable matching may be found in a similar way to a rank-maximal stable matching with the use of exponential weights.

4.1.2 Motivation

For the rank-maximal stable matching problem, Irving et al. [32] suggest a weight of n^{n-i} for each agent assigned to their ith choice and a similar approach can be taken to find a generous stable matching as we demonstrate later in this chapter. In both the rank-maximal and generous cases, the use of exponential weights introduces the possibility of overflow and accuracy errors upon implementation. This may occur as a consequence of limitations of data types: in Java for example, the *int* and *long* primitive types restrict the number of integers that can be represented, and the *double* primitive type may introduce inaccuracies when the number of significant figures is greater than 15. Using a weight of n^{n-i} for each agent assigned to their ith choice as above, it may be that we need to distribute n capacities of size n^{n-1} across the network [32]. As a theoretical example the *long* data type has a maximum possible value of $2^{63} - 1 < 10^{19}$ [63]. Since $16^{15} < 10^{19} < 17^{16}$, when we are dealing with flows or capacities of order n^{n-1}, the largest n possible without risking errors is 16. However, alternative data structures such as Java's *BigInteger* do allow an arbitrary limit on integer size [62], by storing each number as an array of *int*s to the base $2^{31} - 1$ (the maximum *int* value), meaning we are more likely to be dependent on the size of computer memory than any data type limits.

When looking for a rank-maximal or generous stable matching, we describe an alternative approach to finding a maximum flow through a network that does not require exponential weights. This approach is based on using polynomially-bounded weight vectors (henceforth *vector-based weights*) for edge capacities rather than exponential weights. On the surface, performing operations over vector-based weights rather than over equivalent exponential weights, would appear not to improve the time or space complexity of the algorithm, since an exponential weight may naturally be stored as an equivalent array of integers in memory, as in the *BigInteger* case above. However, vector-based weights allow us to explore vector compression that is unavailable in the exponential case. One form of lossless vector compression saves the index and value of each non-zero vector element. This type of vector compression

is used in our experiments in Section 4.7 to show that for randomly-generated instances of size $n = 1000$, we are able to store a network with vector-based weights using approximately 10 times less space than one stored with the equivalent exponential weights. Indeed extrapolating to $n = 100,000$ we achieve an approximate factor of 100 improvement using vector-based weights, with the space required to store a network using exponential weights nearing 1GB. We also show that for a specific instance of size $n = 100,000$, the space required to store exponential weights of a network was over 10GB, whereas the vector-based weights were over $100,000$ times less costly at 0.64MB. Combining these space requirement calculations with the fact that the time complexity of Irving et al.'s [32] $O(nm^2 \log n)$ algorithm to find a rank-maximal stable matching is dominated substantially by the maximum flow algorithm (no other part taking more than $O(m)$ time), it is arguably important to ensure that the network is as small as possible and fits comfortably in RAM.

4.1.3 Contribution

In this chapter we present an $O(nm^2 \log n)$ algorithm to find a rank-maximal stable matching in an instance of SMI using a vector-based weight approach rather than using exponential weights. We also show that a similar process can be used to find a generous stable matching in $O(\min\{m, nd\}^2 d \log n)$ time, where d is the degree of a minimum regret stable matching. Finally, we show that the problems of finding rank-maximal and generous stable matchings in SR are NP-hard. In addition to theoretical contributions we also run experiments using randomly-generated SM instances. In these experiments we compare rank-maximal and generous stable matchings over a range of measures (cost, sex-equal score, degree, number of agents obtaining their first choice and number of agents who obtain a partner in the lower a% of their preference list). An example of this final measure is as follows. If $n = 200$ and $a = 50$ then we record the number of agents who obtain a partner between their 101st and 200th choice inclusive. We additionally compare these profile-based optimal stable matchings with median stable matchings (introduced in Section 2.2.2.2). The median criterion is somewhat unique in that its definition is not based on cost, degree or profile. We were interested in determining whether, in practice, a median stable matching more closely approximates a rank-maximal or a generous stable matching. In these experiments, we find that a generous stable matching typically outperforms both a rank-maximal and a median stable matching when considering cost and sex-equal score measures, and that a median stable matching more closely approximates a generous stable matching in practice.

4.1.4 Structure of the chapter

Section 4.2 introduces preliminary definitions and results used later in this chapter. Section 4.3 gives a description of Irving et al.'s [32] method for finding a rank-maximal stable matching using exponential weights. Sections 4.4 and 4.5 describe the new approach to find a rank-maximal stable matching and a generous stable matching respectively, without the use of exponential weights. Complexity results for rank-maximal and generous stable matchings in SR are presented in Section 4.6. Our experimental evaluation is presented in Section 4.7, whilst future work is discussed in Section 4.8.

4.2 Preliminary definitions and results

In Section 2.2.2.2 we defined the profile of a stable matching. We now generalise this to define a *profile* as a finite vector of integers (positive or negative). Addition over profiles may be defined in the following way. Let $p = \langle p_1, p_2, ..., p_n \rangle$ and $p' = \langle p'_1, p'_2, ..., p'_n \rangle$ be profiles of length n. Then the addition of p' to p is taken pointwise over elements from $1...n$. That is, $p + p' = \langle p_1 + p'_1, p_2 + p'_2, ..., p_n + p'_n \rangle$. We define $p = p'$ if $p_i = p'_i$ for $1 \leq i \leq n$. Now suppose $p \neq p'$. Let k be the first point at which these profiles differ, that is, suppose $p_k \neq p'_k$ and $p_i = p'_i$ for $1 \leq i < k$. Then we define $p \prec p'$ if $p_k < p'_k$. We say $p \preceq p'$ if either $p_k < p'_k$ or $p = p'$. Finally, we define a profile p as *maximum* (*minimum*) among a set of profiles P if for any other profile $p' \in P$, $p \succeq p'$ ($p \preceq p'$). It is trivial to show that an addition or comparison of two profiles would take $O(n)$ time in the worst case (since the length of any profile is bounded by n). Let $p'' = \langle p_1, p_2, ..., p_i, 0, ..., 0 \rangle$ be a profile, where $i \leq n$. Then for ease of description we may shorten this profile to $p'' = \langle p_1, p_2, ..., p_i \rangle$.

As in the previous chapter, we also use the notation and terminology associated with the structure of stable matchings, first introduced in Section 2.2.2.3. We extend this, by defining the *profile* of a rotation as follows. Suppose we have a rotation ρ that, when eliminated, takes us from stable matching M to stable matching M', where M and M' have profiles $p(M) = \langle p_1, p_2, ..., p_n \rangle$ and $p(M') = \langle p'_1, p'_2, ..., p'_n \rangle$ respectively. Then the *profile* of ρ is defined as the net change in profile between M and M', that is, $p(\rho) = \langle p'_1 - p_1, p'_2 - p_2, ..., p'_n - p_n \rangle$. Hence, $p(M') = p(M) + p(\rho)$. It is easy to see that a particular rotation will give the same net change in profile regardless of which stable matching it is eliminated from. For a set of rotations $R = \{\rho_1, \rho_2, ..., \rho_r\}$, we define the *profile* of R as $p(R) = p(\rho_1) + p(\rho_2) + ... + p(\rho_r)$.

We now present several definitions and results regarding the flow over a network.

Let $N = (V, E)$ be a network with source vertex s and sink vertex t. We define the *flow* over a vertex or edge of N as the transmission of a positive (or zero) number of units such that the flow entering the vertex or edge is equal to the flow leaving it and the value of the

flow does not exceed an edge's capacity. The exceptions are the source vertex s which has no flow entering it, and the sink vertex t which has no flow leaving it. The flow f in N is then the sum of all flows leaving s. Finally, a maximum flow is a flow through the network such that f is maximised. An s-t *cut* of N, denoted c_T, is a set of edges (or 'cut' of edges), which if removed, would leave no directed path from s to t. The *capacity* of a cut, $c(c_T)$, is the sum of capacities over the removed edges. A *minimum* s-t *cut* is an s-t cut such that the $c(c_T)$ is minimised.

We now state the *Max Flow-Min Cut* theorem.

Theorem 4.2.1 (Max Flow-Min Cut [17])**.** *The value of a maximum flow through a network N is equal to the capacity of a minimum cut of N.*

By Theorem 4.2.1 we need only find a maximum flow through N in order to find a minimum cut in N.

An *augmenting path* P is a path in N from the source vertex s to the sink vertex t comprising edges of N, but not necessarily in the same direction, such that for each edge $(u, v) \in P$, one of the following must be true:

1. $(u, v) \in E$ and $f(u, v) < c(u, v)$, or;
2. $(v, u) \in E$ and $f(v, u) > 0$.

In order to search for augmenting paths a new network known as the *residual graph* may be constructed. Given a network N and a flow f in N, the *residual graph* relative to N and f, denoted $N_{res}(I, f)$, is defined as follows. The vertex set of $N_{res}(I, f)$ is equal to the vertex set of N. An edge (u, v), known as a forward edge, is added to $N_{res}(I, f)$ with capacity $c(u, v) - f(u, v)$ if $(u, v) \in E$ and $f(u, v) < c(u, v)$. Similarly an edge (u, v), known as a backwards edge, is added to $N_{res}(I, f)$ with capacity $f(v, u)$ if $(v, u) \in E$ and $f(v, u) > 0$. Using a breadth-first search in $N_{res}(I, f)$ we may find an augmenting path or determine that none exists in $O(|E|)$ time. Once an augmenting path P is found we *augment* N in the following way:

- The *residual capacity* c_a is the minimum of the capacities of the edges in P in $N_{res}(I, f)$;
- For each edge $(u, v) \in P$, if (u, v) is a forwards edge, the flow through (u, v) is increased by c_a, whilst if (u, v) is a backwards edge, the flow through (v, u) is decreased by c_a.

Ford and Fulkerson [17] showed that if no augmenting path in N can be found then the flow f in N is maximum. For the remainder of this chapter, maximum flows are found using the

Sleator-Tarjan algorithm [71], which terminates when no augmenting path can be found in N. In Chapter 6, we revisit maximum flows using the Ford-Fulkerson Algorithm [17], which also terminates when no augmenting path can be found.

4.3 Finding a rank-maximal stable matching using exponential weights

In this section we will describe how Irving et al.'s [32] maximum weight stable matching algorithm works and how it can be used to find a rank-maximal stable matching using exponential weights.

4.3.1 Exponential weight network

Irving et al.'s [32] method for finding a maximum weight stable matching involves finding a maximum weight closed subset of the rotation poset. In order to find a maximum weight closed subset of the rotation poset, a network is built and a maximum flow is found over this network.

The rotation digraph is converted to a network $R_n(I)$ as follows. First we add two extra vertices; a source vertex s and a sink vertex t. An edge of capacity ∞ replaces each original edge in the digraph. Since we are finding a rank-maximal stable matching, capacities on other edges of $R_n(I)$ are calculated by converting each profile of a rotation to a single exponential weight. We decide on a weight function of $(2n+1)^{n-i}$ for each person assigned to their ith choice. From this point onwards we refer to the use of this weight function as the *high-weight* scenario, and denote it as w.

Definition 4.3.1. *Given a profile $p = \langle p_1, p_2, ..., p_a \rangle$ such that $|p_1| + |p_2| + ... + |p_a| \leq 2n$ and $1 \leq a \leq n$, define the high-weight function w as,*

$$w(p) = p_1(2n+1)^{n-1} + p_2(2n+1)^{n-2} + ... + p_a(2n+1)^{n-a}.$$

Lemma 4.3.3 shows that when the above function w is used, a matching of maximum weight will be a rank-maximal matching.

Proposition 4.3.2. *Let $p = \langle p_1, p_2, ..., p_n \rangle$ and $p' = \langle p'_1, p'_2, ..., p'_n \rangle$ be profiles such that $|p_1| + |p_2| + ... + |p_n| \leq 2n$ and $|p'_1| + |p'_2| + ... + |p'_n| \leq 2n$. Let $w_i(p) = p_i(2n+1)^{n-i}$ denote the ith term of $w(p)$ and let $w_i^+(p) = \sum_{j=i}^{n} p_j(2n+1)^{n-j}$ denote the sum of $w(p')$ terms for all j such that $i \leq j \leq n$. If $p_i > p'_i$, then $w_i(p) > w_i^+(p')$. Additionally, if i is the first point at which p and p' differ, then $w(p) > w(p')$.*

Proof. Assume $p_i > p'_i$. Then p_i must be at least 1 larger than p'_i since each profile element is an integer by definition. A value of 1 for p_i will contribute $(2n+1)^{n-i}$ to $w_i(p)$ and so it follows that $w_i(p) \geq w_i(p') + (2n+1)^{n-i}$.

Since $(2n+1)^{n-k}$ decreases as k increases and $|p'_1| + |p'_2| + ... + |p'_n| \leq 2n$, the maximum weight contribution that $p'_{i+1}, p'_{i+2}, ..., p'_n$ can make to $w_i^+(p')$ is when $p'_{i+1} = 2n$.

Through the following series of inequalities,

$$\begin{aligned} w_i^+(p') &\leq w_i(p') + 2n(2n+1)^{n-(i+1)} \\ &\leq w_i(p) - (2n+1)^{n-i} + \frac{2n}{2n+1}(2n+1)^{n-i} \\ &\leq w_i(p) + \left(\frac{2n}{2n+1} - 1\right)(2n+1)^{n-i} \\ &< w_i(p) \end{aligned} \tag{4.1}$$

it follows that $w_i(p) > w_i^+(p')$ as required. If i is the first point at which p and p' differ then it follows that $w(p) > w(p')$. □

Lemma 4.3.3. *Let I be an instance of* SMI *and let M be a stable matching in I. If $w(p(M))$ is maximum amongst all stable matchings of I, where $p(M)$ is the profile of M, then M is a rank-maximal stable matching.*

Proof. Suppose $w(p(M))$ is maximum amongst all stable matchings of I. Now, assume for contradiction that M is not rank-maximal. Then, there exists some stable matching M' in I such that M' lexicographically larger than M. Let i be the first point at which $p(M) = \langle p_1, p_2, ..., p_n \rangle$ and $p(M') = \langle p'_1, p'_2, ..., p'_n \rangle$ differ. Since M' is lexicographically larger than M we know that $p'_i > p_i$ and by Proposition 4.3.2 it follows that $w(p(M')) > w(p(M))$.

But this contradicts the fact that $w(p(M))$ is maximum over all stable matchings of I. Therefore our assumption that M is not rank-maximal is false, as required. □

We now continue describing Irving et al.'s technique for finding a maximum weight closed subset of the rotation poset. The rotations are divided into positive and negative vertices as follows. A rotation ρ is positive if $w(p(\rho)) > 0$ and negative if $w(p(\rho)) < 0$. A directed edge is added from the source to each negative vertex and is given a capacity equal to $|w(p(\rho))|$. A directed edge is also added between each positive vertex and t with capacity $w(p(\rho))$. Recall the example SMI instance I_0 in Figure 2.1 of Section 2.2.2 with associated rotation poset and rotation digraph shown in Figure 2.3. The high-weight network of instance I_0 is denoted $R_n(I_0)$ and is shown in Figure 4.1. In this figure, each edge e has a pair of associated integers e_1/e_2 where e_1 is the flow over e and e_2 is the capacity of e.

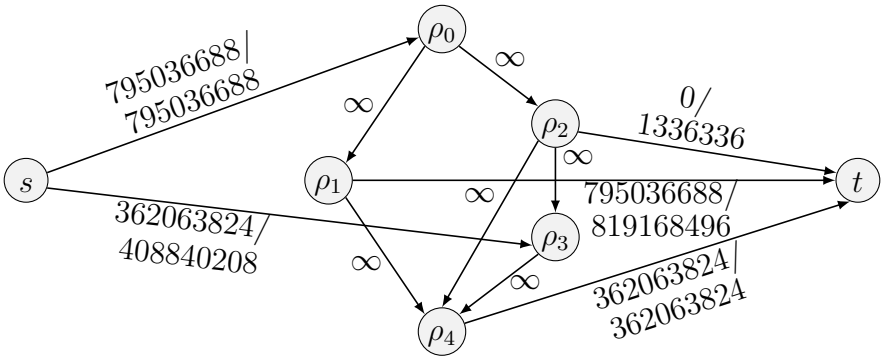


Figure 4.1: The high-weight network $R_n(I_0)$.

4.3.2 Maximum weight closed subset of $R_p(I)$

We wish to find a minimum cut in $R_n(I)$ as this will allow us to calculate the maximum weight closed subset of $R_p(I)$. For future reference, we denote by $R_{res}(I, f)$ the residual graph of $R_n(I)$. As described in Section 4.2 we need only find a maximum flow through $R_n(I)$ in order to find a minimum cut in $R_n(I)$. Irving et al. [32] used the Sleator-Tarjan algorithm [71] to find a maximum flow. This algorithm completes when no augmenting path exists in $R_n(I)$ with respect to the final flow.

In Figure 4.1 we show the high-weight network $R_n(I_0)$ with a maximum flow. There is one minimum cut, $c_T = \{(s, \rho_0), (\rho_4, t)\}$. Note that c_T is a cut since removing these edges leaves no path from s to t. By Theorem 4.2.1, it is also a minimum cut since the capacity of c_T (1157100512) is equal to the value of a maximum flow. For this cut c_T we list every rotation ρ such that (ρ, t) is an edge in $R_n(I)$ and $(\rho, t) \notin c_T$. Then a maximum weight closed subset of the rotation poset is given by this set of rotations and their predecessors. c_T has associated closed subset of $\{\rho_0, \rho_1, \rho_2\}$ which is precisely a maximum weight closed subset of $R_p(I_0)$. The man-optimal stable matching of I_0 is

$$M = \{(m_1, w_5), (m_2, w_3), (m_3, w_8), (m_4, w_6), (m_5, w_7), (m_6, w_1), (m_7, w_2), (m_8, w_4)\}.$$

By eliminating rotations $\{\rho_0, \rho_1, \rho_2\}$ from the man-optimal stable matching, we find the rank-maximal stable matching

$$M' = \{(m_1, w_3), (m_2, w_6), (m_3, w_1), (m_4, w_8), (m_5, w_7), (m_6, w_5), (m_7, w_2), (m_8, w_4)\}.$$

The following Theorem summarises the work in this section.

Theorem 4.3.4 ([32]). *Let I be an instance of* SMI. *A rank-maximal stable matching M of I can be found in $O(nm^2 \log n)$ time using weights that are exponential in n.*

An alternative to high-weight values when looking for a rank-maximal stable matching, is

to use a new approach, involving polynomially-bounded weight vectors, to find a maximum weight closed subset of rotations. This is the focus of the rest of this chapter.

4.4 Finding a rank-maximal stable matching using polynomially-bounded weight vectors

4.4.1 Strategy

Following a similar strategy to Irving et al. [32], we aim to show that we can return a rank-maximal stable matching in $O(nm^2 \log n)$ time without the use of exponential weights. The process we follow is described in the steps below.

1. Calculate man-optimal and woman-optimal stable matchings using the Extended Gale-Shapley Algorithm – $O(m)$ time;
2. Find all rotations using the minimal differences algorithm – $O(m)$ time;
3. Build the rotation digraph and network – $O(m)$ time;
4. Find a minimum cut of the network in $O(nm^2 \log n)$ time without reverting to high weights;
5. Use this cut to find a maximum profile closed subset S of the rotation poset – $O(m)$ time;
6. Eliminate the rotations of S from the man-optimal matching to find the rank-maximal stable matching.

In the next section we discuss required adaptions to the high-weight procedure.

4.4.2 Vb-networks and vb-flows

In this section we look at steps in the strategy to find a rank-maximal stable matching without the use of exponential weights (Section 4.4.1) which either require adaptations or further explanation.

In Step 6 of our strategy we eliminate the rotations of a maximum profile closed subset of the rotation poset from the man-optimal stable matching. We now present Lemma 4.4.1, an analogue of [25, Corollary 3.6.1], which shows that eliminating a maximum profile closed subset of the rotation poset from the man-optimal stable matching results in a rank-maximal stable matching.

Lemma 4.4.1. *Let I be an instance of* SMI *and let M_0 be the man-optimal stable matching in I. A rank-maximal stable matching M may be obtained by eliminating a maximum profile closed subset S of the rotation poset from M_0.*

Proof. Let $R_p(I)$ be the rotation poset of I. By Gusfield and Irving [25, Theorem 2.5.7], there is a 1-1 correspondence between closed subsets of $R_p(I)$ and the stable matchings of I. Let S be a maximum profile closed subset of the rotation poset $R_p(I)$ and let M be the unique corresponding stable matching. Then, $p(M) = p(M_0) + \sum_{\rho_i \in S} p(\rho_i)$. Suppose M is not rank-maximal. Then there is a stable matching M' such that $p(M') \succ p(M)$. As above, M' corresponds to a unique closed subset S' of the rotation poset. Also $p(M') = p(M_0) + \sum_{\rho_i \in S'} p(\rho_i)$. But $p(M') \succ p(M)$ and so S cannot be a maximum profile closed subset of $R_p(I)$, a contradiction. □

Steps 3 and 4 of our strategy are the only places where we are required to check that it is possible to directly substitute an operation involving large weights taking $O(n)$ time with a comparable profile operation taking $O(n)$ time.

The first deviation from Gusfield and Irving's method (described in Section 4.1) is in the creation of a *vector-based network* (abbreviated to *vb-network*). For ease of description we denote this new vb-network as $R'_n(I)$ to distinguish it from the high-weight version $R_n(I)$. We now define a *vb-capacity* in $R'_n(I)$ which is of similar notation to that of a profile.

Definition 4.4.2. *In a vb-network $R'_n(I)$, the vector-based capacity (vb-capacity) of an edge e is a vector $\boldsymbol{c}(e) = \langle c_1, c_2, ..., c_n \rangle$, where n is the number of men or women in I and $c_i \geq 0$ for $1 \leq i \leq n$.*

As before we add a source s and sink t vertex to the rotation digraph. We replace each original digraph edge with an edge with vb-capacity $\langle \infty, \infty, ..., \infty \rangle$ (∞ repeated n times). For convenience these edges are marked with '$\boldsymbol{\infty}$' in vb-network diagrams. The definition of a positive and negative rotation is also amended. Let ρ have profile $p(\rho) = \langle p_1, p_2, ..., p_n \rangle$. Let p_k be the first non-zero profile element where $1 \leq k \leq n$. We now define a positive rotation ρ as a rotation where $p_k > 0$, and a negative rotation is one where $p_k < 0$. Define the absolute value operation, denoted $|p(\rho)|$, as follows. If $p_k > 0$, then leave all elements unchanged. If $p_k < 0$, then reverse the sign of all non-zero profile elements. Figure 4.2 shows the profile and absolute profile for each rotation of I_0. Then we add a directed edge to the vb-network from s to each negative rotation vertex ρ with a vb-capacity of $|p(\rho)|$ and a directed edge from each positive rotation vertex ρ to t with a vb-capacity of $p(\rho)$.

Let $e = (u, v)$ be an edge in $R'_n(I)$ with capacity $\boldsymbol{c}(e)$. We define a *vb-flow* over e, denoted $\boldsymbol{f}(e)$, in the following way.

ρ_0: (m_1, w_5) (m_3, w_8)
ρ_1: (m_1, w_8) (m_2, w_3) (m_4, w_6)
ρ_2: (m_3, w_5) (m_6, w_1)
ρ_3: (m_7, w_2) (m_5, w_7)
ρ_4: (m_3, w_1) (m_5, w_2)

(a) Rotations for instance I_0.

$p(\rho_0) = \langle -2, 1, 1, 1, 0, -1\rangle$
$p(\rho_1) = \langle 2, 0, -1, -1, -1, -2, 1, 2\rangle$
$p(\rho_2) = \langle 0, 0, 1, -1\rangle$
$p(\rho_3) = \langle -1, 0, 1, 1, -1\rangle$
$p(\rho_4) = \langle 1, -2, 0, 0, 0, 1\rangle$

(b) Rotation profiles.

$|p(\rho_0)| = \langle 2, -1, -1, -1, 0, 1\rangle$
$|p(\rho_1)| = \langle 2, 0, -1, -1, -1, -2, 1, 2\rangle$
$|p(\rho_2)| = \langle 0, 0, 1, -1\rangle$
$|p(\rho_3)| = \langle 1, 0 - 1, -1, 1\rangle$
$|p(\rho_4)| = \langle 1, -2, 0, 0, 0, 1\rangle$

(c) Absolute rotation profiles.

Figure 4.2: The profile and absolute profile for rotations of I_0.

Definition 4.4.3. *In a vb-network $R'_n(I)$, the vector-based flow (vb-flow) over an edge e is a vector $\boldsymbol{f}(e) = \langle f_1, f_2, ..., f_n\rangle$, where $\sum_{i=1}^{n} |f_i| \leq 2n$.*

By Definition 4.4.3, the sum of the absolute values of the elements of a flow over an edge cannot exceed $2n$, which implies that each flow element f_i satisfies $-2n \leq f_i \leq 2n$. We will use this fact in the next section when proving the equivalence of the vector-based and high-weight approaches. We also remark that it is possible for $f_i < 0$ for some i $(2 \leq i \leq n)$ and for $\boldsymbol{f}(e)$ to be a positive vb-flow. For example $\langle 0, 0, 0\rangle \prec \langle 0, 1, -10\rangle$, and hence we are not bounding each individual element of a vb-flow over an edge by a minimum of zero.

We may now define a vb-flow over $R'_n(I)$ as follows.

Definition 4.4.4. *In a vb-network $R'_n(I) = (V, E)$, a vector-based flow (vb-flow) is a function $f : E \to \mathbb{R}^n$ such that* [2]*,*

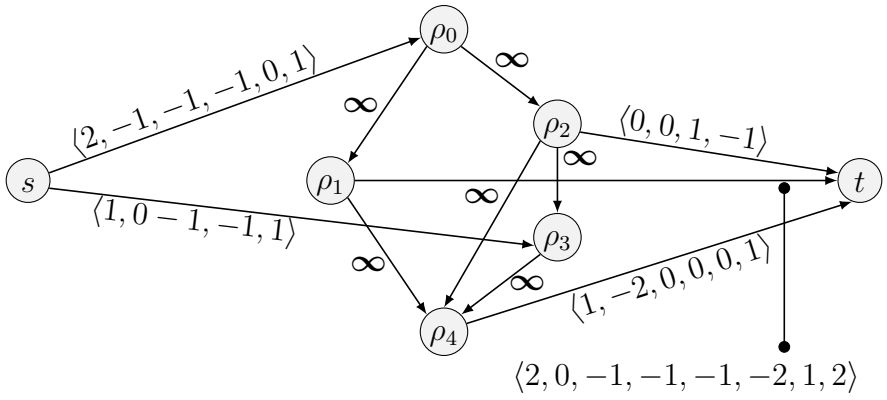
i) (vb-capacity) $\boldsymbol{f}(e) \succeq \mathbf{0}$ and $\boldsymbol{f}(e) \preceq \boldsymbol{c}(e)$ for all $e \in E$;

ii) (vb-conservation) $\sum_{(u,v)\in E} \boldsymbol{f}(u, v) = \sum_{(v,w)\in E} \boldsymbol{f}(v, w)$ for all $v \in V \backslash \{s, t\}$.

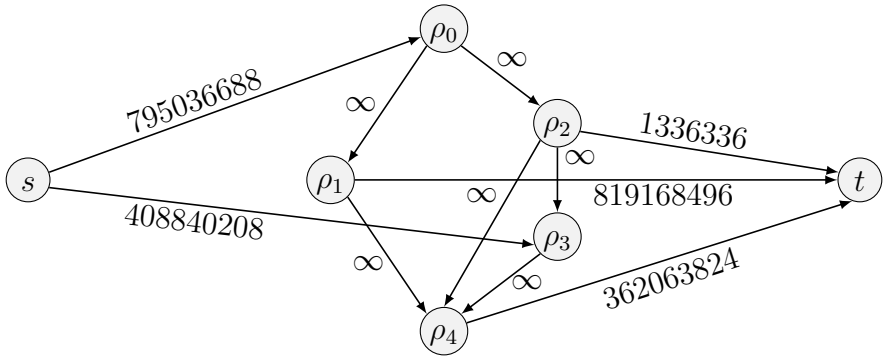
Vb-flows are non-negative by the vb-capacity constraint.

In addition we define the following notation and terminology for vb-flows. Let $\boldsymbol{f}$ and be a vb-flow in $R'_n(I) = (V, E)$. Define $\text{val}(\boldsymbol{f}) = \sum\{f(s, v) : v \in V \wedge (s, v) \in E\}$. We

[2] Sleator and Tarjan's algorithm [71] (used later in this section) may be used in a way that assumes integer value flows, and hence for the rest of this chapter we assume vb-flow value elements will only ever be integers. That is, a vb-flow is a function $f : E \to \mathbb{N}^n$.



(a) Vector-based network $R'_n(I_0)$.



(b) High-weight network $R_n(I_0)$.

Figure 4.3: Vector-based network $R'_n(I_0)$ and network $R_n(I_0)$ with both vector-based and high-weight capacities respectively.

define a maximum vb-flow $\boldsymbol{f}$ to be a vb-flow such that there is no other vb-flow $\boldsymbol{f'}$ where val($\boldsymbol{f'}$) > val($\boldsymbol{f}$).

The vb-network $R'_n(I_0)$ and the corresponding high-weight version $R_n(I_0)$ are shown in Figure 4.3. In order to translate *vector-based values* (vb-values) to high-weight values we use the same formula as for profiles. That is, $(2n+1)^{n-i}$ for each man or woman assigned to their ith choice [32]. As an example, the vb-value $\langle 0,0,1,-1\rangle$ for rotation ρ_2 translates to a high-weight value of $0*17^7+0*17^6+1*17^5-1*17^4 = 1336336$.

An augmenting path in $R'_n(I)$ has an analogous definition to the standard definition of an augmenting path. The *vector-based residual network* (vb-residual network) $R'_{res}(I,\boldsymbol{f})$ of $R'_n(I)$ with vb-capacities is created in the same way as the residual network $R_{res}(I,f)$ of $R_n(I)$. A cut in $R'_n(I)$, denoted c'_T, is defined in a similar way to a cut in $R_n(I)$ and has capacity $\boldsymbol{c}(c'_T) = \sum \boldsymbol{c}(e)$ where e is an edge in c'_T. Where the flow in $R_n(I)$ is equivalent to the vb-flow in $R'_n(I)$, we want to show that a maximum flow in $R_n(I)$ is also equivalent to a maximum vb-flow in $R'_n(I)$, and that the Max Flow-Min Cut Theorem holds for vb-networks.

4.4.3 Rank-maximal stable matchings

In this section, we show how we are able to use our vb-network to find a rank-maximal stable matching. First, we show that the Max Flow-Min Cut Theorem (Theorem 4.2.1) can be extended to a vb-network. Next, we prove that, analogous to the exponential weight case, a maximum profile closed subset of the rotation poset may be found by obtaining a minimum cut of the vb-network. Finally, we show that Sleator and Tarjan's Max Flow algorithm [71] may be adapted to work with vb-networks, and that we are able to find a rank-maximal stable matching in $O(nm^2 \log n)$ time using polynomially-bounded weight vectors.

In Lemma 4.4.6 we show that vb-flows in $R'_n(I)$ correspond to high-weight flows in $R_n(I)$. Let $\boldsymbol{f}$ be a vb-flow in a vb-network, where $\text{val}(\boldsymbol{f}) = \langle f_1, f_2, ..., f_n \rangle$ and let c'_T be a cut where $\boldsymbol{c}(c'_T) = \langle c'_{T_1}, c'_{T_2}, ..., c'_{T_n} \rangle$.

Proposition 4.4.5. *Let $\boldsymbol{f}$ and $\boldsymbol{f'}$ be vb-flows. Let $w_i(val(\boldsymbol{f}))$ denote the ith term of $w(val(\boldsymbol{f}))$ and let $w_i^+(val(\boldsymbol{f'}))$ denote the sum of $w(val(\boldsymbol{f'}))$ terms for all j such that $i \leq j \leq n$. If $f_i > f'_i$, then $w_i(val(\boldsymbol{f})) > w_i^+(val(\boldsymbol{f'}))$. Additionally, if i is the first point at which $\boldsymbol{f}$ and $\boldsymbol{f'}$ differ, then $w(val(\boldsymbol{f})) > w(val(\boldsymbol{f'}))$.*

Identical results hold for vb-capacities.

Proof. Recall from the footnote of Definition 4.4.4 that vb-flow values must contain only integer elements.

The only difference between the structure of a profile p and a vb-flow $\boldsymbol{f}$ is that each profile element must take a value between 0 and $2n$ inclusive, whereas the lower bound of vb-flow elements is relaxed to $-2n$. This difference does not affect the validity of Proposition 4.3.2 and so we may use identical reasoning to show that if $f_i > f'_i$, then $w_i(\text{val}(\boldsymbol{f})) > w_i^+(\text{val}(\boldsymbol{f'}))$, and if i is the first point at which $\boldsymbol{f}$ and $\boldsymbol{f'}$ differ, then $w(\text{val}(\boldsymbol{f})) > w(\text{val}(\boldsymbol{f'}))$.

Since vb-capacities have an identical structure to vb-flows the all results also hold for the vb-capacity case. □

Lemma 4.4.6. *Let $\boldsymbol{f}$ and $\boldsymbol{f'}$ be vb-flows in $R'_n(I)$. Then $val(\boldsymbol{f}) \prec val(\boldsymbol{f'})$ if and only if $w(val(\boldsymbol{f})) < w(val(\boldsymbol{f'}))$.*

Additionally, let c'_T and c''_T be cuts in $R'_n(I)$. Then $\boldsymbol{c}(c'_T) \prec \boldsymbol{c}(c''_T)$ if and only if $w(\boldsymbol{c}(c'_T)) < w(\boldsymbol{c}(c''_T))$.

Proof. Suppose that $\text{val}(\boldsymbol{f}) \prec \text{val}(\boldsymbol{f'})$. We know $\text{val}(\boldsymbol{f}) \neq \text{val}(\boldsymbol{f'})$, and at the first point i at which $\text{val}(\boldsymbol{f})$ and $\text{val}(\boldsymbol{f'})$ differ $f_i < f'_i$. By Proposition 4.4.5, $w(\text{val}(\boldsymbol{f})) < w(\text{val}(\boldsymbol{f'}))$ as required.

Now assume $w(\text{val}(\boldsymbol{f})) < w(\text{val}(\boldsymbol{f'}))$ and suppose for contradiction that $\text{val}(\boldsymbol{f}) \succeq \text{val}(\boldsymbol{f'})$. If $\text{val}(\boldsymbol{f}) = \text{val}(\boldsymbol{f'})$ then clearly $w(\text{val}(\boldsymbol{f})) = w(\text{val}(\boldsymbol{f'}))$ a contradiction. Therefore suppose

$\text{val}(\boldsymbol{f}) \succ \text{val}(\boldsymbol{f'})$. Then, we can use identical arguments to the preceding paragraph to prove that $w(\text{val}(\boldsymbol{f})) > w(\text{val}(\boldsymbol{f'}))$. But this contradicts our original assumption that $w(\text{val}(\boldsymbol{f})) < w(\text{val}(\boldsymbol{f'}))$. Therefore, $\text{val}(\boldsymbol{f}) \prec \text{val}(\boldsymbol{f'})$.

Using identical reasoning to the vb-flow case for the vb-capacity case, we can show that $\boldsymbol{c}(c'_T) \prec \boldsymbol{c}(c''_T)$ if and only if $w(\boldsymbol{c}(c'_T)) < w(\boldsymbol{c}(c''_T))$. □

Lemma 4.4.7 shows that if there is no augmenting path in a vb-network, then the vb-flow existing in this network is maximum.

Lemma 4.4.7. *Let I be an instance of* SMI *and let $R'_n(I)$ and $R_n(I)$ define the vb-network and network of I respectively. For all vb-flows and vb-capacities we define a corresponding flow or capacity for $R_n(I)$ using the high-weight function w (Definition 4.3.1). Suppose $\boldsymbol{f}$ is a vb-flow in $R'_n(I)$ that admits no augmenting path. Then $\boldsymbol{f}$ is a maximum vb-flow in $R'_n(I)$.*

Proof. Let f be the flow corresponding to $\boldsymbol{f}$ in $R_n(I)$. First, we show that f is a maximum flow in $R_n(I)$. Suppose for contradiction that f is not a maximum flow. Then there must exist an augmenting path relative to f in $R_n(I)$. Let E_P denote the edges involved in this augmenting path. Then, for each edge $(u, v) \in E_P$ of this augmenting path either,

- $f(u, v) < c(u, v)$, in which case the vb-flow $\boldsymbol{f}(u, v)$ through edge $(u, v) \in R'_n(I)$ may increase by $\langle 0, 0, ..., 1\rangle$, or;
- $f(v, u) > 0$, and so the vb-flow $\boldsymbol{f}(v, u)$ through edge $(v, u) \in R'_n(I)$ may decrease by $\langle 0, 0, ..., 1\rangle$.

Therefore, there exists an augmenting path relative to $\boldsymbol{f}$ in $R'_n(I)$. But this contradicts the fact that $\boldsymbol{f}$ is a vb-flow in $R'_n(I)$ that admits no augmenting path. Hence our assumption that f is not a maximum flow in $R_n(I)$ is false.

We now show that $\boldsymbol{f}$ is a maximum vb-flow in $R'_n(I)$. Suppose for contradiction that this is not the case. Then, there must exist a vb-flow $\boldsymbol{f'}$ such that $\text{val}(\boldsymbol{f'}) \succ \text{val}(\boldsymbol{f})$. By Lemma 4.4.6, $w(\text{val}(\boldsymbol{f'})) > w(\text{val}(\boldsymbol{f}))$. Let f' be the flow corresponding to $\boldsymbol{f'}$ in $R_n(I)$. Then we have the following inequality:

$$\text{val}(f') = w(\text{val}(\boldsymbol{f'})) > w(\text{val}(\boldsymbol{f})) = \text{val}(f)$$

contradicting the fact that f is a maximum flow in $R_n(I)$. Therefore $\boldsymbol{f}$ is a maximum vb-flow in $R'_n(I)$. □

This means if we use any Max Flow algorithm that terminates with no augmenting paths (such as the Ford-Fulkerson Algorithm [17] adapted to work with vb-flows and vb-capacities) we have found a maximum flow in a vb-network.

We now show that the Max Flow-Min Cut Theorem can be extended to a vb-network.

Theorem 4.4.8. *Let I be an instance of* SMI *and let $R'_n(I) = (V', E')$ and $R_n(I)$ define the vb-network and network of I respectively. For all vb-flows and vb-capacities we define a corresponding flow or capacity for $R_n(I)$ using the high-weight function w (Definition 4.3.1). Let $\boldsymbol{f}$ be a maximum vb-flow through $R'_n(I)$ and c'_T be a minimum cut of $R'_n(I)$. Then $\boldsymbol{c}(c'_T) = val(\boldsymbol{f})$.*

Proof. Given $\boldsymbol{f}$ is a maximum vb-flow in $R'_n(I)$, we define a cut c'_T in $R'_n(I)$ in the following way. A *partial augmenting path* is an augmenting path from the source vertex s to vertex $u \neq t$ in V with respect to $\boldsymbol{f}$ in $R'_n(I)$. Note that, by Lemma 4.4.7 no augmenting path from s to t may exist at this point since $\boldsymbol{f}$ is a maximum vb-flow. Let A be the set of reachable vertices along partial augmenting paths, and let $B = V \backslash A$. Then $s \in A$ and $t \in B$. Define $c'_T = \{(u, v) : u \in A, v \in B\}$. Then c'_T is a cut in $R'_n(I)$. Since there is no partial augmenting path extending between vertices in A and vertices in B, we know that for vertices $u \in A$ and $v \in B$,

- if $(u, v) \in E'$ then $\boldsymbol{f}(u, v) = \boldsymbol{c}(u, v)$, and;
- if $(v, u) \in E'$ then $\boldsymbol{f}(v, u) = \mathbf{0}$.

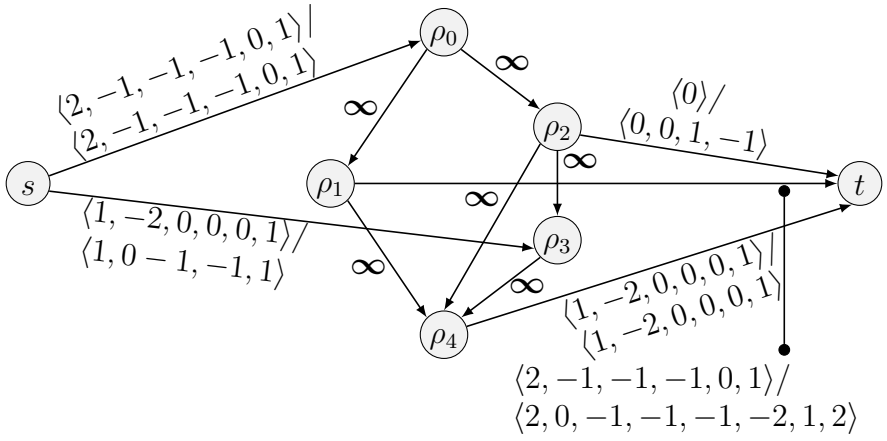
Therefore val$(\boldsymbol{f}) = \boldsymbol{c}(c'_T)$ (see, for example, Cormen et al. [10, p. 721, Lemma 26.4] for a proof of this statement).

Let f be the flow corresponding to $\boldsymbol{f}$ in $R_n(I)$. We now show that c'_T is a minimum cut in $R'_n(I)$. Suppose for contradiction that this is not the case. Then there must exist a cut c''_T in $R'_n(I)$ such that $\boldsymbol{c}(c''_T) \prec \boldsymbol{c}(c'_T)$. By Lemma 4.4.6, $w(\boldsymbol{c}(c''_T)) < w(\boldsymbol{c}(c'_T))$ and so we have the following inequality.

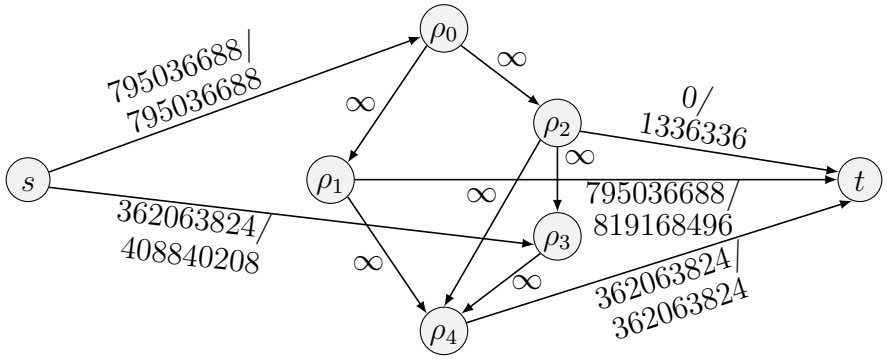
$$c(c''_T) = w(\boldsymbol{c}(c''_T)) < w(\boldsymbol{c}(c'_T)) = w(\text{val}(\boldsymbol{f})) = \text{val}(f)$$

But then c''_T is a cut with smaller capacity than val(f) in $R_n(I)$ contradicting the Max Flow-Min Cut Theorem in $R_n(I)$. Hence c'_T is a minimum cut in $R'_n(I)$. □

As an example, Figure 4.4a shows a maximum flow over the vb-network $R'_n(I_0)$, and Figure 4.4b shows these vb-flows translated into the high-weight network $R_n(I_0)$. Similar to the high weight case, in $R'_n(I_0)$, each edge e has a pair of associated integers e_1/e_2 where e_1 is the vb-flow over e and e_2 is the vb-capacity of e. In Figure 4.4a each edge flow is positive, that is, the first non-zero element of each vb-flow is positive as required. The maximum vb-flow shown in this figure has saturated both edge (s, ρ_0) leaving the source s and edge (ρ_4, t) entering the sink t. Let $c'_T = \{(s, \rho_0), (\rho_4, t)\}$. Clearly c'_T constitutes a cut as removing these



(a) Vector-based network $R'_n(I_0)$.



(b) High-weight network $R_n(I_0)$.

Figure 4.4: Maximum vb-flow and flow in the networks $R'_n(I_0)$ and $R_n(I_0)$ with both vector-based and high-weight capacities respectively.

edges leaves no path from s to t. Since the vb-capacity of c'_T ($\langle 3,-3,-3,1,-1,0,2\rangle$) is equal to the value of a maximum vb-flow of $R'_n(I_0)$, c'_T is also a minimum cut, by Theorem 4.4.8. The equivalent situation in the high-weight network is shown in Figure 4.4b.

In order to determine which rotations must be eliminated from the man-optimal stable matching, we must first determine a maximum profile closed subset of the rotation poset. As with the example in Section 4.3, we find the positive vertices which have edges into t that are not in c'_T. These are ρ_1 and ρ_2. In Theorem 4.4.9 we will prove that a maximum profile closed subset of the vb-network comprises these vertices and their predecessors: $\{\rho_0, \rho_1, \rho_2\}$. It was shown in Section 4.3 that this was indeed a maximum weight closed subset of $R_p(I_0)$.

The following theorem is a restatement of Gusfield and Irving's theorem [25, p. 130] proving that a maximum profile closed subset of the rotation poset $R_p(I)$ can be found by finding a minimum s-t cut of the vb-network $R'_n(I)$, when using a vector-based weight function.

Theorem 4.4.9. *Let I be an instance of* SMI *and let $R_p(I)$, $R_d(I)$ and $R'_n(I)$ denote the rotation poset, rotation digraph and vb-network of I respectively. Let c'_T be a minimum s-t*

cut in $R'_n(I)$, and let $\mathcal{P}_{c'_T}$ be the positive vertices of the network whose edges into t are not in c'_T. Then the vertices $\mathcal{P}_{c'_T}$ and their predecessors define a maximum profile closed subset of $R_p(I)$. Further $\mathcal{P}_{c'_T}$ is exactly the set of positive vertices of this closed subset of the rotation poset.

Proof. Let S be an arbitrary set of rotations in I and define $w(S) = \sum_{s_i \in S} p(s_i)$, that is, $w(S)$ is the total vector-based weight of these rotations. Let $\mathcal{P}$ be the set of all positive rotations. For any set of rotations $S \subseteq \mathcal{P}$, let $N(S)$ be the set of all negative rotation predecessors of S in the rotation digraph $R_d(I)$. Let Q denote a maximum profile closed subset of $R_d(I)$. In order for a negative rotation vertex to exist in Q it must precede at least one positive rotation vertex, otherwise Q could not be of maximum weight. Hence Q can be found by maximising $w(S) + w(N(S))$ over all subsets of $S \subseteq \mathcal{P}$.

We now show that $w(S) + w(N(S))$ is equivalent to $w(S) - |w(N(S))|$ according to our vector-based weight function. We know that $w(N(S))$ is negative (i.e. the first non-zero element of $w(N(S))$ is negative) and therefore taking the absolute value of $w(N(S))$ will reverse the signs of all non-zero elements. Taking the negative of $|w(N(S))|$ reverses the element signs once more and so we have $w(S) + w(N(S)) = w(S) - |w(N(S))|$.

Therefore we may say that Q can be found by maximising $w(S) - |w(N(S))|$ over all subsets of $S \subseteq \mathcal{P}$. But by maximising this function, we also minimise $w(\mathcal{P}) - (w(S) - |w(N(S))|) = w(\mathcal{P} \backslash S) + |w(N(S))|$. That is, we are minimising the total weight of the positive rotations not in S added to the absolute value of the negative rotations that are S's predecessors. This becomes clearer when looking at the vb-network $R'_n(I)$.

Let $\boldsymbol{c}(c'_T)$ denote the capacity of the minimum cut c'_T. We want to show that $\boldsymbol{c}(c'_T)$ is at least as small as $w(\mathcal{P} \backslash S) + |w(N(S))|$ for any $S \subseteq \mathcal{P}$. We can find an upper bound for $\boldsymbol{c}(c'_T)$ by doing the following. If we have a set of edges c''_T that comprises *(1)* all edges from s to vertices in $N(S)$, and *(2)* all edges from vertices in $\mathcal{P} \backslash S$ to t, then c''_T is certainly a cut since there can be no flow through $R'_n(I)$. Moreover $\boldsymbol{c}(c''_T) = w(\mathcal{P} \backslash S) + |w(N(S))|$ and therefore, $\boldsymbol{c}(c'_T) \leq w(\mathcal{P} \backslash S) + |w(N(S))|$ for any $S \subseteq \mathcal{P}$. Now let $S^* \subseteq \mathcal{P}$ be the set of positive rotation vertices that have edges into t that are not in c'_T. Then c'_T must contain all edges from $\mathcal{P} \backslash S^*$ to t. Since c'_T has finite capacity all the edges within it must also have finite capacity and consequently c'_T must also contain all edges in $N(S^*)$ (since it cannot contain any edges with capacity $\boldsymbol{\infty}$). Therefore,

$$\boldsymbol{c}(c'_T) = w(\mathcal{P} \backslash S^*) + |w(N(S^*))| \leq w(\mathcal{P} \backslash S) + |w(N(S))|$$

for all $S \subseteq \mathcal{P}$. Hence, $\mathcal{P}_{c'_T} = S^*$ and, $\mathcal{P}_{c'_T}$ and their predecessors define a maximum profile closed subset of the rotation poset $R_p(I)$. □

It remains to show that we can adapt Sleator and Tarjan's $O(m^2 \log n)$ Max Flow algorithm to work with vb-networks. This is shown in Lemma 4.4.10.

Lemma 4.4.10. *Let I be an instance of* SMI *and let $R'_n(I)$ be a vb-network. We can use a version of Sleator and Tarjan's Max Flow algorithm [71] adapted to work with vb-networks in order to find a maximum flow $\boldsymbol{f}$ of $R'_n(I)$ in $O(nm^2 \log n)$ time.*

Proof. A *blocking flow* in the high-weight setting is a flow such that each path through the network from s to t has a saturated edge. Note that this is different from a maximum flow, since a blocking flow may still allow extra flow to be pushed from s to t using backwards edges in the residual graph. The Sleator-Tarjan algorithm [71] is an adapted version of Dinic's algorithm [11] which improves the time complexity of finding a blocking flow. This is achieved by the introduction of a new *dynamic tree structure*.

The following operations are required from Sleator and Tarjan's *dynamic tree structure* in the max flow setting [71]: *link*, *capacity*, *cut*, *mincost*, *parent*, *update* and *cost*. Each of these processes (not described here) consists of straightforward graph operations (such as adding a parent vertex, deleting an edge etc.) and comparisons, additions, subtractions and updating of edge capacities and flows. Since we have a vector-based interpretation of comparison, addition and subtraction operations, it is possible to adapt Sleator and Tarjan's Max Flow algorithm to work in the vector-based setting.

Sleator and Tarjan's algorithm [71] terminates with a flow that admits no augmenting path. Let $\boldsymbol{f}$ be a vb-flow given at the termination of Sleator and Tarjan's algorithm, as applied to $R'_n(I)$. Since $\boldsymbol{f}$ admits no augmenting path, it follows, by Lemma 4.4.7 that $\boldsymbol{f}$ is a maximum vb-flow in $R'_n(I)$. Sleator and Tarjan's algorithm runs in $O(m^2 \log n)$ time assuming constant time operations for comparison, addition and subtraction. However, in the vb-flow setting, each of these operations takes $O(n)$ time in the worst case. Therefore, using the Sleator and Tarjan algorithm, we have a total time complexity of $O(nm^2 \log n)$ to find a maximum flow of a vb-network. □

Finally, we now show that there is an $O(nm^2 \log n)$ algorithm for finding a rank-maximal stable matching in an instance of SMI, based on polynomially-bounded weight vectors.

Theorem 4.4.11. *Given an instance I of* SMI *there is an $O(nm^2 \log n)$ algorithm to find a rank-maximal stable matching in I that is based on polynomially-bounded weight vectors.*

Proof. We use the process described in Section 4.4.1. All operations from this are well under the required time complexity except number 4. Let $R'_n(I) = (V', E')$ be a vb-network of I. Here, bounds on the number of edges and number of vertices are identical to the maximum weight case, that is $|E'| \leq m$ and $|V'| \leq m$. This is because, despite having alternative

versions of flows and capacities, we have an identical graph structure to the high-weight case.

By using the adaption of Sleator and Tarjan's Max Flow algorithm from Lemma 4.4.10 we achieve an overall time complexity of $O(nm^2 \log n)$ to find a maximum vb-flow $\boldsymbol{f}$ in $R'_n(I)$. Let c'_T denote a minimum cut in $R'_n(I)$. By Theorem 4.4.8, $\boldsymbol{c}(c'_T) = \text{val}(\boldsymbol{f})$. Therefore using the process described in Section 4.4.1, with vector-based adaptations, we can find a rank-maximal stable matching in $O(nm^2 \log n)$ time without reverting to high weights. □

Hence we have an $O(nm^2 \log n)$ algorithm for finding a rank-maximal stable matching, without reverting to high-weight operations.

4.5 Generous stable matchings

We now show how to adapt the techniques in Section 4.4 to the generous setting. Let I be an instance of SMI and let M be a matching in I with profile $p(M) = \langle p_1, p_2, ..., p_k \rangle$. Recall the *reverse profile* $p_r(M)$ is the vector $p_r(M) = \langle p_k, p_{k-1}, ..., p_1 \rangle$.

As with the rank-maximal case, we wish to use an approach to finding a generous stable matching that does not require exponential weights. Recall n is the number of men in I. A simple $O(n)$ operation on a matching profile allows the rank-maximal approach described in the previous section to be used in the generous case.

Let M be a stable matching in I with degree k and profile $p(M) = \langle p_1, p_2, ..., p_{k-1}, p_k \rangle$. Since we wish to minimise the reverse profile $p_r(M) = \langle p_k, p_{k-1}, ..., p_1 \rangle$ we can simply maximise the reverse profile where the value of each element is negated. A short proof of this is given in Proposition 4.5.1. We denote this profile by $p'_r(M)$, where

$$p'_r(M) = \langle -p_k, -p_{k-1}, ..., -p_2, -p_1 \rangle. \tag{4.2}$$

Thus in general, profile elements corresponding to $p'_r(M)$ can take negative values. All profile operations described in Section 4.3 still apply to profiles of this type.

Proposition 4.5.1. *Let M be a matching in an instance I of* SMI. *Then,*

$$M \in \arg\min\{p_r(M') : M' \textit{ is a matching in } I\}$$

if and only if

$$M \in \arg\max\{p'_r(M') : M' \textit{ is a matching in } I\}.$$

Proof. Suppose M is a matching in I such that $p_r(M)$ is minimum taken over all matchings in I and $p'_r(M)$ is not maximum taken over all matchings in I. Then, there is a matching M' in I such that $p'_r(M') \succ p'_r(M)$.

Let $p_r(M) = \langle p_1, p_2, ..., p_k \rangle$ (note that we use indices from 1 to k, despite $p_r(M)$ being a reverse profile) and therefore $p'_r(M) = \langle -p_1, -p_2, ..., -p_k \rangle$. Also let $p'_r(M') = \langle p'_1, p'_2, ..., p'_l \rangle$. Since $p'_r(M') \succ p'_r(M)$, there must exist some i $(1 \leq i \leq \min\{k, l\})$ such that $p'_i > -p_i$ and $p'_j = -p_j$ for $1 \leq j < i$.

Then,

$$\begin{aligned} p_r(M') &= \langle -p'_1, -p'_2, ..., -p'_l \rangle \\ &= \langle p_1, p_2, ..., p_{i-1}, -p'_i, ..., -p'_l \rangle \\ &\prec \langle p_1, p_2, ...p_{i-1}, p_i, ..., p_k \rangle \\ &= p_r(M). \end{aligned} \tag{4.3}$$

Hence, $p_r(M)$ cannot be minimum taken over all matchings in I, a contradiction. Therefore, M is a matching such that $p'_r(M)$ is maximum taken over all matchings in I.

Now conversely, suppose that M is a matching such that $p'_r(M)$ is maximum taken over all matchings in I, but $p_r(M)$ is not minimum taken over all matchings in I. Then there is a matching M' in I such that $p_r(M') \prec p_r(M)$.

Let $p_r(M) = \langle p_1, p_2, ..., p_k \rangle$ and therefore $p'_r(M) = \langle -p_1, -p_2, ..., -p_k \rangle$. Also let $p'_r(M') = \langle p'_1, p'_2, ..., p'_l \rangle$ and so $p_r(M') = \langle -p'_1, -p'_2, ..., -p'_l \rangle$. Since $p_r(M') \prec p_r(M)$, there must exist some i $(1 \leq i \leq \min\{k, l\})$ such that $-p'_i < p_i$ and $-p'_j = p_j$ for $1 \leq j < i$.

Then,

$$\begin{aligned} p'_r(M') &= \langle p'_1, p'_2, ..., p'_l \rangle \\ &= \langle -p_1, -p_2, ..., -p_{i-1}, p'_i, ..., p'_l \rangle \\ &\succ \langle -p_1, -p_2, ... - p_{i-1}, -p_i, ..., -p_k \rangle \\ &= p'_r(M). \end{aligned} \tag{4.4}$$

Hence, $p'_r(M)$ cannot be maximum taken over all matchings in I, a contradiction, meaning that M is a matching such that $p_r(M)$ is minimum taken over all matchings in I. □

We now show that a generous stable matching may be found by eliminating a maximum profile closed subset of the rotation poset as in the rank-maximal case. Let ρ be the rotation that takes us from stable matching M to stable matching M', where M and M' have profiles $p(M) = \langle p_1, p_2, ..., p_k \rangle$ and $p(M') = \langle p'_1, p'_2, ..., p'_k \rangle$ with each profile having length k without loss of generality. Then $p(\rho) = \langle p'_1 - p_1, p'_2 - p_2, ..., p'_k - p_k \rangle$ and so $p(M') = p(M) + p(\rho)$. We also know that $p'_r(M) = \langle -p_k, -p_{k-1}, ..., -p_1 \rangle$ and $p'_r(M') = \langle -p'_k, -p'_{k-1}, ..., -p'_1 \rangle$.

Now, since $p'_r(\rho) = \langle p_k - p'_k, p_{k-1} - p'_{k-1}, ..., p_1 - p'_1 \rangle$, in the generous case we have $p'_r(M') = p'_r(M) + p'_r(\rho)$. We next present Lemma 4.5.2 which is an analogue of Lemma 4.4.1 and shows that a generous stable matching may be found by eliminating a maximum profile closed subset of the rotation poset.

Lemma 4.5.2. *Let I be an instance of* SMI *and let M_0 be the man-optimal stable matching in I. A generous stable matching M may be obtained by eliminating a maximum profile closed subset of the rotation poset S from M_0.*

Proof. Let $R_p(I)$ be the rotation poset of I. There is a 1-1 correspondence between closed subsets of $R_p(I)$ and the stable matchings of I [25]. Let S be a maximum profile closed subset of the rotation poset $R_p(I)$, whose rotation profiles are reversed and negated, and let M be the unique corresponding stable matching. Then by addition over reversed negated profiles described in the text above this lemma, $p'_r(M) = p'_r(M_0) + \sum_{\rho_i \in S} p'_r(\rho_i)$. Suppose M is not generous. Then there is a stable matching M' such that $p'_r(M') \succ p'_r(M)$. M' corresponds to a unique closed subset S' of the rotation poset, such that $p'_r(M') = p'_r(M_0) + \sum_{\rho_i \in S'} p'_r(\rho_i)$. But $p'_r(M') \succ p'_r(M)$ and so S cannot be a maximum profile closed subset of $R_p(I)$, a contradiction.

By Proposition 4.5.1, since M is a matching such that $p'_r(M)$ is maximum among all stable matchings, M is also a matching such that $p_r(M)$ is minimum among all stable matchings. Therefore M is a generous stable matching in I. □

Recall that in Definition 4.3.1 we defined the high-weight function w in order to show that a stable matching of maximum weight is a rank-maximal stable matching and then showed that vb-flows and vb-capacities correspond directly with this high-weight setting. In the generous case, since we seek a matching M that maximises $p'_r(M)$, the negation of the reverse profile, the constraints of Definition 4.3.1 still apply. By Proposition 4.5.1 and Lemma 4.5.2, all processes from the previous section to find a rank-maximal stable matching may now be used to find a generous stable matching in $O(nm^2 \log n)$ time.

However, it is also possible to exploit the structure of a generous stable matching to bound some part of the overall time complexity by the generous stable matching degree rather than by the number of men or women n.

Let I be an instance of SMI. First we find a minimum regret stable matching M' of I as described in Section 2.2.2.2 in $O(m)$ time. It must be the case that the degree $d(M)$ of a generous stable matching M is the same as the degree of M'. Therefore, since no man or women can be assigned to a partner of rank higher than $d(M)$, it is possible to simply truncate all preference lists beyond rank $d(M)$, which has a positive effect on the overall time complexity of finding a generous stable matching. Theorem 4.5.3 shows that a generous

stable matching may be found in $O(\min\{m, nd\}^2 d \log n)$ time, where d is the degree of a minimum regret stable matching.

Theorem 4.5.3. *Given an instance I of* SMI *there is an $O(\min\{m, nd\}^2 d \log n)$ algorithm to find a generous stable matching in I using polynomially-bounded weight vectors, where d is the degree of a minimum regret stable matching.*

Proof. Each step required to find a generous stable matching in I is outlined below along with its time complexity.

1. Calculate the degree d of a minimum regret stable matching and truncate preference lists accordingly. A minimum regret stable matching may be found in $O(m)$ time [23]. We now assume all preference lists are truncated below rank d.

2. Calculate the man-optimal and woman-optimal stable matchings. The Extended Gale-Shapley Algorithm takes $O(\min\{m, nd\})$ time since the number of acceptable pairs is now $\min\{m, nd\}$.

3. Find all rotations using the Minimal Differences Algorithm [25] in $O(\min\{m, nd\})$ time, since the number of acceptable pairs is now $\min\{m, nd\}$.

4. Build the rotation digraph and vb-network, using the process described in Section 4.4, but where rotation profiles are reversed and negated for the building of the vb-network. We know that no man-woman pair can appear in more than one rotation. This means that the number of vertices in each of the associated rotation poset, rotation digraph and vb-network is also $O(\min\{m, nd\})$. Identical reasoning (with adapted time complexities to suit the generous case) to that of Gusfield and Irving [25, p. 112] may be used to obtain a bound of $O(\min\{m, nd\})$ on the number of edges. This is because we have a bound of $O(\min\{m, nd\})$ for the creation of both type 1 and type 2 edges of the rotation digraph. Therefore we may build the rotation digraph and vb-network in $O(\min\{m, nd\})$ time.

5. Find a minimum cut of the vb-network using the process described in Section 4.4. With $O(\min\{m, nd\})$ vertices and edges and a maximum length of d for any preference list, the Sleator and Tarjan algorithm [71] has a time complexity of $O(\min\{m, nd\}^2 \log n)$, with an additional factor of $O(d)$ to perform operations over vectors. Hence this step takes a total of $O(\min\{m, nd\}^2 d \log n)$ time.

6. Use this cut to find a maximum profile closed subset S of the rotations in $O(\min\{m, nd\})$ time, since the numbers of vertices and edges are bounded by $O(\min\{m, nd\})$.

7. Eliminate the rotations of S from the man-optimal matching to find the corresponding rank-maximal stable matching.

Therefore the operation that dominates the time complexity is still Step 5 and the overall time complexity to find a generous stable matching for an instance I of SMI is $O(\min\{m, nd\}^2 d \log n)$. □

4.6 Complexity of finding profile-based stable matchings in SR

SR, a generalisation of SMI, was first introduced in Section 2.3. In this section, we look at the complexity of finding rank-maximal and generous stable matchings in SR.

First let RMSR be the problem of finding a rank-maximal stable matching in an instance I of SR, and let GENSR be the problem of finding a generous stable matching in an instance I of SR. We now define their respective decision problems.

Definition 4.6.1. *We define* RMSR-D*, the decision problem of* RMSR*, as follows. An instance* (I, σ) *of* RMSR-D *comprises an instance* I *of* SR *and a profile* σ*. The problem is to decide whether there exists a stable matching* M *in* I *such that* $p(M) \succeq \sigma$*.*

Definition 4.6.2. *We define* GENSR-D*, the decision problem of* GENSR*, as follows. An instance* (I, σ) *of* GENSR-D *comprises an instance* I *of* SR *and a profile* σ*. The problem is to decide whether there exists a stable matching* M *in* I *such that* $p_r(M) \preceq \sigma_r$*, where* σ_r *is the reverse profile of* σ*.*

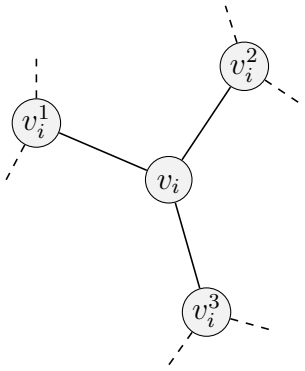
We now show that both RMSR-D and GENSR-D are NP-complete.

Theorem 4.6.3. RMSR-D *is* NP*-complete.*

Proof. We first show that RMSR-D is in NP. Given a matching M it is possible to check that M is stable in $O(m)$ time by iterating through the preference lists of all men and women, comparing the rank of a given agent on a preference list with the rank of the assigned partner in M. It is also possible to calculate profile $p(M)$ and to test whether $p(M) \succeq \sigma$ in $O(n)$ time. Therefore RMSR-D is in NP.

We reduce from VERTEX COVER in cubic graphs [21, 45] (VC-3). An instance (G, K) of VC-3 comprises a graph $G = (V, E)$, with vertex set $V = \{v_1, v_2, ..., v_n\}$ such that each vertex has degree 3, and a positive integer K. The problem is to determine whether there exists a set of vertices $V' \subseteq V$ such that for every edge $e \in E$ at least one endpoint of e is in V', where $|V'| \leq K$.

We now construct an instance (I, σ) of RMSR-D from (G, K).



(a) The neighbourhood of v_i in the graph G of a VC-3 instance.

For each $v_i \in V$:

v_i: w_i, v_i^1, v_i^2, v_i^3, y_i, ...
w_i: x_i, a_1, a_2, a_3, a_4, v_i, ...
x_i: a_1, y_i, w_i, ...
y_i: v_i, x_i, ...

(b) Preference lists of I for a given v_i.

a_j: b_j, ... $\quad (1 \leq j \leq 4)$
b_j: a_j, ... $\quad (1 \leq j \leq 4)$

(c) Additional preference lists for instance I.

Figure 4.5: Creation of an instance I of SR.

Agents of the constructed instance I of SR comprise $A \cup B \cup V \cup W \cup X \cup Y$ where $A = \{a_1, a_2, a_3, a_4\}$, $B = \{b_1, b_2, b_3, b_4\}$, $V = \{v_1, v_2, ..., v_n\}$, $W = \{w_1, w_2, ..., w_n\}$, $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_n\}$. For each vertex $v_i \in V$ with neighbours v_i^1, v_i^2, v_i^3 in G, shown in Figure 4.5a, we create complete preference lists for instance I of SR as in Figure 4.5b, with members of the sets A, B, V, W, X and Y comprising new roommate agents of the constructed SR instance. Note that relative order of v_i^1, v_i^2 and v_i^3 in v_i's preference list is not important. Additional complete preference lists are created as shown in Figure 4.5c. In these figures, the symbol "..." at the end of preference lists indicates that all other agents in I who are not already specified in the preference list are then listed in any order.

Finally, set the profile $\sigma = \langle 2n - K + 8, 2K, n - K, 0, n - K, K \rangle$.

We claim that G has a vertex cover of size $\leq K$ if and only if I has a stable matching M such that $p(M) \succeq \sigma$.

Suppose that G has a vertex cover C such that $|C| = k \leq K$. We create a matching M as follows.

- Add, $\{a_j, b_j\}$ to M for $1 \leq j \leq 4$ giving 8 first choices.
- If $v_i \in C$ then add $\{v_i, w_i\}$ and $\{x_i, y_i\}$ to M. This means v_i is assigned their first choice in I, w_i is assigned their sixth choice, and x_i and y_i both have their second choices.

- If $v_i \notin C$ then add $\{v_i, y_i\}$ and $\{w_i, x_i\}$ to M. Then v_i is assigned their fifth choice in I, y_i and w_i are assigned their first choices, and x_i is assigned their third choice.

We now show that M is stable. Agents from A and B are always assigned to each other, and so cannot block M. We now look through the four addition types of pair of M. Pair $\{v_i, w_i\}$ cannot block M since w_i cannot prefer v_i to their assigned partner. Pair $\{x_i, y_i\}$ cannot block M since y_i cannot prefer x_i to their assigned partner. Pair $\{v_i, y_i\}$ cannot block M since v_i cannot prefer y_i to their assigned partner. Finally, pair $\{w_i, x_i\}$ cannot block M since x_i cannot prefer w_i to their assigned partner. The only remaining possibility is that $\{v_i, v_j\}$ blocks M. Assume for contradiction that this is the case. Then, it must be that $\{v_i, y_i\}$ and $\{v_j, y_j\}$ are in M. But, by construction this implies that neither v_i nor v_j are in C, meaning C is not a valid vertex cover, a contradiction since $\{v_i, v_j\} \in E$.

We are now able to calculate the profile $p(M) = \langle 2n - k + 8, 2k, n - k, 0, n - k, k\rangle$ by totalling the choices for each rank described in the bullet points above. Since $k \leq K$, we have $p(M) \succeq \sigma$.

Conversely, suppose M is a stable matching in I such that $p(M) \succeq \sigma$.

We first show that each agent may only be assigned to a subset of agents shown on their preference lists above.

- For $1 \leq j \leq 4$, since a_j ranks b_j as first choice and vice versa, $\{a_j, b_j\} \in M$;
- v_i can never be assigned lower on their preference list than y_i, or else $\{v_i, y_i\}$ would block M;
- w_i can never be assigned lower on their preference list than v_i, or else $\{v_i, w_i\}$ would block M;
- x_i can never be assigned lower on their preference list than w_i, or else $\{w_i, x_i\}$ would block M;
- y_i cannot be assigned lower on their preference list than x_i, or else $\{x_i, y_i\}$ would block M, noting that $\{x_i, a_1\} \notin M$ by the first point above;
- Finally, suppose $\{v_i, v_j\} \in M$. Then one of w_i, x_i, y_i is unassigned in M, a contradiction to the above. So, either $\{v_i, w_i\} \in M$ or $\{v_i, y_i\} \in M$.

If $\{v_i, w_i\} \in M$ then we add v_i to C. It remains to prove that C is a vertex cover of G. Suppose for a contradiction that it is not. Then there exists an edge $\{v_i, v_j\} \in E$ such that $v_i \notin C$ and $v_j \notin C$. By construction this means that $\{v_i, y_i\} \in M$ and $\{v_j, y_j\} \in M$. But then $\{v_i, v_j\}$ blocks M, a contradiction.

Finally, suppose for contradiction that $|C| = k > K$. Note that if $v_i \in C$ then $\{v_i, w_i\} \in M$, and in turn $\{x_i, y_i\} \in M$ and also if $v_i \notin C$ then $\{v_i, y_i\} \in M$, and in turn $\{w_i, x_i\} \in M$. Therefore, using similar logic to above we can calculate the profile of M as $p(M) = \langle 2n - k + 8, 2k, n - k, 0, n - k, k\rangle$. Since $k > K$ we have $p(M) \prec \sigma$, a contradiction.

We have shown that G has a vertex cover of size $\leq K$ if and only if I has a stable matching M such that $p(M) \succeq \sigma$. Since the reduction described above can be completed in polynomial time, RMSR-D is NP-hard. Finally, as RMSR-D is in NP, RMSR-D is also NP-complete. □

Corollary 4.6.4. GENSR-D *is* NP*-complete.*

Proof. We use the same reduction and a similar argument as Theorem 4.6.3 to show that G has a vertex cover of size $\leq K$ if and only if I admits a stable matching M such that $p_r(M) \preceq \sigma_r$. □

We are able to further extend these results looking at a two restricted decision problems.

Definition 4.6.5. *We define* RMSR-FIRST-D *as follows. An instance of* RMSR-FIRST-D*,* (I, K)*, comprises an instance* I *of* SR *and an integer* K*. The problem is to decide whether there exists a stable matching* M *in* I *such that* $p_1 \geq K$*, where* $p(M) = \langle p_1, p_2, ..., p_n\rangle$*.*

Definition 4.6.6. *We define* GENSR-FINAL-D *as follows. An instance of* GENSR-FINAL-D*,* (I, K)*, comprises an instance* I *of* SR *and an integer* K*. The problem is to decide whether there exists a minimum regret stable matching* M *in* I *such that* $p_i \leq K$*, where* $i = d(M)$ *and* $p(M) = \langle p_1, p_2, ..., p_n\rangle$*.*

Corollary 4.6.7. RMSR-FIRST-D *is* NP*-complete.*

Proof. Using identical constructions and logic as in Theorem 4.6.3 above, we can see that by considering only the first element of the profiles we are able to prove RMSR-FIRST-D is NP-complete. □

Corollary 4.6.8. GENSR-FINAL-D *is* NP*-complete.*

Proof. We first show that any stable matching M constructed as described in Theorem 4.6.3 will have degree 6. Suppose for contradiction that there exists a stable matching M with $d(M) \leq 5$. Then $\{w_i, x_i\} \in M$ for all i $(1 \leq i \leq n)$ which implies that $\{v_i, y_i\} \in M$ for all i $(1 \leq i \leq n)$. If we pick any edge $\{v_i, v_j\} \in E$ then $\{v_i, v_j\}$ blocks M, a contradiction.

Now, using a similar proof to Theorem 4.6.3, we note that any stable matching M constructed from a vertex cover C of size $\leq K$ satisfies $d(M) = 6$ and $p_6 \leq K$, where $p(M) = \langle p_1, p_2, ..., p_n\rangle$. Conversely, given a minimum regret stable matching M such that $p_i \leq K$ where $i = d(M)$, it follows that $i = 6$. We then proceed as before, obtaining a vertex cover C of size $\leq K$. □

4.7 Experiments and evaluations

4.7.1 Methodology

For our experiments we used randomly-generated data to compare rank-maximal, generous and median[3] stable matchings over a range of measures (cost, sex-equal score, degree, number of agents obtaining their first choice and number of agents who obtain a partner in the lower $a\%$ of their preference list). This final measure (an example of which was given in Section 4.1.3) may be more formally defined as the number of agents who obtain a partner between their bth choice and nth choice inclusive, where $b = (100 - a)\frac{n}{100} + 1$. We also investigated the effect of varying instance size (in terms of the number of men or women) on these properties. Our experiments explored 19 separate instance types with the number of men (and women) taking the values of $\{10, 20, ..., 100, 200, ..., 1000\}$ and with 1000 instances tested in each case. As in the experimental work of Chapter 3, all instances tested were complete with uniform distributions on preference lists, as this in general produces a larger number of stable matchings than using incomplete lists or linear distributions.

Also as in Chapter 3, the set of all stable matchings of an instance I of SMI were found using Gusfield's $O(m + n|\mathcal{M}_S|)$ time Enumeration Algorithm [23]. The Enumeration Algorithm comprises two runs of the extended Gale-Shapley Algorithm [19] (finding both the man-optimal and woman-optimal stable matchings), one run of the Minimal Differences Algorithm [25] (finding all rotations of an instance), creation of the rotation digraph, and finally enumeration of all stable matchings using this digraph [23]. Using the list of enumerated stable matchings we were then able to compute all the types of optimal stable matchings described above. A timeout of 1 hour was used for this algorithm. Note that since the time complexities of the algorithms described in this chapter do not beat the best known for these problems, we did not implement our algorithms to test their performance. Experiments were carried out on the machine described in Chapter 1, running Ubuntu version 17.10. Instance generation and statistics collection programs were written in Python and run on Python version $2.7.14$. The plot and table generation program was written in Python and run using Python version 3.6.1. All other code was written in Java and compiled using Java version $1.8.0$. All Java code was run on a single thread, with GNU Parallel [72] used to run multiple instances in parallel. Java garbage collection was run in serial and a maximum heap size of 1GB was distributed to each thread. Code and data repositories for these experiments can be found at `https://doi.org/10.5281/zenodo.2545798` and `https://doi.org/10.5281/zenodo.2542703` respectively.

[3]Recall from Section 4.1.3 that although the median criterion is not profile-based, we are interested in determining whether the median stable matching more closely approximates a rank-maximal or a generous stable matching, in practice.

In addition to the above experiments, we also compared the minimum amount of memory required to store edge capacities of the network (based on exponential weights) and associated vb-network (based on vector-based weights) for each instance. The instances described above were used along with five additional instances each for $n \in \{2000, 3000, 4000, 5000\}$. Unlike the instances described above for $n \leq 1000$, space requirement calculations for these larger instances were carried out on a machine running Ubuntu version 14.04 with 4 cores, 16GB RAM and Intel®, Core™ i7-4790 processors, and compiled using Java version 1.7.0. This machine, compared to the one described in Chapter 1, was able to calculate solutions to individual instances more quickly, although it could run fewer threads in parallel. In these experiments, we were solving a small number of more complex instances and were not interested in the time taken, but rather in the memory requirements, therefore this change in machine did not impact our results. A larger timeout of 24 hours was used for each of two runs of the extended Gale-Shapley Algorithm and one run of the Minimal Differences Algorithm. Stable matchings were not enumerated for these instances. All other configurations were the same as before.

A description of how space requirements were calculated now follows. In these calculations we did not assume any particular implementation, but more generally estimated the minimum number of bits required theoretically to store the capacities in each case. As mentioned previously, the Minimal Differences Algorithm was used to find all rotations of an instance, and only instances that did not timeout and had at least one rotation were used in these calculations. For each rotation, a rotation profile was easily computed, and vector-based weight and exponential weight edge capacities were then calculated directly from these rotation profiles.

Let R be the set of rotations in an instance of SMI and let ρ be a rotation with profile $p(\rho) = \{p_1, p_2, ..., p_n\}$. The *degree* of $p(\rho)$ may be described as the maximum index i for which there exists some p_i such that $p_i \neq 0$. Let d_t denote the maximum degree over all rotations in R. An exponential weight w_e was calculated from $p(\rho)$ according to Irving et al.'s [32] original formula of a weight of n^{n-i} for each agent assigned to their ith choice. In reality, we reduced this to $d_t^{d_t-i}$ which allowed a smaller number to be stored. Then, the number of bits required to store w_e was calculated as $\lceil \log_2 w_e \rceil$ with an additional standard 32-bit word used to store the length of this bit representation[4,5]. A vector-based weight w_v was calculated from

[4]For the case where $w_e = 0$, we calculated the number of bits required as 1 with an additional standard 32-bit word. For both the exponential weight and vector-based weight cases, if $d_t = 0$, we calculated the number of bits required as a standard 32-bit word.

[5]We note that since $\lceil \log_2 w_e \rceil$ is in general far greater than 32 (for large n), we chose to use a standard 32-bit word to store the number of bits for each exponential number as opposed to assuming all exponential numbers are an equal size. As an example, consider an instance of size $n = 100$, where there exists one rotation with profile $\langle 1, 0, ..., -1 \rangle$ of length n (requiring $\lceil \log_2(100^{99} - 1) \rceil = 658$ bits to store the associated exponential number) and several rotations with profile $\langle 0, 0, ..., 0 \rangle$ (requiring 1 bit to store the associated exponential number). Were we to require all exponential numbers to be stored using the same number of bits, far more space

$p(\rho)$ using lossless vector compression, as described in Section 4.1.2. Let z be the number of non-zero elements of $p(\rho)$. Then, the number of bits required to hold indices of non-zero elements is given by $z\lceil\log_2 n\rceil$ (since the length of the profile is bounded by n), and the number of bits required to hold values of non-zero elements is given by $z(\lceil\log_2 2n\rceil + 1)$ (since each element is bounded below by $-2n$ and above by $2n$). The addition of a 32-bit word then allowed the number z to be stored. Finally, two additional 32-bit words are required over the whole vb-network (for storing the numbers n and $2n$).

Correctness testing was conducted in the following way. All stable matchings produced by all instances of size up to $n = 1000$ were checked for *(1) capacity*: each man (woman) may only be assigned to one woman (man) respectively; and *(2) stability*: no blocking pair exists. Additional correctness testing was also conducted for all instances of size $n = 10, ..., 60$. For these instances, in addition to the above testing, a process took place to determine whether the number of stable matchings found by the Enumeration Algorithm matched the number found by an IP model. This was developed in Python version 2.7.14 with the IP modelling framework PuLP (version 1.6.9) [59] using the CPLEX solver [28], version 12.8.0. Each instance was run on a single thread with a time limit of 10 hours (all runs completed within this time), using the same machine that was used for instances of size $n \leq 1000$. All correctness tests passed successfully.

4.7.2 Experimental results summary

Table 4.1 shows the 19 instance types of size up to $n = 1000$. As in Chapter 3, we label instance types according to n, e.g., S100 is the instance type containing instances where $n = 100$. In Columns 3 and 4 of this table we show the mean number of rotations $|\mathcal{R}|_{av}$ and the mean number of stable matchings $|\mathcal{M}_S|_{av}$ per instance type respectively. Column 5 displays the number of instances that did not complete within the 1 hour timeout and the mean time taken to run the Enumeration Algorithm over a completed instance is shown in Column 6. Figure 4.7 shows the mean number of stable matchings as n increases.

Figures 4.8 and 4.9 show the mean number of first choices and mean degree respectively, for rank-maximal, median and generous stable matchings, as n increases. Figures 4.10 and 4.11 show the mean cost and sex-equal score respectively, of the above types of stable matchings with the addition of their respective mean optimal values. Note that for any given instance, the cost and sex-equal scores of all rank-maximal stable matchings are equal. This is also the case for generous stable matchings. Our definition of the median stable matching (given in Section 2.2.2.2) ensures that the median stable matching is unique. Thus, in contrast to the case for Chapter 3, we are not required to find an optimal matching with best cost or sex-equal score. Data for these plots may be seen as Tables B.1, B.2, B.3 and B.4 in Appendix B,

would be used than necessary.

where Table B.1 shows statistics for cost and sex-equal score, and Tables B.2, B.3 and B.4 display statistics for rank-maximal, generous and median stable matchings. Additionally, these latter tables show the minimum, maximum and mean number of assignments l_a in the last $a\%$ of all preference lists.

Finally, Figure 4.12 (with associated Table B.5 in Appendix B) shows a plot comparing the mean number of bits required to store edge capacities of a network and vb-network using exponential weights and vector-based weights respectively. In this plot, circles represent the mean number of bits required for different values of n. The exact space requirements were calculated according to the process described in Section 4.7.1. Solid circles represent data points $n \in \{100, 200, ..., 1000\}$ and these were used to calculate the best fit curves shown when assuming a second order polynomial model. 90% confidence intervals using the 5th and 95th percentile measurements for each representation are also displayed. Above $n = 1000$ we extrapolate up until $n = 100,000$, showing the expected trend with an increasing n. Data points for instances of size $n \in \{10, 20, ..., 90, 2000, 3000, 4000, 5000\}$ are represented as unfilled circles. These data were not used to calculate the best fit curves, but are added to the figure to help determine the validity of our extrapolation mentioned above.

The main findings of these experiments are:

- *Number of stable matchings*: From Figure 4.7 we can see that the mean number of stable matchings increases with instance size. Lennon and Pittel [44] showed that the number of expected stable matchings in an instance of size n tends to the order of $n \log n$. Our experiments confirm this result and show a reasonably linear correlation between $n \log n$ and the mean number of stable matchings for instances with $n \geq 100$.

- *Number of first choices*: As expected, rank-maximal stable matchings obtain the largest number of first choices by some margin, when compared to generous and median stable matchings. When looking at the mean number of first choices, this margin appears to increase from almost 1:1 for instance type S10 (6.9 for rank-maximal compared to 6.0 and 6.1 for generous and median respectively) to approximately 3:1 for instance type S1000 (158.4 for rank-maximal compared to 63.5 and 71.5 for generous and median respectively). Generous and median stable matchings are far more aligned, however generous is increasingly outperformed by median on the mean number of first choices with ratios starting at around 1:1 for S10, gradually increasing to 1.1:1 for S1000. This is summarised in the plot shown in Figure 4.8.

- *Number of last $a\%$ choices*: For rank-maximal stable matchings, the mean number of assignments in the final 10% of preference lists was low, increasing from 0.4 for instance type S10 to 1.4 for instance type S1000. Note that this increase is far lower than the rate of instance size increase. The mean number of generous stable matching

choices in the final 50% of preference lists decreased from 2.4 to 0.0 over all instance types. As the generous criteria minimises final choices, this is likely due to the number of stable matchings increasing with larger instance size. Finally, it is interesting to note that the mean number of median stable matching choices in the final 20% of preference lists decreases from 0.5 to 0.0 despite the number of lower ranked choices not being directly minimised. Figure 4.9 shows how the mean degree changes with respect to n for rank-maximal, median and generous stable matchings. We can see that on average the rank-maximal criteria performs badly, putting men or women very close to the end of their preference list. As above the generous criteria outperforms either of the other optimisations, with median somewhere in between.

- *Cost and sex-equal score*: The range between minimum and maximum optimal costs and sex-equal scores over all instance types (Table B.1) is small when compared with results found for these measures in the rank-maximal, generous and median stable matching experiments. In Figure 4.10, we can see that a generous stable matching is a close approximation of an egalitarian stable matching in practice. This is followed by the median and then the rank-maximal solution concepts. A similar, though less pronounced, result holds for the sex-equal stable matching, as can be seen in Figure 4.11.

- *Median stable matchings:* We can see from Figures 4.8, 4.9, 4.10 and 4.11 that over all measures, a median stable matching, on average, more closely approximates a generous stable matching. One possible reason for this is as follows. A rank-maximal stable matching may be seen as prioritising some agents obtaining high choice partners (out of their set of possible partners in any stable matching), possibly at the expense of some agents obtaining low choice partners (out of their set of possible partners in any stable matching). A generous stable matching on the other hand, ensures no-one is assigned to a partner of worse rank than the degree of a minimum-regret stable matching, and so it is less likely that agents achieve low choice partners and, consequently (by similar reasoning to the rank-maximal case), less likely that agents achieve high choice partners as well. Thus, a median stable matching, which assigns each agent with their middle-choice partner (out of the set of all the possible partners in any stable matching) is more likely to approximate a generous stable matching.

- *Space requirement:* From Figure 4.12, we can see clearly that the exponential weights of the network require more space on average than the vector-based weights of the associated vb-network, and that this difference increases as n grows large. Note that the additional small number of data points (not used to calculate the curves) at $n \in \{2000, 3000, 4000, 5000\}$ fit this model well. At $n = 1000$, the vector-based weights require approximately 10 times less space than the exponential weights. Fi-

Men's preferences:				Women's preferences:			
m_1:	w_1	...	w_2	w_1:	m_2	m_1	...
m_2:	w_2	...	w_1	w_2:	m_1	m_2	...
...				...			
m_{2i-1}:	w_{2i-1}	...	w_{2i}	w_{2i-1}:	m_{2i}	m_{2i-1}	...
m_{2i}:	w_{2i}	...	w_{2i-1}	w_{2i}:	m_{2i-1}	m_{2i}	...
...				...			
m_{n-1}:	w_{n-1}	...	w_n	w_{n-1}:	m_n	m_{n-1}	...
m_n:	w_n	...	w_{n-1}	w_n:	m_{n-1}	m_n	...

Figure 4.6: SM instance I_1, with i satisfying $1 \leq i \leq n/2$.

nally, we can see that at $n = 100,000$, we expect the vector-based weights to be around 100 times less costly in terms of space than the exponential weights, with the space requirement for the exponential weights nearing 1GB.

Finally, we show that for a specific family of instances, the result above is even more pronounced. Consider the family of SM instances represented by instance I_1 in Figure 4.6, where n is even. In I_1, the "..." symbol in each man's preference list indicates that all other women, not already specified, are listed in any order (after his first choice and before his last choice). Similarly in each woman's preference list, all other men, not already specified, are listed in any order after her second choice. For $n = 100,000$, I_1 requires over 10GB to store exponential weights of the network, and only 0.64MB to store the equivalent vector-based weights of the vb-network. This shows that in certain circumstances, the vector-based weights can be over $100,000$ times less costly in terms of space than the exponential weights.

Case	n	$\lvert\mathcal{R}\rvert_{av}$	$\lvert\mathcal{M}\rvert_{av}$	Timeout	Time (ms)
S10	10	1.8	3.0	0	50.3
S20	20	4.2	6.5	0	60.2
S30	30	6.5	10.9	0	75.2
S40	40	8.9	15.7	0	93.2
S50	50	11.2	20.9	0	113.8
S60	60	13.4	27.2	0	133.9
S70	70	15.9	34.0	0	163.7
S80	80	18.2	40.6	0	205.5
S90	90	20.0	46.4	0	235.1
S100	100	22.4	54.2	0	278.0
S200	200	41.9	138.8	0	1084.5
S300	300	59.2	231.0	0	2886.1
S400	400	76.2	337.6	0	7972.7
S500	500	90.7	442.0	0	15934.8
S600	600	105.8	566.1	0	32925.3
S700	700	119.1	675.5	0	50802.4
S800	800	131.4	804.0	1	87169.2
S900	900	144.9	937.6	0	128878.0
S1000	1000	157.6	1115.2	1	196029.9

Table 4.1: General instance information and algorithm timeout results.

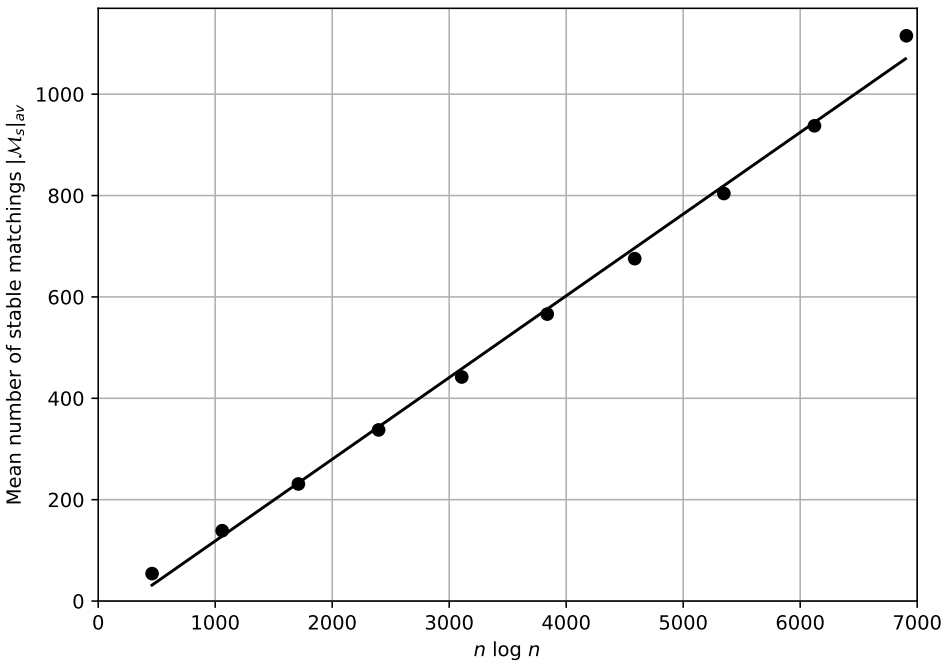


Figure 4.7: Plot of the mean number of stable matchings $|\mathcal{M}_S|_{av}$ with increasing n, where n is the number of men or women. A linear model has been assumed for the best-fit line.

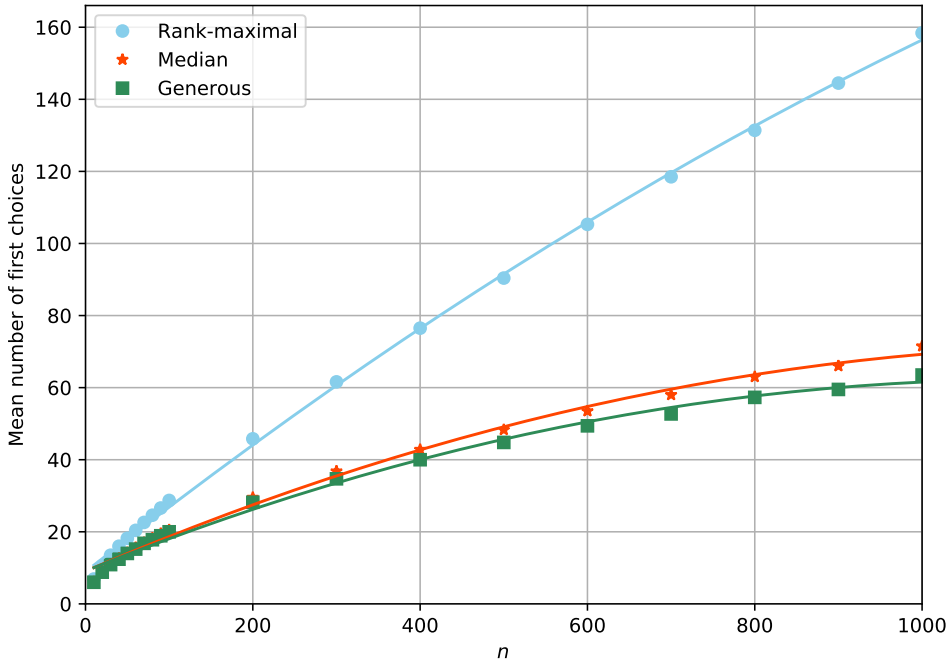


Figure 4.8: Plot of the mean number of first choices for rank-maximal, median and generous stable matchings with increasing n, where n is the number of men or women. A second order polynomial model has been assumed for all best-fit lines.

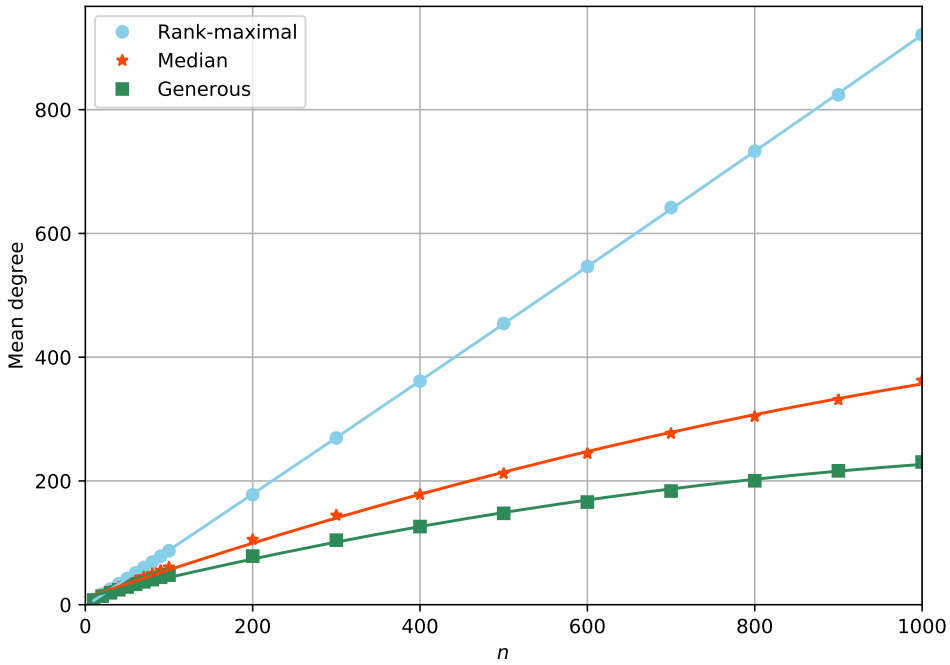


Figure 4.9: Plot of the mean degree for rank-maximal, median and generous stable matchings with increasing n, where n is the number of men or women and the degree of a matching is the rank of a worst ranking man or woman. A second order polynomial model has been assumed for all best-fit lines.

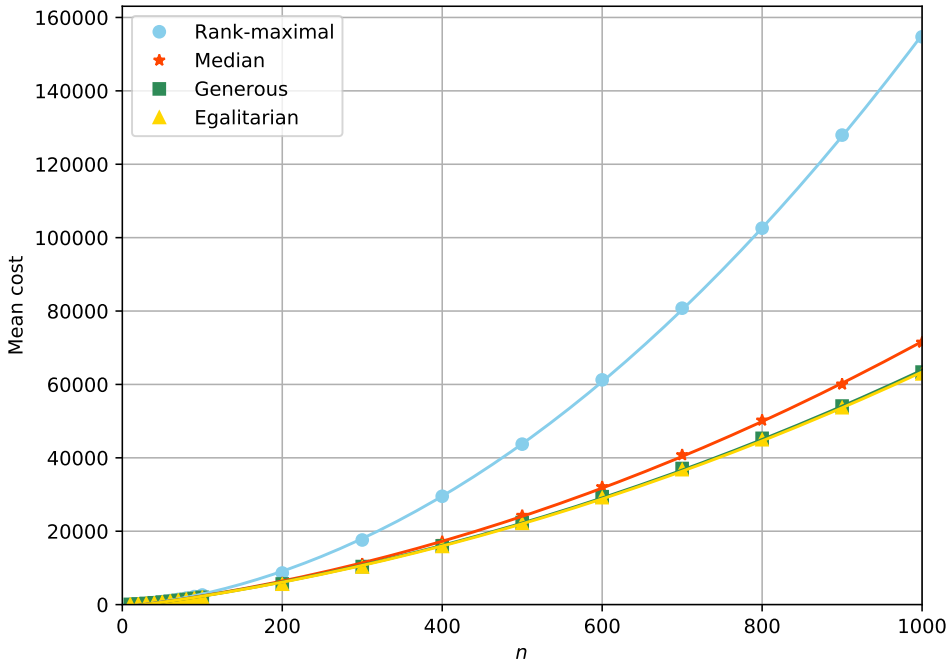


Figure 4.10: Plot of the mean cost for rank-maximal, median, generous and egalitarian stable matchings with increasing n, where n is the number of men or women and the cost of a matching is the sum of ranks of all men and women. A second order polynomial model has been assumed for all best-fit lines.

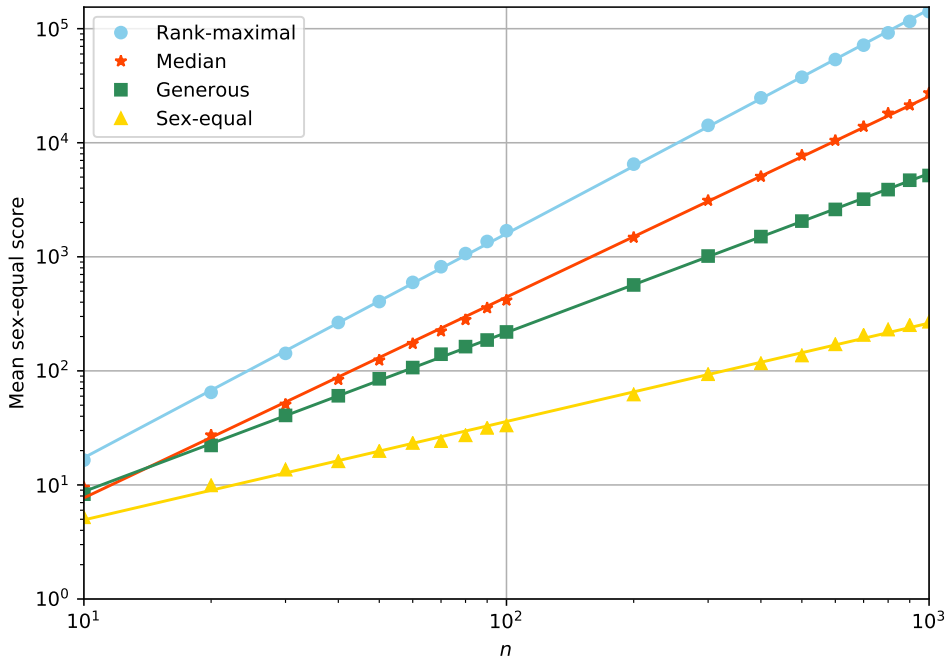


Figure 4.11: A log-log plot of the mean sex-equal score for rank-maximal, median, generous and sex-equal stable matchings with increasing n, where n is the number of men or women and the sex-equal score of a matching is the absolute difference in cost between the set of men and set of women. A first order polynomial model has been assumed for all best-fit lines.

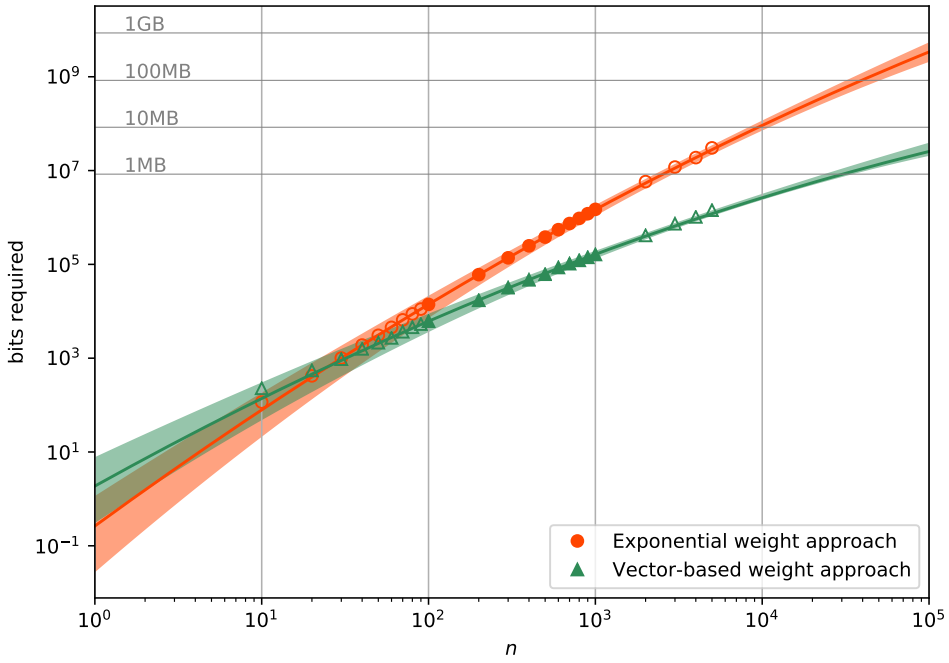


Figure 4.12: A log-log plot of the number of bits required to store a network and vb-network for varying the number of men or women n up to $n = 100,000$, comparing exponential weight and vector-based weight representations. A second order polynomial model has been assumed for all best-fit lines.

4.8 Conclusions and future work

In this chapter we have described a new method for computing rank-maximal and generous stable matchings for an instance of SMI using polynomially-bounded weight vectors that avoids the use of weights that can be exponential in the number of men. By using this approach we were able to avoid high-weight calculation problems such as overflow, inaccuracies and limitations in memory that may occur with some data types. We were also able to demonstrate an approximate factor of 10 improvement when using polynomially-bounded weight vectors with vector compression, as opposed to exponential weights, in terms of the space required to store the network (used to find a maximum flow when computing the optimal matchings above) for instances where $n = 1000$. We also showed that for a specific instance of size $n = 100,000$, the space required to store exponential weights was over 10GB, whereas the vector-based weights were over $100,000$ times less costly, requiring only 0.64MB. This improvement is expected to increase further as n grows large. An additional benefit to our new approach is that as all operations are conducted on rotation profiles (which denote the change in number of men or women with an ith choice partner), it is arguable that our algorithm is more transparent in terms of its execution.

Finally, we also showed that the problem of finding rank-maximal and generous stable matchings in SR is NP-hard. This results still applies even in the restricted cases where

we are either finding a stable matching that maximises the number of first choices or finding a minimum regret stable matching M that minimises the number of dth choices where $d = d(M)$.

In Section 4.1, two potential improvements that could be made to the process of finding a rank-maximal stable matching in an instance of SMI were highlighted. First was the adaptation of Orlin's [64] Max Flow algorithm to work in the vector-based setting. This adaptation would result in a time complexity of $O(nm^2)$ to find a rank-maximal stable matching, improving on the method outlined in this chapter by a factor of $\log n$, however it is not clear that Orlin's algorithm can be adapted to the vb-flow setting. Additionally, Feder [15] used an entirely different technique based on weighted SAT for finding a rank-maximal stable matching in $O(n^{0.5}m^{1.5})$ time. It remains to be seen if this could be adapted to work in the vector-based setting.

Chapter 5

Large stable matchings in SPA-ST

5.1 Introduction

5.1.1 Background

In this chapter we consider stable matchings in the Student-Project Allocation problem, and in particular we present a $\frac{3}{2}$-approximation algorithm for the problem of finding a maximum stable matching in an instance of SPA-ST. An introduction to SPA-ST was given in Section 2.5.2.2. Recall that for a special case of SPA-ST, known as HRT, the problem of finding a maximum stable matching (MAX-HRT) is NP-hard [51]. This result also extends to the SPA-ST case where the problem of finding a maximum stable matching in SPA-ST (MAX-SPA-ST) is also NP-hard. As described in Section 2.4.2 Király [40] developed a $\frac{3}{2}$-approximation algorithm for MAX-HRT. In this chapter, we extend this algorithm to MAX-SPA-ST.

5.1.2 Motivation

In universities all over the world, students need to be assigned to projects as part of their degree programmes. Lecturers typically offer a range of projects, and students may rank a subset of the available projects in preference order. Lecturers may have preferences over students, or over the projects they offer, or they may not have explicit preferences at all. There may also be capacity constraints on the maximum numbers of students that can be allocated to each project and lecturer. We consider the case where lecturers have preferences over students that rank their projects. This problem may be modelled by SPA-S (introduced in Section 2.5.2). In the case of SPA-S, the desired matching must be stable with respect to the given preference lists. It is also reasonable to assume that a student (lecturer) may be indifferent between two or more projects (students) on their preference list. With this

extension, the problem may be modelled by SPA-ST and since stable matchings in SPA-ST may be of different sizes [6], finding a large stable matching is highly desirable.

Finding a large stable allocation of students to projects manually is time-consuming and error-prone. Consequently many universities automate the allocation process using a centralised algorithm. Given the typical sizes of problem instances (e.g., 152 students at the School of Computing Science, University of Glasgow in 2019), the efficiency of the matching algorithm is of paramount importance.

5.1.3 Contribution

In this chapter we describe a linear-time $\frac{3}{2}$-approximation algorithm for MAX-SPA-ST. This algorithm is a non-trivial extension of Király's approximation algorithm for MAX-HRT [40]. In Chapter 6, we describe an IP model to solve MAX-SPA-ST optimally and perform a series of experiments on randomly-generated data to test our approximation algorithm's performance. See Chapter 6 for more information on this work.

A natural "cloning" technique, involving transforming an instance I of SPA-ST into an instance I' of SMTI, and then using Király's $\frac{3}{2}$-approximation algorithm for SMTI [40] in order to obtain a similar approximation in SPA-ST, does not work in general, as we show in Section 5.3. This motivates the need for a bespoke algorithm for the SPA-ST case. Note that we use Király's SMTI algorithm rather than his HRT algorithm as the former was more precisely described in [40].

5.1.4 Structure of the chapter

Section 5.2 gives some preliminary definitions for the chapter. Section 5.3 describes a technique to convert an SPA-ST instance to an SMTI instance, and gives an example where using this technique with Király's $\frac{3}{2}$-approximation algorithm for SMTI does not, in general, allow for the retention of the $\frac{3}{2}$ bound in the original SPA-ST instance. Section 5.4 describes our new $\frac{3}{2}$-approximation algorithm for MAX-SPA-ST, with correctness proofs presented in Section 5.5. Finally, Section 5.6 discusses future work.

5.2 Preliminary definitions

The definitions of a blocking pair and stability in SPA-ST were introduced in Section 2.5.2.2, with the formal definition of a blocking pair given in Section 2.5.2.1. In order to more easily describe certain stages of the approximation algorithm, we provide the following additional definitions. Let (s_i, p_j) be a blocking pair of M. Then we say that (s_i, p_j) is of *type* $(3x)$

if Conditions 1, 2 and $3(x)$ are true in the blocking pair definition, where $x \in \{a, b, c\}$. Blocking pairs of type $(3b)$ are split into two subtypes as follows. $(3bi)$ defines a blocking pair of type $(3b)$ where s_i is already assigned to another project of l_k's. $(3bii)$ defines a blocking pair of type $(3b)$ where this is not the case.

Finally, we let M_{opt} denote a maximum stable matching for a given instance of SPA-ST.

5.3 Cloning from SPA-ST to SMTI

Manlove [47, Theorem 3.11] describes a polynomial transformation to convert a stable matching in an instance of HRT to a stable matching in an instance of SMTI, and vice versa, where the sizes of the matchings is conserved. An obvious question, which we address in this section, relates to whether a similar transformation could be used for SPA-ST, which could allow existing approximation algorithms for HRT and SMTI to be used for the problem of finding a maximum stable matching in SPA-ST.

A natural cloning method to convert instances of SPA-ST to instances of HRT is given as Algorithm 5.1. This algorithm involves converting students into residents and projects into hospitals. Hospitals inherit their capacity from projects. Residents inherit their preference lists naturally from students. Hospitals inherit their preference lists from the lecturer who offers their associated project; a resident entry r_i is ranked only if r_i also ranks this hospital. In order to translate lecturer capacities into the HRT instance, a number of *dummy residents* R_d^k are created for each lecturer l_k. The number of dummy residents created for lecturer l_k, denoted f_k, is equal to the sum of capacities of their offered projects P_k minus the capacity of l_k. We will ensure that all dummy residents are assigned in any stable matching. To this end, each dummy resident has a first position tie of all hospitals associated with projects of l_k, and each hospital h_j in this set has a first position tie of all dummy residents associated with l_k. In this way, as all dummy residents must be assigned in any stable matching by Proposition 5.3.1, lecturer capacities are automatically adhered to.

Proposition 5.3.1. *Let I' be an instance of* HRT *created from an instance I of* SPA-ST *using Algorithm 5.1. All dummy residents must be assigned in any stable matching in I'.*

Proof. In I', for each lecturer l_k, the number of dummy residents created is equal to $f_k = \sum_{p_r \in P_k} c_r - d_k$. Assume for contradiction that one of the dummy residents $r_{d_1}^k$ is unassigned in some stable matching M' of I'.

Let H_k denote the set of hospitals associated with projects of l_k. Since $r_{d_1}^k$ is a dummy resident, it must have all hospitals in H_k tied in first position. Also, each hospital in H_k must rank all f_k dummy residents (associated with l_k) in tied first position. Since $r_{d_1}^k$ is unassigned in M', $r_{d_1}^k$ would prefer to be assigned to any hospital in H_k. Also, since there is at least one

dummy resident unassigned, there must be at least one hospital $h^k_{d_2}$ in H_k that has fewer first-choice assignees than its capacity. Hospital $h^k_{d_2}$ must exist since if it did not, then all dummy residents would be assigned. But then $(r^k_{d_1}, h^k_{d_2})$ would be a blocking pair of M', a contradiction. □

Algorithm 5.1 Clone-SPA-ST, converts an SPA-ST instance into an HRT instance.

Require: An instance I of SPA-ST
Ensure: Return an instance I' of HRT
1: **for each** student s_i in S **do**
2: Create a resident r_i
3: r_i inherits their preference list from s_i's list, ranking hospitals rather than projects
4: **end for**
5: **for each** project p_j in P **do**
6: Create a hospital h_j
7: h_j's capacity is given by $c'_j = c_j$
8: Let l_k be the lecturer offering project p_j
9: h_j inherits their preference list from l_k's list, where a resident entry r_i is retained only if r_i also ranks h_j
10: **end for**
11: **for each** lecturer l_k in L **do**
12: **if** $d_k < \sum_{p_j \in P_k} c_j$ **then**
13: let $f_k = \sum_{p_j \in P_k} c_j - d_k$
14: Create f_k new dummy residents $R^k_d = \{r^k_1, r^k_2, \ldots, r^k_{f_k}\}$
15: Let H_k denote the set of all hospitals in I' associated with the projects of P_k in I
16: The preference list of each dummy resident is given by a first position tie of all hospitals in H_k
17: A first position tie of all residents in R^k_d is added to the start of the preference list of each hospital in H_k
18: **end if**
19: **end for**
20: Let HRT instance I' be formed from all residents (including dummy residents) and hospitals
21: **return** I'

Theorem 5.3.2. *Given an instance I of* SPA-ST *we can construct an instance I' of* HRT *in $O(n_1 + Dn_2 + m)$ time with the property that a stable matching M in I can be converted to a stable matching M' in I' in $O(Dn_2 + m)$ time, where $|M'| = |M| + \sum_{l_k \in L} \sum_{p_r \in P_k} (c_r) - d_k$. Here, n_1 denotes the number of students, n_2 the number of projects, D the total capacities of lecturers and m the total length of student preference lists.*

Proof. Suppose M is a stable matching in I. We construct an instance I' of HRT using Algorithm 5.1. The time complexity of $O(n_1 + Dn_2 + m)$ for the reduction carried out by the algorithm is achieved by noting that I' has a maximum of $n_1 + n_2 + D$ agents and that there are a maximum of $Dn_2 + m$ acceptable resident-hospital pairs.

Initially let $M' = M$ (such that residents take the place of students and hospitals take the place of projects). By Proposition 5.3.1, all dummy residents of I' must be assigned in M' and so we let the set of dummy residents R_d form a resident-complete matching with the set of all hospitals and add these pairs to M'. This is possible because for each lecturer l_k, each dummy resident associated with l_k, denoted r_d^k, finds all hospitals in H_k acceptable, and moreover the total number of remaining positions of the hospitals is equal to $\sum_{p_r \in P_k}(c_r) - |M(l_k)| = \delta_k$ and the number of dummy residents f_k satisfies $f_k = \sum_{p_r \in P_k}(c_r) - d_k \leq \delta_k$, since $|M(l_k)| \leq d_k$.

We claim that M' is stable in I'. Suppose for contradiction that (r_i, h_j) blocks M' in I'.

- All dummy residents must be assigned in M' to their first-choice hospital by above, hence r_i corresponds to a student s_i in I. Resident r_i inherited their preference list from s_i hence we know that s_i finds p_j acceptable. Therefore by the definition given in Section 2.5.2.1, Condition 1 of a blocking pair of M in I is satisfied.
- Resident r_i is either unassigned in M' or prefers h_j to $M'(r_i)$. Student s_i is therefore in an equivalent position and Condition 2 of a blocking pair of M in I is satisfied.
- Hospital h_j is either undersubscribed or prefers r_i to their worst assignee in M'.
  - If h_j is undersubscribed, then p_j must also be undersubscribed. If l_k were full in M then $|M(l_k)| = d_k$ and so the number of remaining positions of hospitals in H_k before dummy residents are added, δ_k, is equal to the number of dummy residents f_k in this scenario. But then all hospitals in H_k (including h_j) would be full in M', contradicting the fact that h_j is undersubscribed. Therefore l_k must be undersubscribed, but then this satisfies Condition $3(a)$ of a blocking pair.
  - If h_j prefers r_i to their worst assignee in M', then l_k must prefer s_i to their worst assignee in $M(p_j)$. This satisfies Condition $3(c)$ of a blocking pair.

Therefore by the definition in Section 2.5.2.1, (s_i, p_j) is a blocking pair of M in I, a contradiction.

Since dummy residents are added in the algorithm's execution it is clear that in general $|M| \neq |M'|$. However, since all dummy residents must be assigned by Proposition 5.3.1, it is trivial to calculate the difference

$$|M'| = |M| + |R_d| = |M| + \sum_{l_k \in L} \sum_{p_r \in P_k} (c_r) - d_k.$$

□

Student preferences:
s_1: $\boldsymbol{p_1}$ p_2
s_2: p_2 $\boldsymbol{p_3}$

Project details:
p_1: lecturer l_1, $c_1 = 1$
p_2: lecturer l_1, $c_2 = 1$
p_3: lecturer l_2, $c_3 = 1$

Lecturer preferences:
l_1: s_2 $\boldsymbol{s_1}$ $d_1 = 1$
l_2: $\boldsymbol{s_2}$ $d_2 = 1$

(a) Example SPA-ST instance I_0. Non-stable matching $M = \{(s_1, p_1), (s_2, p_3)\}$ derived from M' is shown in bold.

Resident preferences:
r_1: $\boldsymbol{h_1}$ h_2
r_2: h_2 $\boldsymbol{h_3}$
r_3: (h_1 $\boldsymbol{h_2}$)

Hospital preferences:
h_1: r_3 $\boldsymbol{r_1}$ $c'_1 = 1$
h_2: $\boldsymbol{r_3}$ r_2 r_1 $c'_2 = 1$
h_3: $\boldsymbol{r_2}$ $c'_3 = 1$

(b) HRT instance I'_0 created from the SPA-ST instance in Figure 5.1a Stable matching $M' = \{(r_1, h_1), (r_2, h_3), (r_3, h_2)\}$ is shown in bold.

Figure 5.1: Conversion of a stable matching M' in HRT into matching M in SPA-ST.

The converse of Theorem 5.3.2 is not true in general, as shown in the example in Figure 5.1. Here, a stable matching M' in an instance I'_0 of HRT does not convert into a stable matching M of the associated instance I_0 of SPA-ST.

A natural question arises as to whether, using a cloning process, we may retain the $\frac{3}{2}$ bound in specific cases where the converted matching M of our original instance of SPA-ST does in fact turn out to be stable. This might occur if, for example, a specific stable matching returned by Király's algorithm as applied to the cloned instance turns out to be stable in the original SPA-ST instance. The cloning process in question would be as follows. For instance I of SPA-ST, we use the cloning process described in Algorithm 5.1 to convert to instance I' of HRT then further convert to instance I'' of SMTI using the process described by Manlove [47, Theorem 3.11]. Next, Király's $\frac{3}{2}$-approximation algorithm is used on I'' generating stable matching M''. Finally, M'' is converted to matching M of I. Then, assuming M is stable, the question is whether it is always the case that M is a $\frac{3}{2}$-approximation to a maximum stable matching of I.

The following example demonstrates Algorithm 5.1 in use and shows that the process described above is *not* sufficient to retain the $\frac{3}{2}$-approximation in an SPA-ST instance I even if the constructed matching M is stable in I.

Algorithm 5.1 is used to convert the SPA-ST instance I_1 in Figure 5.2a to an instance I'_1 of HRT in Figure 5.2b which is then itself converted to the instance I''_1 of SMTI in Figure 5.2c using the process described by Manlove [47, Theorem 3.11]. In this process men correspond to hospitals in I'_1 (projects in I_1) and women correspond to residents in I'_1 (students

Student preferences:
s_1: p_3
s_2: $p_4\ p_1\ p_2$
s_3: p_3
s_4: $(p_2\ p_3)\ p_4\ p_1$

Project details:
p_1: lecturer l_1, $c_1 = 2$
p_2: lecturer l_1, $c_2 = 2$
p_3: lecturer l_2, $c_3 = 2$
p_4: lecturer l_2, $c_4 = 1$

Lecturer preferences:
l_1: $s_2\ s_4$ $d_1 = 2$
l_2: $s_4\ (s_1\ s_2\ s_3)$ $d_2 = 2$

(a) Example SPA-ST instance I_1.

Resident preferences:
r_1: h_3
r_2: $h_4\ h_1\ h_2$
r_3: h_3
r_4: $(h_2\ h_3)\ h_4\ h_1$
r_5: $(h_1\ h_2)$
r_6: $(h_1\ h_2)$
r_7: $(h_3\ h_4)$

Hospital preferences:
h_1: $(r_5\ r_6)\ r_2\ r_4$ $c'_1 = 2$
h_2: $(r_5\ r_6)\ r_2\ r_4$ $c'_2 = 2$
h_3: $r_7\ r_4\ (r_1\ r_3)$ $c'_3 = 2$
h_4: $r_7\ r_4\ r_2$ $c'_4 = 1$

(b) HRT instance I'_1 converted from the SPA-ST instance in Figure 5.2a.

Women's preferences:
w_1: $(m_3\ m_7)$
w_2: $m_4\ (m_1\ m_5)\ (m_2\ m_6)$
w_3: $(m_3\ m_7)$
w_4: $(m_2\ m_3\ m_6\ m_7)\ m_4\ (m_1\ m_5)$
w_5: $(m_1\ m_2\ m_5\ m_6)$
w_6: $(m_1\ m_2\ m_5\ m_6)$
w_7: $(m_3\ m_4\ m_7)$

Men's preferences:
m_1: $(w_5\ w_6)\ w_2\ w_4$
m_2: $(w_5\ w_6)\ w_2\ w_4$
m_3: $w_7\ w_4\ (w_1\ w_3)$
m_4: $w_7\ w_4\ w_2$
m_5: $(w_5\ w_6)\ w_2\ w_4$
m_6: $(w_5\ w_6)\ w_2\ w_4$
m_7: $w_7\ w_4\ (w_1\ w_3)$

(c) SMTI instance I''_1 converted from the HRT instance in Figure 5.2b.

Figure 5.2: Conversion of an SPA-ST instance to an SMTI instance.

in I_1). Executing Király's [40] $\frac{3}{2}$-approximation algorithm on the SMTI instance I''_1 could (depending on order of proposals) yield the matching

$$M'' = \{(m_2, w_5), (m_3, w_4), (m_4, w_2), (m_6, w_6), (m_7, w_7)\}.$$

A trace of how this matching is created is given in Table 5.1. As w_5, w_6 and w_7 were created from dummy residents in Algorithm 5.1, M'' (stable in I''_1) converts into the stable matching $M = \{(s_2, p_4), (s_4, p_3)\}$ of size 2 in I_1. But a maximum stable matching in I_1 is of size 4, given by $M_{opt} = \{(s_1, p_3), (s_2, p_1), (s_3, p_3), (s_4, p_2)\}$. Therefore using the cloning method described above and Király's algorithm does not result in a $\frac{3}{2}$-approximation to the maximum stable matching for instances of SPA-ST, even when the resultant SPA-ST matching is stable. This motivates the development of a $\frac{3}{2}$-approximation algorithm to the maximum stable matching specifically for instances of SPA-ST.

In Appendix C, we introduce instance I_2, which is almost identical to I_1. However, applying the above process to I_2 does yield a stable matching M in I_2 that is a $\frac{3}{2}$-approximation to a maximum stable matching M_{opt}. Comparing these two instances, we give an intuitive idea as to how the addition of dummy residents in the conversion of an SPA-ST instance to an SMTI instance can prevent the retention of the $\frac{3}{2}$ bound. See Section C.1 for more details.

	Action	m_1	m_2	m_3	m_4	m_5	m_6	m_7
1	m_7 applies to w_7, accepted							w_7
2	m_6 applies to w_5, accepted						w_5	w_7
3	m_5 applies to w_6, accepted					w_6	w_5	w_7
4	m_4 applies to w_7, rejected, m_4 removes w_7					w_6	w_5	w_7
5	m_4 applies to w_4, accepted				w_4	w_6	w_5	w_7
6	m_3 applies to w_7, rejected, m_3 removes w_7				w_4	w_6	w_5	w_7
7	m_3 applies to w_4, accepted, m_4 removes w_4			w_4		w_6	w_5	w_7
8	m_4 applies to w_2, accepted			w_4	w_2	w_6	w_5	w_7
9	m_2 applies to w_5, rejected, m_2 removes w_5			w_4	w_2	w_6	w_5	w_7
10	m_2 applies to w_6, rejected, m_2 removes w_6			w_4	w_2	w_6	w_5	w_7
11	m_2 applies to w_2, rejected, m_2 removes w_2			w_4	w_2	w_6	w_5	w_7
12	m_2 applies to w_4, rejected, m_2 removes w_4			w_4	w_2	w_6	w_5	w_7
13	m_2 advantaged			w_4	w_2	w_6	w_5	w_7
14	m_2 applies to w_5, accepted, m_6 removes w_5		w_5	w_4	w_2	w_6		w_7
15	m_6 applies to w_6, rejected, m_6 removes w_6		w_5	w_4	w_2	w_6		w_7
16	m_6 applies to w_2, rejected, m_6 removes w_2		w_5	w_4	w_2	w_6		w_7
17	m_6 applies to w_4, rejected, m_6 removes w_4		w_5	w_4	w_2	w_6		w_7
18	m_6 advantaged		w_5	w_4	w_2	w_6		w_7
19	m_6 applies to w_5, rejected, m_6 removes w_5		w_5	w_4	w_2	w_6		w_7
20	m_6 applies to w_6, accepted, m_5 removes w_6		w_5	w_4	w_2		w_6	w_7
21	m_5 applies to w_5, rejected, m_5 removes w_5		w_5	w_4	w_2		w_6	w_7
22	m_5 applies to w_2, rejected, m_5 removes w_2		w_5	w_4	w_2		w_6	w_7
23	m_5 applies to w_4, rejected, m_5 removes w_4		w_5	w_4	w_2		w_6	w_7
24	m_5 advantaged		w_5	w_4	w_2		w_6	w_7
25	m_5 applies to w_5, rejected, m_5 removes w_5		w_5	w_4	w_2		w_6	w_7
26	m_5 applies to w_6, rejected, m_5 removes w_6		w_5	w_4	w_2		w_6	w_7
27	m_5 applies to w_2, rejected, m_5 removes w_2		w_5	w_4	w_2		w_6	w_7
28	m_5 applies to w_4, rejected, m_5 removes w_4		w_5	w_4	w_2		w_6	w_7
29	m_5 inactive		w_5	w_4	w_2	$-$	w_6	w_7
30	m_1 applies to w_5, rejected, m_1 removes w_5		w_5	w_4	w_2	$-$	w_6	w_7
31	m_1 applies to w_6, rejected, m_1 removes w_6		w_5	w_4	w_2	$-$	w_6	w_7
32	m_1 applies to w_2, rejected, m_1 removes w_2		w_5	w_4	w_2	$-$	w_6	w_7
33	m_1 applies to w_4, rejected, m_1 removes w_4		w_5	w_4	w_2	$-$	w_6	w_7
34	m_1 advantaged		w_5	w_4	w_2	$-$	w_6	w_7
35	m_1 applies to w_5, rejected, m_1 removes w_5		w_5	w_4	w_2	$-$	w_6	w_7
36	m_1 applies to w_6, rejected, m_1 removes w_6		w_5	w_4	w_2	$-$	w_6	w_7
37	m_1 applies to w_2, rejected, m_1 removes w_2		w_5	w_4	w_2	$-$	w_6	w_7
38	m_1 applies to w_4, rejected, m_1 removes w_4		w_5	w_4	w_2	$-$	w_6	w_7
39	m_1 inactive	$-$	w_5	w_4	w_2	$-$	w_6	w_7

Table 5.1: Trace of running Király's SMTI $\frac{3}{2}$-approximation algorithm for instance I_1'' in Figure 5.2c. In this table, the phrase "m_i removes w_j" indicates that man m_i removes woman w_j from their preference list.

5.4 $\frac{3}{2}$-approximation algorithm

5.4.1 Introduction and preliminary definitions

We begin by defining key terminology before describing the approximation algorithm itself in Section 5.4.2, which is a non-trivial extension of Király's HRT algorithm [40].

A student $s_i \in S$ is either in *phase* 1, 2 or 3. In *phase* 1 there are still projects on s_i's list that they have not applied to. In *phase* 2, s_i has iterated once through their list and are doing so again whilst a priority is given to s_i on each lecturer's preference list, compared to other students who tie with s_i. In *phase* 3, s_i is considered unassigned and carries out no more applications. A project p_j is *fully available* if p_j and l_k are both undersubscribed, where lecturer l_k offers p_j. A student s_i *meta-prefers* project p_{j_1} to p_{j_2} if either (i) rank$(s_i, p_{j_1}) <$ rank(s_i, p_{j_2}), or (ii) rank$(s_i, p_{j_1}) =$ rank(s_i, p_{j_2}) and p_{j_1} is fully available, whereas p_{j_2} is not. In phase 1 or 2, s_i may be either *available*, *provisionally assigned* or *held*. Student s_i is *available* if they are not assigned to a project. Student s_i is *provisionally assigned* to project p_j if s_i has been assigned in phase 1 to p_j and there is a project still on s_i's list that they meta-prefer to p_j. Otherwise, s_i is *held*.

If a student s_i is a provisionally assigned to project p_j, then (s_i, p_j) is said to be *precarious*. If s_i is held in their assignment to p_j, then (s_i, p_j) is said to be *non-precarious*. A project p_j is *precarious* if it is assigned a student s_i such that (s_i, p_j) is precarious, otherwise p_j is *non-precarious*. A lecturer is *precarious* if they offer a project p_j that is precarious, otherwise l_k is *non-precarious*. Lecturer l_k *meta-prefers* s_{i_1} to s_{i_2} if either (i) rank$(l_k, s_{i_1}) <$ rank(l_k, s_{i_2}), or (ii) rank$(l_k, s_{i_1}) =$ rank(l_k, s_{i_2}) and s_{i_1} is in phase 2, whereas s_{i_2} is not. The *favourite* projects F_i of a student s_i are defined as the set of projects on s_i's preference list for which there is no other project on s_i's list meta-preferred to any project in F_i. A *worst assignee* of lecturer l_k is defined to be a student in $M(l_k)$ of worst rank, with priority given to phase 1 students over phase 2 students. Similarly, a *worst assignee of lecturer* l_k *in* $M(p_j)$ is defined to be a student in $M(p_j)$ of worst rank, prioritising phase 1 over phase 2 students, where l_k offers p_j.

We remark that some of the above terms such as *favourite* and *precarious* have been defined for the SPA-ST setting by extending the definitions of the corresponding terms as given by Király in the HRT context [40].

5.4.2 Description of the algorithm

Algorithm Max-SPA-ST-Approx (Algorithm 5.2) begins with an empty matching M which will be built up over the course of the algorithm's execution. All students are initially set to

be available and in phase 1. The algorithm proceeds as follows. While there are still available students in phase 1 or 2, choose some such student s_i. Student s_i applies to a favourite project p_j at the head of their list, that is, there is no project on s_i's list that s_i meta-prefers to p_j. Let l_k be the lecturer who offers p_j. We consider the following cases.

- If p_j and l_k are both undersubscribed then (s_i, p_j) is added to M. Clearly if (s_i, p_j) were not added to M, it would potentially be a blocking pair of type $(3a)$.

- If p_j is undersubscribed, l_k is full and l_k is precarious where precarious pair $(s_{i'}, p_{j'}) \in M$ for some project p'_j offered by l_k, then we remove $(s_{i'}, p_{j'})$ from M and add pair (s_i, p_j). This notion of precariousness allows us to find a stable matching of sufficient size even when there are ties in student preference lists (there may also be ties in lecturer preference lists). Allowing a pair $(s_{i'}, p_{j'}) \in M$ to be precarious means that we are noting that $s_{i'}$ has other fully available project options in their preference list at equal rank to $p_{j'}$. Hence, if another student applies to $p_{j'}$ when $p_{j'}$ is full, or to a project offered by l_k where l_k is full, we allow this assignment to happen removing $(s_{i'}, p_{j'})$ from M, since there is a chance that the size of the resultant matching could be increased. Note that since $s_{i'}$ does not remove $p_{j'}$ from their preference list, $s_{i'}$ will get a chance to reapply to $p_{j'}$ if applications to other fully available projects at the same rank are unsuccessful.

- If on the other hand p_j is undersubscribed, l_k is full and l_k meta-prefers s_i to a worst assignee $s_{i'}$, where $(s_{i'}, p_{j'}) \in M$ for some project $p_{j'}$ offered by l_k, then we remove $(s_{i'}, p_{j'})$ from M and add pair (s_i, p_j). It makes intuitive sense that if l_k is full and gets an offer to an undersubscribed project p_j from a student s_i that they meta-prefer to a worst assigned student $s_{i'}$, then l_k would want to remove $s_{i'}$ from $p_{j'}$ and take on s_i for p_j. Student $s_{i'}$ will subsequently remove $p_{j'}$ from their preference list as l_k will not want to assign to them on re-application. This is done via Algorithm 5.3.

- If p_j is full and precarious then pair (s_i, p_j) is added to M while precarious pair $(s_{i'}, p_j)$ is removed. As before, this allows $s_{i'}$ to potentially assign to other fully available projects at the same rank as p_j on their list. Since $s_{i'}$ does not remove p_j from their preference list, $s_{i'}$ will get another chance to assign to p_j if these other applications to fully available projects at the same rank are not successful.

- If p_j is full and l_k meta-prefers s_i to a worst assignee $s_{i'}$ in $M(p_j)$, then pair (s_i, p_j) is added to M while $(s_{i'}, p_j)$ is removed. As this lecturer's project is full (and non-precarious) the only time they will want to add a student s_i to this project (meaning the removal of another student) is if s_i is meta-preferred to a worst student $s_{i'}$ assigned to that project. Similar to before, $s_{i'}$ will not subsequently be able to assign to this project and so removes it from their preference list via Algorithm 5.3.

Algorithm 5.2 Max-SPA-ST-Approx(I), $\frac{3}{2}$-approximation algorithm for SPA-ST.

Require: An instance I of SPA-ST
Ensure: Return a stable matching M where $|M| \geq \frac{2}{3}|M_{opt}|$

$M \leftarrow \emptyset$
All students are initially set to be available and in phase 1
while there exists an available student $s_i \in S$ who is in phase 1 or 2 **do**
    Let l_k be the lecturer who offers p_j
    s_i applies to a favourite project $p_j \in A(s_i)$
    if p_j is fully available **then**
        $M \leftarrow M \cup \{(s_i, p_j)\}$
    else if p_j is undersubscribed, l_k is full **and** (l_k is precarious **or** l_k meta-prefers s_i to a worst assignee) **then** ▷ according to the *worst assignee* definition in Section 5.4.1
        if l_k is precarious **then**
            Let $p_{j'}$ be a project in P_k such that there exists $(s_{i'}, p_{j'}) \in M$ that is precarious
        else ▷ l_k is non-precarious
            Let $s_{i'}$ be a worst assignee of l_k such that l_k meta-prefers s_i to $s_{i'}$ and let $p_{j'} = M(s_{i'})$
            Remove-Pref($s_{i'}, p_{j'}$)
        end if
        $M \leftarrow M \backslash \{(s_{i'}, p_{j'})\}$
        $M \leftarrow M \cup \{(s_i, p_j)\}$
    else if p_j is full **and** (p_j is precarious **or** l_k meta-prefers s_i to a worst assignee in $M(p_j)$) **then**
        if p_j is precarious **then**
            Identify a student $s_{i'} \in M(p_j)$ such that $(s_{i'}, p_j)$ is precarious
        else ▷ p_j is non-precarious
            Let $s_{i'}$ be a worst assignee of l_k in $M(p_j)$ such that l_k meta-prefers s_i to $s_{i'}$
            Remove-Pref($s_{i'}, p_j$)
        end if
        $M \leftarrow M \backslash \{(s_{i'}, p_j)\}$
        $M \leftarrow M \cup \{(s_i, p_j)\}$
    else
        Remove-Pref(s_i, p_j)
    end if
end while
Promote-Students(M)
return M;

Algorithm 5.3 Remove-Pref(s_i, p_j), subroutine for Algorithm 5.2. Removes a project p_j from a student s_i's preference list.

Require: An instance I of SPA-ST and a student s_i and project p_j
Ensure: Return an instance I where p_j is removed from s_i's preference list
1: Remove p_j from s_i's preference list
2: **if** s_i's preference list is empty **then**
3: Reinstate s_i's preference list
4: **if** s_i is in phase 1 **then**
5: Move s_i to phase 2
6: **else if** s_i is in phase 2 **then**
7: Move s_i to phase 3
8: **end if**
9: **end if**
10: **return** I

Algorithm 5.4 Promote-Students(M), subroutine for Algorithm 5.2. Removes all blocking pairs of type $(3bi)$.

Require: SPA-ST Instance I and matching M that does not contain blocking pairs of type $(3a)$, $(3bii)$ or $(3c)$.
Ensure: Return a stable matching M.
1: **while** there are still blocking pairs of type $(3bi)$ **do**
2: Let $(s_i, p_{j'})$ be a blocking pair of type $(3bi)$
3: $M \leftarrow M \backslash \{(s_i, M(s_i))\}$
4: $M \leftarrow M \cup \{(s_i, p_{j'})\}$
5: **end while**
6: **return** M

When removing a project from a student s_i's preference list (the Remove-Pref operation), if s_i has removed all projects from their preference list and is in phase 1 then their preference list is reinstated and they are set to be in phase 2. If on the other hand they were already in phase 2, then they are set to be in phase 3 and are hence inactive. The proof that Algorithm 5.2 produces a stable matching (see Section 5.5) relies only on the fact that a student iterates once through their preference list. Allowing students to iterate through their preference lists a second time when in phase 2 allows us to find a stable matching of sufficient size when there are ties in lecturer preference lists (there may also be ties in student preference lists). This is due to the meta-prefers definition where a lecturer favours one student s_i over another $s_{i'}$ if they are the same rank and s_i is in phase 2 whereas $s_{i'}$ is not. Similar to above, this then allows s_i to steal a position from $s_{i'}$ with the chance that $s_{i'}$ may find another assignment and increase the size of the resultant matching.

After the main while loop has terminated, the final part of the algorithm begins where all blocking pairs of type $(3bi)$ are removed using the Promote-Students operation (Algorithm 5.4).

Student preferences:	Project details:	Lecturer preferences:	
s_1: $(p_2\ p_3)\ p_1$	p_1: lecturer l_1, $c_1 = 1$	l_1: $(s_1\ s_2)\ s_6\ s_5$	$d_1 = 2$
s_2: $p_2\ p_1$	p_2: lecturer l_1, $c_2 = 2$	l_2: $s_1\ s_5$	$d_2 = 2$
s_3: $(p_7\ p_8)$	p_3: lecturer l_2, $c_3 = 2$	l_3: $(s_8\ s_9)\ s_7$	$d_3 = 1$
s_4: p_7	p_4: lecturer l_3, $c_4 = 1$	l_4: $s_7\ (s_3\ s_4)$	$d_4 = 1$
s_5: $p_1\ (p_2\ p_3)$	p_5: lecturer l_3, $c_5 = 1$	l_5: s_3	$d_5 = 1$
s_6: p_1	p_6: lecturer l_4, $c_6 = 1$	l_6: $s_{12}\ (s_{10}\ s_{11})$	$d_6 = 2$
s_7: $(p_4\ p_6)$	p_7: lecturer l_4, $c_7 = 1$	l_7: s_{10}	$d_7 = 2$
s_8: p_5	p_8: lecturer l_5, $c_8 = 1$		
s_9: p_5	p_9: lecturer l_6, $c_9 = 1$		
s_{10}: $p_9\ p_{12}$	p_{10}: lecturer l_6, $c_{10} = 1$		
s_{11}: $p_9\ p_{10}$	p_{11}: lecturer l_6, $c_{11} = 1$		
s_{12}: p_{11}	p_{12}: lecturer l_7, $c_{12} = 1$		

Figure 5.3: SPA-ST instance I_3.

5.4.3 Example execution of the algorithm

A detailed trace of Algorithm Max-SPA-ST-Approx (Algorithm 5.2) over the course of its execution, as applied to the example instance I_3 of SPA-ST shown in Figure 5.3, is given in Table 5.2. In this trace, each application by a student to a project is recorded along with their effects on the instance and matching (such as adding or removing a pair from a matching, removing a preference list element and a student changing phase). The line numbers of Algorithm 5.2 where these effects take place are also recorded. The state of the matching after each application may be seen in the final twelve columns, indicating the assignments of each of the twelve students. An asterisk next to a project p_j for column s_i indicates that (s_i, p_j) is a precarious pair. From this trace, we can see that all parts of the algorithm are executed at some point (although not all line numbers are listed in the trace, this observation may be easily verified by examining the structure of Algorithm 5.2).

For this particular instance, the algorithm outputs stable matching M where

$$M = \{(s_1, p_3), (s_2, p_2), (s_3, p_8), (s_5, p_3), (s_6, p_1),\\ (s_7, p_6), (s_9, p_5), (s_{10}, p_{12}), (s_{11}, p_9), (s_{12}, p_{11})\}.$$

	Action	Line(s)	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}
1	s_1 applies to p_2													
	(s_1, p_2) added to M	7	p_2*											
2	s_2 applies to p_2													
	(s_2, p_2) added to M	7	p_2*	p_2										
3	s_3 applies to p_7													
	(s_3, p_7) added to M	7	p_2*	p_2	p_7*									
4	s_4 applies to p_7													
	(s_3, p_7) removed from M, (s_4, p_7) added to M	24, 25	p_2*	p_2		p_7								
5	s_5 applies to p_1													
	(s_1, p_2) removed from M, (s_5, p_1) added to M	15, 16		p_2		p_7	p_1							
6	s_6 applies to p_1													
	s_5 removes p_1, (s_5, p_1) removed from M, (s_6, p_1) added to M	22, 24, 25		p_2		p_7		p_1						
7	s_7 applies to p_4													
	(s_7, p_4) added to M	7		p_2		p_7		p_1	p_4					
8	s_8 applies to p_5													
	s_7 removes p_4, (s_7, p_4) removed from M, (s_8, p_5) added to M	13, 15, 16		p_2		p_7		p_1		p_5				
9	s_9 applies to p_5													
	s_9 rejected, s_9 removes p_5, s_9 moves to phase 2	27		p_2		p_7		p_1		p_5				
10	s_9 applies to p_5													
	s_8 removes p_5, s_8 moves to phase 2, (s_8, p_5) removed from M, (s_9, p_5) added to M	22, 24, 25		p_2		p_7		p_1			p_5			
11	s_{10} applies to p_9													
	(s_{10}, p_9) added to M	7		p_2		p_7		p_1			p_5	p_9		
12	s_{11} applies to p_9													
	s_{11} rejected, s_{11} removes p_9	27		p_2		p_7		p_1			p_5	p_9		

Table continued on next page.

	Action	Line(s)	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}
13	s_{11} applies to p_{10}													
	(s_{11}, p_{10}) added to M	7		p_2		p_7		p_1			p_5	p_9	p_{10}	
14	s_{12} applies to p_{11}													
	s_{10} removes p_9, (s_{10}, p_9) removed from M, (s_{12}, p_{11}) added to M	13, 15, 16		p_2		p_7		p_1			p_5		p_{10}	p_{11}
15	s_3 applies to p_8													
	(s_3, p_8) added to M	7		p_2	p_8	p_7		p_1			p_5		p_{10}	p_{11}
16	s_1 applies to p_3													
	(s_1, p_3) added to M	7	p_3	p_2	p_8	p_7		p_1			p_5		p_{10}	p_{11}
17	s_5 applies to p_3													
	(s_5, p_3) added to M	7	p_3	p_2	p_8	p_7	p_3	p_1			p_5		p_{10}	p_{11}
18	s_7 applies to p_6													
	s_4 removes p_7, s_4 moves to phase 2, (s_4, p_7) removed from M, (s_7, p_6) added to M	13, 14, 15	p_3	p_2	p_8		p_3	p_1	p_6		p_5		p_{10}	p_{11}
19	s_8 applies to p_5													
	s_8 rejected, s_8 removes p_5, s_8 moves to phase 3	27	p_3	p_2	p_8		p_3	p_1	p_6	-	p_5		p_{10}	p_{11}
20	s_{10} applies to p_{12}													
	(s_{10}, p_{12}) added to M	7	p_3	p_2	p_8		p_3	p_1	p_6	-	p_5	p_{12}	p_{10}	p_{11}
21	s_4 applies to p_7													
	s_4 rejected, s_4 removes p_7, s_4 moves to phase 3	27	p_3	p_2	p_8	-	p_3	p_1	p_6	-	p_5	p_{12}	p_{10}	p_{11}
22	Identifying blocking pairs of type $(3bi)$													
	(s_{11}, p_{10}) removed from M, (s_{11}, p_9) added to M	30	p_3	p_2	p_8	-	p_3	p_1	p_6	-	p_5	p_{12}	p_9	p_{11}

Table 5.2: Detailed trace of running Algorithm Max-SPA-ST-Approx for instance I_3 in Figure 5.3. Projects that form part of a precarious pair with their associated student are marked with an "*". In this table, the phrase "s_i removes p_j" indicates that student s_i removes project p_j from their preference list.

5.5 $\frac{3}{2}$-approximation algorithm correctness proofs

5.5.1 Introduction

In this section we present proofs of correctness for Algorithm Max-SPA-ST-Approx (Algorithm 5.2). These involve showing firstly that the algorithm always produces a stable matching, secondly that the algorithm runs in linear time, and finally that the performance guarantee is $\frac{3}{2}$. The proofs required for this algorithm are naturally longer and more complex than those given by Király [40] for SMTI (formal proofs were not provided for his HRT algorithm), as SPA-ST generalises SMTI to the case that lecturers can offer multiple projects, and projects and lecturers may have capacities greater than 1. These extensions add extra components to the definition of a blocking pair (given in Section 2.5.2.1) which in turn adds complexity to the algorithm and its proof of correctness.

The rest of this section is structured as follows. Section 5.5.2 presents proofs of several preliminary results that are used throughout Section 5.5. Proof that the algorithm produces a stable matching is given in Section 5.5.3. Section 5.5.4 shows that the algorithm runs in linear time with respect to the total length of student preference lists. Finally in Section 5.5.5, we present proof of the $\frac{3}{2}$ performance guarantee.

5.5.2 Proofs of preliminary results

This section comprises several proofs of preliminary results that are used in the following sections. In general, they concern the issue of when in the algorithm's execution a project may be fully available, or when a project or lecturer may be precarious.

First we show in Proposition 5.5.3 (the proof of which uses Propositions 5.5.1 and 5.5.2), that if a project is not fully available at some point in the algorithm's execution, then it cannot subsequently become fully available.

Proposition 5.5.1. *Let T_0 denote the point in Algorithm 5.2's execution at the end of the main while loop. If a project p_j is not fully available at some point before T_0, then it cannot subsequently become fully available before T_0.*

Proof. Assume for contradiction that project p_j is not fully available at some point before T_0, but then subsequently becomes fully available before T_0. At a point where p_j is not fully available, either p_j is full or l_k is full (or both), where l_k offers p_j. If l_k is full, it is clear that l_k must remain so, since students can only be removed from a project of l_k's if they are immediately replaced by another student assigning to a project of l_k. Therefore assume that p_j is full. Then they must somehow become undersubscribed in order to be classified as fully

available. The only way this can happen before T_0 is if lecturer l_k removes a student assigned to p_j in order to replace them with a student becoming assigned to another project of l_k's. But then this deletion can only occur if l_k is full and as above l_k remains full, so p_j cannot become fully available before T_0, a contradiction. □

Proposition 5.5.2. *Suppose a blocking pair $(s_i, p_{j'})$ of type $(3bi)$ exists at the end of the main while loop of Algorithm 5.2, where l_k offers $p_{j'}$, and denote this time by T_0. Then at time T_0, l_k is full.*

Proof. Assume for contradiction that l_k is undersubscribed at T_0. We know that once a lecturer is full they must remain full (since we can only remove a pair associated with a lecturer if we are immediately replacing it with an associated pair). Therefore l_k must have always been undersubscribed. At T_0, $p_{j'}$ must be undersubscribed for $(s_i, p_{j'})$ to be a blocking pair of type $(3bi)$. Therefore at T_0, $p_{j'}$ is fully available and must always have previously been fully available by Proposition 5.5.1. But s_i must have applied to $p_{j'}$ at least once before T_0 and as $p_{j'}$ was fully available this must have been accepted. Then since $(s_i, p_{j'})$ is not in the matching at T_0 it must have been removed before T_0. But in order for this to happen either $p_{j'}$ or l_k would have to be full, contradicting the fact that $p_{j'}$ was fully available before T_0. Hence l_k must be full at T_0. □

Proposition 5.5.3. *During the execution of Algorithm 5.2, if a project p_j is not fully available at some point, then it cannot subsequently become fully available.*

Proof. Let T_0 denote the point in the algorithm's execution at the end of the main while loop. We know from Proposition 5.5.1 that if p_j is not fully available before T_0 then it cannot subsequently become fully available before T_0.

Let lecturer l_k offer project p_j and assume p_j is not fully available at T_0. If l_k contains no blocking pairs of type $(3bi)$ then there can be no changes to allocations of p_j after T_0. Therefore assume l_k contains at least one blocking pair of type $(3bi)$ at T_0. Then by Proposition 5.5.2, l_k is full at T_0. But Algorithm 5.4 does not change the student allocations for any lecturer and hence l_k remains full and p_j cannot subsequently become fully available.

It remains to show that if p_j is fully available at T_0, that it cannot subsequently cease to be fully available and then return to be fully available before the end of the algorithms execution. Since l_k is undersubscribed at T_0, l_k cannot contain any blocking pairs by Proposition 5.5.2. Since Algorithm 5.4 can only affect allocations of projects offered by a full lecturer it is not possible for p_j to change to being not fully available after T_0. □

In Proposition 5.5.4 we show that after the main while loop, no student can be promoted to a fully available project and cannot create a precarious pair. This proposition is used as a stepping stone for other propositions in this section.

Proposition 5.5.4. *Algorithm 5.4 cannot promote a student to a fully available project, nor can it create a precarious pair.*

Proof. Suppose in Algorithm 5.4, s_i is being promoted from project p_j to project $p_{j'}$ both offered by lecturer l_k. We know that in the main while loop s_i must have iterated over their preference list at least to the position of p_j (and perhaps further if s_i has been previously promoted). Therefore, s_i has either been removed from and / or rejected by all projects at the same rank as $p_{j'}$ in their preference list at least once. This can only occur if each of those projects was not fully available at the time and by Proposition 5.5.3 none of these projects could subsequently be fully available. Therefore when s_i is promoted to $p_{j'}$, it can never form a precarious pair. □

Propositions 5.5.5 and 5.5.6 describe conditions in the algorithm's execution under which it is not possible for a particular project or lecturer to be precarious.

Proposition 5.5.5. *Let p_j be a project and let l_k be the lecturer who offers p_j. If l_k is full and non-precarious at some point, then they cannot subsequently become precarious. Similarly, if a project p_j is full and non-precarious at some point, then it cannot subsequently become precarious. Further if l_k is full and p_j is non-precarious then p_j cannot subsequently become precarious.*

Proof. We know that a precarious pair cannot be created in Algorithm 5.4 by Proposition 5.5.4, therefore we focus on the main while loop of Algorithm 5.2. Let lecturer l_k be full and non-precarious at some point during the main while loop Algorithm 5.2's execution and assume that they later becomes precarious. Since l_k is currently non-precarious, the only way they can become so is by a student s_i forming a precarious assignment to a project of l_k's. But recall that a student will first apply to fully available projects at the head of their preference list. Since l_k is full, no project of l_k's can be considered fully available. In order for s_i to apply to a project of l_k's they must first apply to all fully available projects at the head of their list, gain the assignment and then be removed from M. But if a pair (s_i, p_j) is removed from M, p_j cannot be fully available. Ultimately, s_i will have exhausted all previously fully available projects at the head of their list before eventually applying to a project of l_k's. But then at that point s_i cannot create a precarious pair giving a contradiction.

Now, let p_j be full and non-precarious at some point during the main while loop of Algorithm 5.2's execution and assume that it later becomes precarious. If p_j remains full until the time at which it becomes precarious then using similar reasoning to above, any student assigning to p_j cannot be precarious giving a contradiction. If p_j becomes undersubscribed at any point then l_k must be full for the remainder of the algorithm by Proposition 5.5.3. It is possible at this point that l_k is precarious (with precarious pairs that include projects other than p_j)

but since l_k is full, p_j is not fully available. Therefore using similar reasoning to before, any student assigning to p_j cannot create a precarious pair giving a contradiction.

It also follows then that if l_k is full and p_j is non-precarious then p_j cannot subsequently become precarious. □

Proposition 5.5.6. *Suppose a blocking pair $(s_i, p_{j'})$ of type $(3bi)$ exists at the end of the main while loop of Algorithm 5.2, where l_k offers $p_{j'}$, and denote this time by T_0. Then at time T_0, l_k is non-precarious.*

Proof. Let M_0 be the matching being built at T_0 and let $(s_i, p_j) \in M_0$ with l_k offering p_j. Suppose for contradiction that l_k is precarious at T_0.

As $(s_i, p_{j'})$ is a blocking pair of type $(3bi)$, $p_{j'}$ must be undersubscribed at T_0. Also, since $(s_i, p_{j'})$ is a blocking pair we know that s_i prefers $p_{j'}$ to p_j. Therefore s_i must have removed $p_{j'}$ from their preference list. Denote this time as T_1. The removal at T_1 occurred either because $(s_i, p_{j'})$ was removed as a non-precarious pair, or because s_i was rejected on application to $p_{j'}$.

- If $(s_i, p_{j'})$ was removed as a non-precarious pair at T_1 then either l_k was full, non-precarious and s_i was a worst student assigned to l_k, or $p_{j'}$ was full, non-precarious and s_i was a worst student assigned to $p_{j'}$.
- If on the other hand, s_i was rejected on application to $p_{j'}$ at T_1, we know that either l_k was full, non-precarious and l_k did not meta-prefer s_i to a worst student in $M(l_k)$, or $p_{j'}$ was full, non-precarious and l_k did not meta-prefer s_i to a worst student in $M(p_j)$.

Whichever of these possibilities occurred we know that at T_1, either l_k was full and non-precarious or p_j was full and non-precarious.

- Firstly suppose l_k was full and non-precarious at T_1. In this case by Proposition 5.5.5, l_k cannot subsequently become precarious, a contradiction to the fact that l_k is precarious at T_0.
- Therefore, $p_{j'}$ must have been full and non-precarious at T_1. By Proposition 5.5.5, $p_{j'}$ cannot subsequently become precarious. We also know that $p_{j'}$ must go from being full to being undersubscribed since $p_{j'}$ is undersubscribed at T_0. Denote this point in the algorithm's execution as T_2. At T_2, $p_{j'}$ must be non-precarious by above and so a non-precarious pair involved with $p_{j'}$ is removed and replaced with a pair involved with some other project of l_k's. This could only happen if l_k was full and non-precarious and so as before cannot again become precarious, a contradiction.

Therefore, l_k must be non-precarious at T_0. □

In Proposition 5.5.7 we show that after the main while loop, it is not possible for any project to change its fully available status or for any project or lecturer to change their precarious status.

Proposition 5.5.7. *Algorithm 5.4 cannot change the fully available status of any project or the precarious status of any project or lecturer.*

Proof. By Proposition 5.5.4, in Algorithm 5.4 it is not possible to assign a student to a fully available project. Therefore we cannot change a fully available project to be not fully available. Also, any promotions that take place will be to remove a blocking pair of type $(3bi)$, and so by definition the lecturer involved, say l_k, will be full and $|M(l_k)|$ will remain the same. Therefore, all of l_k's projects are not fully available at the start of Algorithm 5.4 and must remain so.

By Proposition 5.5.4, in Algorithm 5.4 it is not possible to create a precarious pair, meaning we cannot change a non-precarious project or lecturer to being a precarious project or lecturer. Finally, by Proposition 5.5.6 no changes can be made to any assignments involving a precarious project or lecturer, hence we cannot change a precarious project or lecturer to be non-precarious. □

Recall, a *worse* student than student s_i, according to lecturer l_k, is any student with a lower rank than s_i on l_k's preference list, or if s_i is in phase 2, any student of the same rank that is in phase 1.

Proposition 5.5.8 examines two specific circumstances. Firstly, it shows that if a lecturer l_k is full, then they cannot subsequently accept a student worse than or equal to a worst student in $M(l_k)$, unless l_k is precarious at the point of application. Secondly, it shows that if a project p_j is full and non-precarious at some point before the end of the main while loop, then p_j cannot subsequently accept a student worse than or equal to a worst student in $M(p_j)$ (according to the lecturer who offers p_j), before the end of the main while loop. In all propositions of this section up to this point, T_0 has been used to denote the point in the algorithm's execution at the end of the main while loop. In Proposition 5.5.8 we change this notation to T_{end}. This is done in order for T_0, $T_{0.5}$, T_1 and T_{end} to denote times that are ordered chronologically.

Proposition 5.5.8. *Let T_{end} denote the point in Algorithm 5.2's execution at the end of the main while loop. Then the following statements are true.*

1. *If a lecturer l_k is full, then a student s_i worse than or equal to l_k's worst assignee(s) cannot subsequently become assigned to a project p_j offered by l_k unless this occurs during the main while loop and l_k is precarious when s_i applies to p_j.*

2. *If a project p_j offered by l_k is full and non-precarious before T_{end}, then a student s_i worse than or equal to l_k's worst ranked assignee(s) in $M(p_j)$ cannot subsequently become assigned to p_j before T_{end}.*

Proof. We deal with each case separately.

1. Let T_0 be a point of the algorithm's execution, mentioned in the statement of the propostion, where l_k is full. Let T_1 be the first point after T_0 where a student s_i worse than or equal to l_k's worst assignee(s) applies to p_j. As this is the first such point, it is not possible for $M(l_k)$ at T_1 to have students of lower rank than existed in $M(l_k)$ at T_0.

   It is clear that in Algorithm 5.4, students assigned to a particular lecturer cannot change, hence we concentrate only on the main while loop of Algorithm 5.2. Since l_k is full at T_0, l_k must be full at T_1 since a pair may only be added and taken away from the same project, or from different projects offered by the same lecturer. Assume l_k is non-precarious at T_1. Either p_j is full or undersubscribed. Suppose p_j is full. Since l_k is non-precarious, p_j is also non-precarious by definition. But then, since l_k does not meta-prefer s_i to a worst assignee in $M(l_k)$, the conditions on Line 17 cannot be satisfied and so s_i cannot become assigned to p_j. Now, assume that p_j is undersubscribed. Since l_k is full, non-precarious, and does not meta-prefer s_i to a worst assignee in $M(l_k)$, the conditions on Line 8 are also not satisfied and so s_i cannot become assigned to p_j. Since it is not possible for a student s_i worse than or equal to l_k's worst assignee(s) at T_0 to become assigned to p_j when this application happens for the first time, it is easy to extend this to any subsequent occurence during the main while loop.

   If on the other hand, l_k is precarious at T_1, since l_k is also full at T_1, it is easy to see that s_i may become assigned to p_j (if either p_j is undersubscribed or p_j is full and precarious).

2. Similar to above, let T_0 be a point of the algorithm's execution, mentioned in the statement of the propostion, where p_j is full and non-precarious. Let T_1 be the first point after T_0 and before T_{end} where a student s_i worse than or equal to l_k's worst assignee(s) in $M(p_j)$ applies to p_j. As this is the first such point, it is not possible for $M(p_j)$ at T_1 to have students of lower rank than existed in $M(p_j)$ at T_0. This result can clearly be extended to apply not only to T_1, but also to any point after T_0 and before T_1.

   Since p_j is full and non-precarious at T_0, it must also be non-precarious at all future points by Proposition 5.5.5. At T_1 either p_j is full or undersubscribed.

- If p_j is full at T_1, then since it is also non-precarious and there cannot be lower ranked students in $M(p_j)$ at T_1 than there were in $M(p_j)$ at T_0, the conditions on Line 17 cannot be satisfied and so s_i cannot become assigned to p_j.
- If p_j is undersubscribed at T_1, then it first became undersubscribed at some point after T_0 and before T_1. Let $T_{0.5}$ be the point just prior to p_j becoming undersubscribed. At $T_{0.5}$, l_k must be full since p_j can only become undersubscribed on Line 15. Let $(s_{i'}, p_j)$ be the pair that is removed from the matching just after $T_{0.5}$. We know p_j is non-precarious after T_0 and so pair $(s_{i'}, p_j)$ is removed as a non-precarious pair, by definition. But, since $(s_{i'}, p_j)$ is non-precarious, it can only have been removed just after $T_{0.5}$ (on Line 15) if l_k is non-precarious and $s_{i'}$ was not only a worst assignee in $M(p_j)$ but also in $M(l_k)$. This means that a worst assignee in $M(l_k)$ at $T_{0.5}$ is of equal rank to a worst assignee in $M(p_j)$ at $T_{0.5}$ and cannot be worse than a worst assignee in $M(p_j)$ at T_0 (by the result given at the start of this case). From this we can deduce that s_i is worse than or equal to l_k's worst assignee(s) in $M(l_k)$ at $T_{0.5}$. Thus, by the proof of Case 1 above, since l_k was full and non-precarious at $T_{0.5}$ with s_i worse than or equal to l_k's worst assignee(s) in $M(l_k)$ at $T_{0.5}$, s_i cannot subsequently become assigned to p_j at T_1. Since it is not possible for a student s_i worse than or equal to l_k's worst assignee(s) in $M(p_j)$ at T_0 to become assigned to p_j when this application happens for the first time before T_{end}, it is easy to extend this to any subsequent occurence at any time before T_{end}.

Therefore each case is proved. □

5.5.3 Stability

In this section we present several results building up to Theorem 5.5.13 which shows that Algorithm Max-SPA-ST-Approx always produces a stable matching.

First, in Lemma 5.5.9, we show that after the main while loop of Algorithm 5.2, only blocking pairs of type $(3bi)$ can exist relative to M.

Lemma 5.5.9. *Let M_1 denote the matching constructed immediately after the main while loop in Algorithm 5.2 has completed and let T_1 denote this point in the algorithm's execution. At T_1, no blocking pair of type $(3a)$, $(3bii)$ or $(3c)$ can exist relative to M_1.*

Proof. Assume for contradiction that (s_{b_1}, p_{b_2}) is a blocking pair of M_1 of type $(3a)$, $(3bii)$ or $(3c)$. Let l_{b_3} be the lecturer who offers p_{b_2}.

It must be the case that in M_1, s_{b_1} is either assigned to a project of lower rank than p_{b_2} or is assigned to no project. Therefore, s_{b_1} must have removed p_{b_2} from their preference list during

the main while loop of Algorithm 5.2. Let M_0 denote the matching constructed immediately before p_{b_2} was first removed from s_{b_1}'s list and let T_0 denote this point in the algorithm's execution. We know that p_{b_2} cannot be fully available at T_0 (otherwise (s_{b_1}, p_{b_2}) would have been added to M_0) and cannot subsequently become fully available by Proposition 5.5.3. There are three places where p_{b_2} could be removed from s_{b_1}'s list, namely Lines 13, 22 and 27. We look at each type of blocking pair in turn.

- $(3a)$ - Assume we have a blocking pair of type $(3a)$ in M_1. Then, p_{b_2} and l_{b_3} are both undersubscribed (and hence p_{b_2} is fully available) in M_1. But this contradicts the above statement that p_{b_2} cannot be fully available after T_0.

- $(3bii)$ & $(3c)$ - Assume we have a blocking pair of type $(3bii)$ or $(3c)$ in M_1. At T_0, p_{b_2} is not fully available and so either p_{b_2} is undersubscribed with l_{b_3} being full, or p_{b_2} is full.

  - If p_{b_2} was undersubscribed and l_{b_3} was full at T_0 then l_{b_3} cannot have been precarious (since s_{b_1} is about to remove p_{b_2} from their list) and by Proposition 5.5.5, cannot subsequently become precarious. By Proposition 5.5.8, l_{b_3} cannot subsequently accept a student ranked lower than a worst student in $M_0(l_{b_3})$. Also either s_{b_1} is a worst assignee in $M_0(l_{b_3})$ (Line 13), or l_{b_3} ranks s_{b_1} at least as badly as a worst student in $M_0(l_{b_3})$ (Line 27). Lecturer l_{b_3} must remain full for the rest of the algorithm since if a student is removed from a project offered by l_{b_3} then they are immediately replaced. Therefore l_{b_3} must be full in M_1, and since l_{b_3} cannot have accepted a worse ranked student than s_{b_1} after T_0, (s_{b_1}, p_{b_2}) cannot be a blocking pair of M_1 of type $(3bii)$ or $(3c)$.
  - Instead assume at T_0 that p_{b_2} is full in M_0. As s_{b_1} is about to remove p_{b_2}, we know p_{b_2} cannot be precarious, and by Proposition 5.5.5, cannot subsequently become precarious. Either s_{b_1} is a worst assignee in $M_0(p_{b_2})$ (Line 22), or l_{b_3} ranks s_{b_1} at least as badly as a worst student in $M_0(p_{b_2})$ (Line 27).

    As p_{b_2} is full and non-precarious, by Proposition 5.5.8, p_j cannot accept a worse student than already exists in $M(p_{b_2})$ at T_0. If p_{b_2} is full at T_1, then clearly (s_{b_1}, p_{b_2}) cannot block M_1. So assume p_{b_2} becomes undersubscribed at some point between T_0 and T_1 for the first time, say at $T_{0.5}$. Since p_{b_2} is non-precarious, all pairs in M associated with p_{b_2} are also non-precarious. Therefore p_{b_2} can only become undersubscribed at $T_{0.5}$ if l_{b_3} is full and there is a student s_i who assigns to another project p_j that l_{b_3} offers, where s_i is meta-preferred to a worst student in $M_0(l_{b_3})$. This worst student must also be a worst student in $M_0(p_{b_2})$ since we are removing from M_0 a pair associated with p_{b_2}. But then s_{b_1} must be ranked at least as badly as a worst student in $M_0(l_{b_3})$. Using similar reasoning to the

previous case, l_{b_3} must be full in M_1, non-precarious, and since l_{b_3} cannot have accepted a worse ranked student than s_{b_1} after $T_{0.5}$, (s_{b_1}, p_{b_2}) cannot be a blocking pair of M_1 of type $(3bii)$ or $(3c)$.

Hence it is not possible for (s_{b_1}, p_{b_2}) to be a blocking pair of M of type $(3a)$, $(3bii)$ or $(3c)$ after the main while loop. □

Let p_j be a project, where l_k offers p_j. Propositions 5.5.10 and 5.5.11 show that from the end of the main while loop to the end of the execution of Algorithm 5.2, if (s_i, p_j) is a blocking pair of type $(3bi)$, then s_i must be one of the worst students in $M(l_k)$.

Proposition 5.5.10. *Let M_1 be the matching constructed immediately at the end of the main while loop of Algorithm 5.2's execution, and let T_1 denote this point in the algorithm's execution. At T_1, for each blocking pair $(s_i, p_{j'})$ of type $(3bi)$, s_i must be one of the worst assignees of $M_1(l_k)$, where l_k offers $p_{j'}$.*

Proof. Since $(s_i, p_{j'})$ is a blocking pair of type $(3bi)$, we know that s_i is assigned to another project, say p_j, of l_k's in M_1, where s_i prefers $p_{j'}$ to p_j.

During the main while loop's execution, student s_i must have removed $p_{j'}$ from their preference list in order to eventually assign to p_j. We note that although s_i may be in either phase, s_i must have removed $p_{j'}$ from their preference list in the same phase that s_i assigned to p_j (also the same phase s_i is in at T_1). For the remainder of this proof, we discuss removal of $p_{j'}$ from s_i's preference list in the context of this phase. Student s_i's removal of $p_{j'}$ could only happen if $p_{j'}$ was non-precarious at this point. By Proposition 5.5.5, since $p_{j'}$ or l_k are full, $p_{j'}$ cannot subsequently become precarious. Let the matching constructed immediately before this removal be denoted by M_0 and let T_0 denote this point in the algorithm's execution. Project $p_{j'}$ was either full or undersubscribed at T_0. We show in the former case that, s_i is one of the worst students in $M_1(l_k)$, and that the latter case leads to a contradiction.

- Suppose $p_{j'}$ was full at T_0. Then as $p_{j'}$ is non-precarious, $p_{j'}$ cannot subsequently be assigned a student worse than the worst assignee in $M_0(p_{j'})$ up until T_1, by Proposition 5.5.8. Since s_i removed $p_{j'}$ from their list while $p_{j'}$ was full we know that either s_i is a worst assignee in $M_0(p_{j'})$ (Line 22), or l_k ranks s_i at least as badly as a worst student in $M_0(p_{j'})$ (Line 27). Between T_0 and T_1, s_i assigns to the project p_j, at a worse rank than $p_{j'}$ in s_i's list, where p_j is also offered by l_k.

  Now, we know that $p_{j'}$ becomes undersubscribed by T_1 and so it must be the case that there is a point $T_{0.5}$ between T_0 and T_1, such that another student $s_{i'}$ assigns to a project (not $p_{j'}$) of l_k's which removes pair $(s_{i''}, p_{j'})$ from the matching constructed just before that removal, denoted by $M_{0.5}$. Let $T_{0.5}$ be the first point at which $p_{j'}$ becomes

undersubscribed after T_0. Lecturer l_k must be full at this point since the addition of $s_{i'}$ removes a student (namely $s_{i''}$) from a different project (namely $p_{j'}$). Also, lecturer l_k cannot have been precarious, otherwise $p_{j'}$ would have been identified as a precarious project at Line 10, but we know $p_{j'}$ cannot have been precarious after T_0. So $s_{i''}$ must have been a worst assignee in $M_{0.5}(l_k)$ and therefore $M_{0.5}(p_{j'})$. From the beginning of this bullet point, we know that the worst student in $M_{0.5}(p_{j'})$ can be no worse than the worst student in $M_0(p_{j'})$. Thus since student s_i cannot have changed phase from point T_0, s_i must be either a worst student in $M_{0.5}(l_k)$, or be as bad as a worst student in $M_{0.5}(l_k)$. By Propositions 5.5.5 and 5.5.8, since l_k is full and non-precarious at $T_{0.5}$, l_k cannot be assigned a worse student than s_i between $T_{0.5}$ to T_1, and so as s_i is assigned to l_k at T_1 and s_i remains in the same phase until T_1, then they must be one of the worst students in $M_1(l_k)$.

- Suppose then that $p_{j'}$ is undersubscribed at T_0. Since a preference element is being removed, l_k must have been full, non-precarious and either s_i is a worst assignee in $M_0(l_k)$ (Line 13), or l_k ranks s_i at least as badly as a worst student in $M_0(l_k)$ (Line 27). But then by Propositions 5.5.5 and 5.5.8, l_k must have remained non-precarious until T_1 and been unable to assign to a student worse than or equal to s_i, including s_i in the same phase, a contradiction.

Therefore, for any blocking pair $(s_i, p_{j'})$ of type $(3bi)$ of M_1, s_i must be one of the worst students in $M_1(l_k)$. □

Proposition 5.5.11. *In Algorithm 5.4, if a blocking pair $(s_{i'}, p_j)$ of type $(3bi)$ is created (in the process of removing a different blocking pair of type $(3bi)$) then $s_{i'}$ must be one of the worst students in $M_2(l_k)$, where l_k is the lecturer who offers p_j and M_2 is the matching constructed immediately after this removal occurs.*

Proof. Let M_0 denote the matching at the end of the main while loop of Algorithm 5.2, and let T_0 denote this point in the algorithm's execution. Assume that during Algorithm 5.4's execution, the first promotion to reveal a blocking pair $(s_{i'}, p_j)$ of type $(3bi)$ occurs such that $s_{i'}$ is not a worst student in $M_1(l_k)$. Let M_1 be the matching constructed just before this promotion occurs. Suppose the promotion involves student s_i moving from a less preferred p_j to a more preferred project $p_{j'}$. It is clear that in the removal of blocking pairs of type $(3bi)$ there is no change in regards to which students are assigned to projects of l_k, therefore the same students are assigned to each lecturer in M_0, M_1 and M_2. By Proposition 5.5.10, s_i is and remains one of the worst assignees of l_k in M_0, M_1 and M_2. Since $s_{i'}$ is not a worst assignee in $M_2(l_k)$, l_k must prefer $s_{i'}$ to s_i. But this would mean $(s_{i'}, p_j)$ was a blocking pair of type $(3c)$ in M_0 if p_j were full or $(3bii)$ in M_0 if p_j were undersubscribed, both a contradiction to Lemma 5.5.9. □

In Proposition 5.5.12 we show that only blocking pairs of type $(3bi)$ may be created in M after the main while loop of Algorithm 5.2.

Proposition 5.5.12. *It is not possible in Algorithm 5.4's execution, for a blocking pair of any type other than* $(3bi)$ *to be created.*

Proof. Let M_0 denote the matching constructed immediately after the main while loop of Algorithm 5.2 terminates and let T_0 denote this point in the algorithm's execution. By Lemma 5.5.9, only blocking pairs of type $(3bi)$ of M_0 may exist at T_0 therefore we restrict our attention to the removal of such pairs. Assume for a contradiction that during Algorithm 5.4's execution, the first promotion to reveal a blocking pair of type not equal to $(3bi)$ occurs. Let M_1 (respectively M_2) be the matching constructed just before (respectively after) this promotion occurs with T_1 (respectively T_2) denoting this point in the algorithm's execution. Suppose that this promotion involves student s_i being promoted from project p_j to project $p_{j'}$ as pair $(s_i, p_{j'})$ is a blocking pair of M_1 of type $(3bi)$. Since $(s_i, p_{j'})$ is a blocking pair of M_1 of type $(3bi)$ we know that p_j and $p_{j'}$ are both offered by the same lecturer, say l_k. Assume that this promotion has now revealed a blocking pair $(s_{i'}, p_j)$ of type $(3a)$, $(3bii)$ or $(3c)$ in M_2. We look at each case in turn.

- $(3a)$ - Since in M_1, $(s_i, p_{j'})$ was a blocking pair of type $(3bi)$ we know that l_k is full at T_1. The promotion involves moving s_i from one project offered by l_k to another, therefore at T_2, l_k must be full and so p_j cannot be involved in a blocking pair of type $(3a)$ in M_2, a contradiction.

- $(3bii)$ - Suppose $(s_{i'}, p_j)$ is a blocking pair of type $(3bii)$ in M_2. Since it is of type $(3bii)$, l_k must prefer $s_{i'}$ to a worst assignee in $M_2(l_k)$ (and consequently $M_1(l_k)$ as students do not change lecturer). If p_j was undersubscribed at T_1 then $(s_{i'}, p_j)$ would have constituted a blocking pair of type $(3bii)$, a contradiction. Therefore p_j must have been full in M_1. We know that $(s_i, p_j) \in M_1$ and that s_i is a worst assignee in $M_1(l_k)$ by Proposition 5.5.10 and 5.5.11, therefore l_k prefers $s_{i'}$ to s_i. It follows that $(s_{i'}, p_j)$ would have constituted a blocking pair in M_1 of type $(3c)$, a contradiction to the fact that no blocking pair of any type other than $(3bi)$ was revealed prior to T_2.

- $(3c)$ - Suppose finally that $(s_{i'}, p_j)$ is a blocking pair of M_2 of type $(3c)$. But blocking pairs of type $(3c)$ require p_j to be full in M_2 which it cannot be since (s_i, p_j) has been removed just before T_2, hence p_j cannot be involved in a blocking pair of type $(3c)$.

Therefore it is not possible for a blocking pair of type $(3a)$, $(3bii)$ or $(3c)$ to be created during the first promotion of a student, and hence any promotion. □

Finally, Theorem 5.5.13 proves that any matching produced by Algorithm Max-SPA-ST-Approx is stable.

Theorem 5.5.13. *Any matching produced by Algorithm 5.2 must be stable.*

Proof. Let M_0 be the matching constructed immediately after the termination of the main while loop of Algorithm 5.2 and let T_0 denote this stage of the algorithm. Recall that by Lemma 5.5.9, only blocking pairs of type $(3bi)$ may exist relative to M_0. Also, by Lemma 5.5.12, no blocking pair of any other type can exist relative to the matching constructed after T_0.

Algorithm 5.4 systematically removes blocking pairs of type $(3bi)$ in a series of student promotions. Each promotion improves the outcome for a student.

Therefore there are no blocking pairs of any type in the finalised matching M_s and so M_s is stable. □

Since this proof relies only on the fact that p_{b_2} is removed from s_{b_1}'s list once for (s_{b_1}, p_{b_2}) not to become a blocking pair, we can infer that if we allowed students to only iterate once through their preference preference list rather than twice, this would still result in a stable matching.

5.5.4 Time complexity and termination

In this section we prove that Algorithm Max-SPA-ST-Approx runs in linear time with respect to the total length of student preference lists.

First, in Proposition 5.5.14, we show that during the main while loop each student may only apply to a project on their preference list a maximum of three times.

Proposition 5.5.14. *The maximum number of times a student s_i can apply to a project p_j on their preference list during the main while loop of Algorithm 5.2 is three.*

Proof. First we note that as soon as s_i removes p_j from their preference list once during Algorithm 5.2's execution, (s_i, p_j) cannot subsequently become a precarious pair by definition (since a precarious pair must be assigned in phase 1).

Focussing on the main while loop, assume for some iteration, that phase 1 student s_i applies to project p_j on their preference list. Either (s_i, p_j) is added to the matching being built M, or p_j is removed from s_i's list. If p_j is removed from s_i's list then s_i may still apply to project p_j in phase 2 but as noted above (s_i, p_j) cannot become a precarious pair.

Assume instead that (s_i, p_j) is added to M. If it remains in M until the algorithm completes then s_i cannot apply to p_j again. So assume that (s_i, p_j) is removed from M at some point

due to another pair being added to M. If (s_i, p_j) was non-precarious at the point it is removed from M then s_i removes p_j from their list and the next time s_i could apply to p_j is when s_i is in phase 2 when as above (s_i, p_j) cannot become a precarious pair.

Assume therefore that (s_i, p_j) was precarious when removed from M. Then s_i does not remove p_j from their list and s_i can again apply to p_j during phase 1. Note that if (s_i, p_j) is re-added to M it must be as a non-precarious pair. This is because, using similar reasoning that was used in Proposition 5.5.5, at the point at which s_i reapplies to p_j they must have exhausted all fully available projects at the head of their list, therefore (s_i, p_j) cannot again become precarious. Therefore, s_i can apply to p_j a maximum of three times during the execution of the while loop: at most twice while s_i is in phase 1 (twice only if (s_i, p_j) is removed as a precarious pair) and at most once in phase 2. □

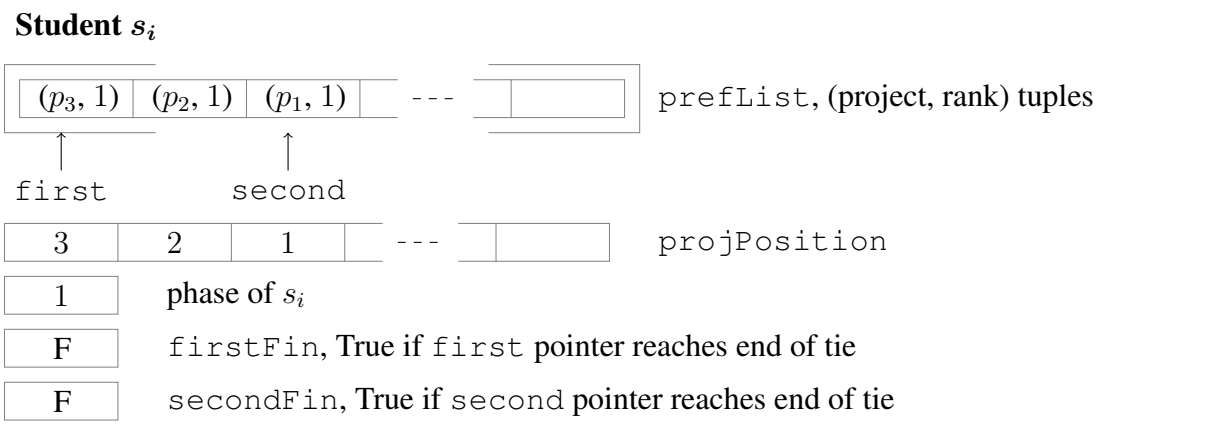
Next, using the data structures summarised in Figure 5.4, Lemma 5.5.15 proves that all operations inside the main while loop of Algorithm 5.2 run in constant time.

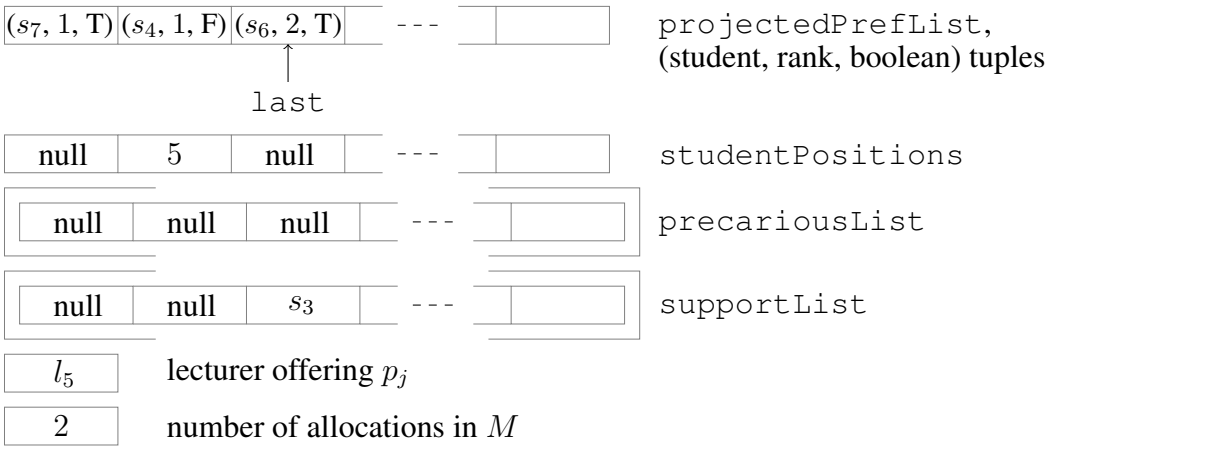
Lemma 5.5.15. *All operations inside the main while loop of Algorithm 5.2 run in constant time.*

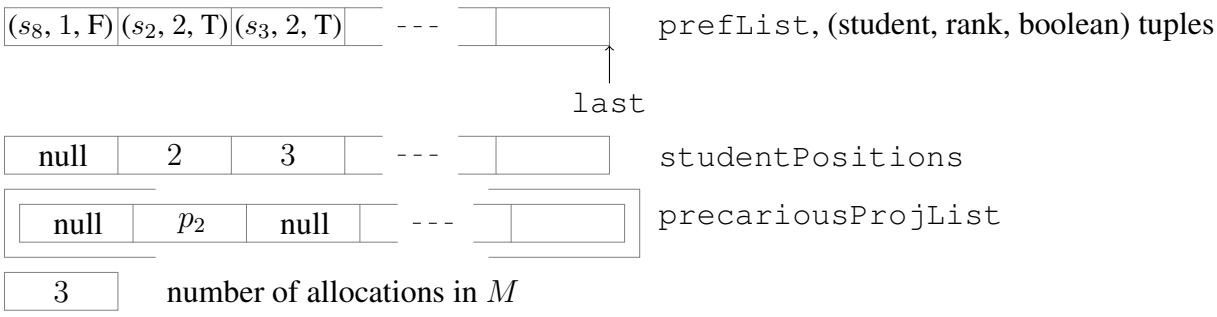
Proof. The data structures required are described below and are summarised in Figure 5.4. For initialisation purposes, each student, project and lecturer has a list of length n_2, n_1 and n_1 respectively, each entry of which requires $O(1)$ space. In order to not exceed a time complexity of order the sum of lengths of preference lists, a process of virtual initialisation is used on these data structures [8, p. 149].

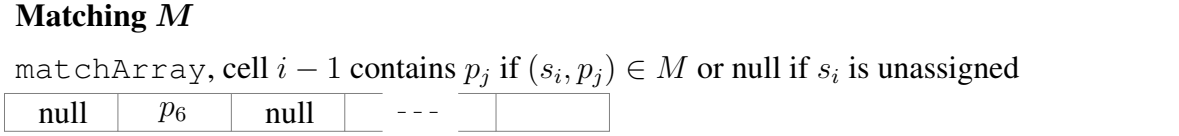
Student data structures. For each student a doubly-linked list of (project, rank) tuples embedded in an array, `prefList`, stores their preference list in order of rank, representing the undeleted entries. A small example is shown in Figure 5.4 with p_3, p_2 and p_1 all of rank 1 on s_i's preference list. Entries may be deleted from this array; a copy of this list prior to any deletions being carried out is retained in order to allow a second iteration through a student's preference list, if they move into phase 2. An array `projPosition` of length n_2 retains links to the position of (project, rank) tuples in `prefList`, allowing a constant time removal of projects from `prefList`. An integer variable associated with each student stores which phase this student is in. Examples for these final two data structures are also shown in Figure 5.4.

Project data structures. Each project has a link to their supervising lecturer. An array, `projectedPrefList` stores the projected preference list of l_k for p_j in the form of (student, rank, boolean) tuples. As an example, suppose p_j has a projected preference list starting with s_7 at rank 1, s_4 at rank 1 and s_6 at rank 2 as is shown in Figure 5.4. The boolean values indicate which student-project pairs are currently in the matching.









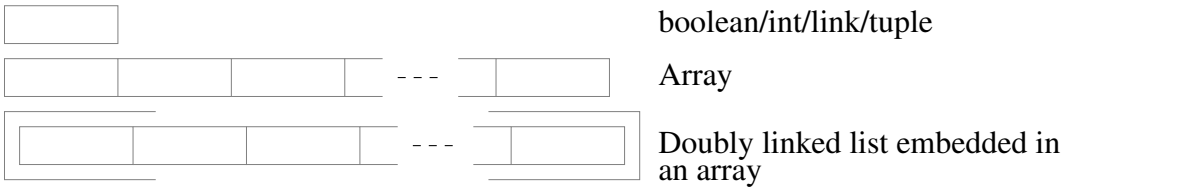


Figure 5.4: Data structures guide for Lemma 5.5.15.

Once a project is full and non-precarious it cannot accept a worse student than it already has for the remainder of the algorithm, according to Proposition 5.5.5. Assume p_j is full and non-precarious. Let the worst student assigned to p_j be given by s_w. We retain a pointer, `last`, which points to the rightmost student at the same rank as s_w in p_j's `projectedPrefList`. This pointer must move from right to left in a linear fashion (moving up in ranks) given the above proposition.

During the course of the algorithm, we may need to remove the worst student according to l_k from $M(p_j)$. It is possible that there are two or more students who are worst assignees (according to rank) with some being in phase 1 and some in phase 2. In order to ensure that we prioritise the removal of phase 1 students, two pointers are added for each entry in `projectedPrefList`, which point to the head of a phase 1 and a phase 2 doubly-linked list associated with that tie embedded in the `projectedPrefList` array (this data structure is not shown in Figure 5.4). Adding or removing a phase 1 or 2 student to either list takes constant time, as they do not need to be kept in order. Then, un-assigning a student requires a check to be made in the tie associated with the worst position (found using `last`), in order to prioritise a phase 1 student's removal. In total this takes constant time. Note that a student can only change phase if they are not allocated and therefore updating an allocated student's phase in these lists is not necessary unless they have just been added.

Each project also contains a doubly-linked list embedded in an array of students, denoted by `precariousList`, containing students who have formed a precarious pair with this project. In the example in Figure 5.4, p_j is non-precarious and so no students form a precarious pair with p_j. Adding to and removing from this list takes constant time if we assume that s_i is stored at index $i-1$. A project p_j *supports* a student s_i in being precarious if $(s_i, p_{j'}) \in M$ and p_j is the first fully available project at the same rank as $p_{j'}$ in s_i's list. Then, a doubly-linked list embedded in an array of students, `supportList`, stores the students for which p_j gives their support. As before, adding to and removing from this list takes constant time. A counter stores the number of students assigned to p_j in M.

Lecturer data structures. For each lecturer an array of (student, rank, boolean) tuples, `prefList`, stores their preference list in order of rank, with a True value stored in the ith boolean if student s_i is assigned to a project of l_k's. Figure 5.4 shows an example with s_8 not assigned to l_k at rank 1 and s_2 and s_3 both assigned at rank 2. Each lecturer also has an array of length n_1, `studentPositions` which retains links to the position of (student, rank, boolean) tuples in l_k's `prefList`, and a counter stores the number of students assigned to l_k in M. A doubly-linked list embedded in an array of projects, `precariousProjList`, stores the projects offered by l_k that are precarious, where project p_j is stored at index $j-1$ in this list if it is precarious. Figure 5.4 shows p_2 being a precarious project of l_k.

Similar to projects, by Proposition 5.5.5, once a lecturer is full and non-precarious they

cannot accept a worse student than they already have for the remainder of the algorithm. Assume l_k is full and non-precarious. Using similar data structures described in the Project Data Structures section above we are able to find l_k's worst assigned student in constant time.

Student pointers. A student s_i retains two pointers for the project(s) tied at the head of their list. One pointer `first` stores the first fully available project when iterating from left to right, and `second` stores the second fully available project. If `first` (respectively `second`) reaches the end of the tie, then a boolean `firstFin` (respectively `secondFin`) is set to True. For each iteration of the main while loop of Algorithm 5.2, each student s_i first seeks a project at the head of their list that is fully available. This will be precisely the project p_j that `first` points to. Then if `second` has not reached the end of the tie, (s_i, p_j) (if added to the matching) is precarious and the project that `second` points to, $p_{j'}$, supports p_j. If however, `secondFin` is set to True, then (s_i, p_j) (if added to the matching) is non-precarious. Finally, if `firstFin` is set to True, then the leftmost project at the head of s_i's list is a favourite project, with (s_i, p_j) also being unable to become precarious. Proposition 5.5.14 shows that maximum number of applications a student can make to a project on their preference list is 3. At most twice in phase 1 (twice if removed as a precarious pair) and once in phase 2.

During phase 2 there are no fully available projects on s_i's list since s_i must have applied and been rejected (in some way) from every project on their preference list at least once already. Therefore, the `first` and `second` pointers are not required in phase 2. During phase 1 the the `first` and `second` pointers are only required to iterate once over each tie as described above. Hence, the maximum number of times a student's list is iterated over is 4; once each for the two pointers `first` and `second`, once again after `first` and `second` have reached the end of the tie at the head of a student's list (the student may have retained projects at the head of their list after this point if the projects were precarious), and finally once during phase 2.

Matching data structures. The current matching is stored in an array of cells `matchArray` where cell $i - 1$ contains project p_j if $(s_i, p_j) \in M$ or null otherwise. Figure 5.4 shows and example with student s_2 being assigned to project p_6.

Processes (in the order encountered in Algorithm 5.2):

Let lecturer l_k offer project p_j.

1. *A student s_i applies to a favourite project*: if s_i's `FirstFin` is set to True then there are no fully available projects at the head of s_i's list, and a favourite project of s_i will be the leftmost project. If however, `FirstFin` is False then there are fully available projects at the head of their list and a favourite project of s_i is pointed to by `first`, which is retrievable in constant time.

2. *Deciding if a project p_j is undersubscribed or full or deciding if a lecturer l_k is undersubscribed or full*: Using the counters described above a comparison can be made between p_j's capacity and their current number of allocations. A similar comparison can be made for l_k. Both can be achieved in constant time.

3. *Deciding if project p_j is fully available*: p_j would not be fully available if either p_j is full or l_k is full. Therefore a comparison of the number of allocations for p_j and l_k and their respective capacities is required. Again this can be achieved in constant time.

4. *Adding a pair (s_i, p_j) to M*: Project p_j is placed in the $i-1$'th cell of `matchArray` and p_j and l_k's allocation counters are incremented. Project p_j's `projectedPrefList` and lecturer l_k's `prefList` booleans are updated in constant time using their associated `studentPositions` data structures. Each tuple in these lists has a link to the head of phase 1 and phase 2 lists for their tie. When the pair is added the tuple is added to either the phase 1 or phase 2 list. If (s_i, p_j) is precarious then s_i is added to p_j's `precariousList` and the project pointed to by `second`, $p_{j'}$ adds s_i to their `supportList`. If p_j has just changed from being non-precarious to precarious then l_k adds p_j to their `precariousProjList`. If the addition of (s_i, p_j) to M means that p_j goes from being fully available to not being fully available then we need to ensure that other students who rely on p_j as their support are updated. Therefore p_j alerts each student on their `supportList` that they are no longer to be relied upon as a fully available project. This triggers each of those students to update their `second` pointers. The time required for this can be attributed to the movement of `second` pointers as noted earlier. After being alerted, some other pair in M may stop being precarious, but any changes can be conducted in constant time as described above. If on adding pair (s_i, p_j), p_j has now become full and non-precarious then the `last` pointer will move from right to left over `projectedPrefList` until it reaches the end of a tie whose phase 1 and phase 2 lists are non-empty. From this point on `last` is only updated upon removing a pair from M (Point 8).

5. *Deciding if l_k is precarious, and returning a precarious project if one exists*: Checking whether `precariousProjList` is empty for l_k is a simple process that takes constant time. Retrieving a precarious pair should one exist requires selection of the first student from l_k's `precariousProjList` and can be done in constant time.

6. *Finding a worst assignee of l_k and deciding if l_k meta-prefers s_i to this worst assignee*: This operation only needs to be executed if l_k is full and non-precarious. In that situation l_k's `last` pointer will point to the rightmost position in a tie in `prefList` such that l_k's current worst student s_w is assigned at the same rank. Then as previously discussed, all that is required is to check the links to phase 1 and phase 2 students for

this tie and return a phase 1 student if one exists, or phase 2 student if not. This can be conducted in constant time. Deciding if l_k meta-prefers s_i to s_w can also be done in constant time by comparing rank and phase.

7. *Removing a preference list entry from s_i's list*: This process is shown in Algorithm 5.3 which runs in constant time, since we can find a specific project p_j in s_i's `prefList` using the `projPosition` array.

8. *Removing a pair (s_i, p_j) from M*: The $i-1$'th cell of `matchArray` is set to null, and p_j and l_k's allocation counters are decremented. Project p_j's `projectedPrefList` and lecturer l_k's `prefList` booleans are updated in constant time. The tuples associated with s_i in these lists are removed from their phase 1 or phase 2 list in constant time. If a pair (s_i, p_j) is removed from M, then this is either because p_j or l_k is full. By Proposition 5.5.3, p_j cannot subsequently become fully available. Thus, the removal of a pair cannot change p_j's fully available status. All that is required then is to check whether (s_i, p_j) was precarious, and update p_j's `precariousList` and l_k's `precariousProjList`. If on removing pair (s_i, p_j), the `last` pointer now points to a tie with empty phase 1 and phase 2 lists, `last` needs to be updated and accordingly moves from right to left until it reaches the end of a tie with a non-empty phase 1 or phase 2 list.

9. *Deciding if p_j is precarious, and returning a precarious pair if one exists*: Similar to Point 5 above but using the `precariousList` of p_j.

10. *Finding a worst assignee of p_j according to l_k and deciding if l_k meta-prefers s_i to this worst assignee*: Similar to Point 6 above, this operation is only required if p_j is full and non-precarious, at which point p_j's `last` pointer will point to the rightmost position in a tie in `projectedPrefList` such that l_k's current worst student assigned to p_j, s_w is assigned at the same rank. As above retrieving s_w and comparing its rank and phase with s_i takes constant time.

Therefore, all operations inside the main while loop of Algorithm 5.2 run in constant time. □

Algorithm 5.5 is a more detailed version of Algorithm 5.4, indicating how the operations in Algorithm 5.4 can be implemented efficiently. Proposition 5.5.16 shows that Algorithm 5.5 runs in linear time.

Proposition 5.5.16. *The time complexity of Algorithm 5.5 is $O(m)$ where m is the total length of student preference lists.*

Algorithm 5.5 Promote-Students(M), subroutine for Algorithm 5.2 (detailed view). Removes all blocking pairs of type $(3bi)$.

Require: SPA-ST instance I and matching M which does not contain blocking pairs of type *(3a)*, *(3bii)* or *(3c)*.
Ensure: Return a stable matching M.

```
1: Create data structures as described in Proposition 5.5.16
2: while S ≠ ∅ do
3:     Pop p_j from stack S
4:     Remove the first student s_i from list ρ_j
5:     Let p_k = M(s_i)
6:     if s_i prefers p_j to p_k then      ▷ p_j is undersubscribed, s_i is assigned and prefers p_j to M(s_i)
7:         M ← M\{(s_i, p_k)}
8:         M ← M ∪ {(s_i, p_j)}
9:         Let ρ_k be the list of student and rank tuples associated with project p_k and let boolean β_k indicate whether M(s_i) is on stack S
10:        if ρ_k ≠ ∅ then
11:            Push p_k onto stack S if it is not already on S          ▷ Using β_k.
12:        end if
13:    end if
14:    if ρ_j ≠ ∅ and p_j is undersubscribed then
15:        Push p_j onto stack S                                       ▷ p_j cannot currently be on S
16:    end if
17: end while
18: return M
```

Proof. Abraham *et al.* [4] describes the process of a sequence of promotions for houses in HA in order to return a *trade-in-free* matching. A similar process is described here to remove all blocking pairs of type *(3bi)*. We create the following data structures.

- A linked list ρ_j of students s_i for each project p_j such that s_i is assigned in M and finds p_j acceptable. We may also start by assuming that ρ_j involves only students who prefer p_j to $M(s_i)$, however the time complexity is unaffected by this.
- A ranking list r_i for each student s_i built as an array such that $r_i = j$ contains the rank of p_j for student s_i;
- A stack S of undersubscribed projects p_j, such that ρ_j is non-empty, is created;
- A variable β_j for each project p_j which records whether p_j is already in S.

These data structures can be initialised in $O(m)$ time where m is the total length of student preference lists.

Execution of Algorithm 5.5 proceeds as follows. For each iteration of the while loop a project p_j is taken from stack S. Project p_j must be undersubscribed and have non-empty

list ρ_j. The first student s_i from ρ_j, is removed and if s_i would prefer to be assigned to p_j than $p_k = M(s_i)$ (found by comparing ranking list entries for p_j and p_k) then we remove pair (s_i, p_k) from M and add (s_i, p_j). Now, p_k is certainly an undersubscribed project and it is added to S (unless it already exists on S). Whether or not (s_i, p_j) is added to M, p_j may still be undersubscribed. If ρ_j is non-empty and p_j is undersubscribed, then p_j is added to S.

With each iteration we remove a tuple from some project's ρ list. These lists must be finite because preference lists are finite and therefore Algorithm 5.5 will terminate with empty S. It is clear that Algorithm 5.5 will take $O(m)$ time where m is the total length of student preference lists. □

Finally, Theorem 5.5.17 establishes that Algorithm Max-SPA-ST-Approx runs in linear time with respect to the total length of student preference lists.

Theorem 5.5.17. *Algorithm 5.2 always terminates and runs in linear time with respect to the total length of student preference lists.*

Proof. By Proposition 5.5.14 each student can apply to a project on their preference list a maximum of three times during the main while loop of Algorithm 5.2. Since all operations within this while loop run in constant time by Proposition 5.5.15, this part of the algorithm must run in $O(3m) = O(m)$ time, where m is the total length of student preference lists. Proposition 5.5.16 shows that Algorithm 5.5 also runs in $O(m)$ time, therefore so must Algorithm 5.2. Finally, since student preference lists are of finite length, Algorithm 5.2 must terminate. □

5.5.5 Performance guarantee

5.5.5.1 Introduction

In Section 5.5.3 we showed that Algorithm Max-SPA-ST-Approx always produces a stable matching. In this section we show that any stable matching produced by this algorithm is always at least two-thirds of the size of a maximum stable matching.

We begin in Section 5.5.5.2 by giving preliminary definitions of an *underlying graph* and *mapped graph* of an instance of SPA-ST, which are used throughout the proofs in this section. Both the *underlying graph* and the *mapped graph* are illustrated in an example in Section 5.5.5.3. In Section 5.5.5.4 we move on to look at the possible structures that may exist in *mapped graphs*. Finally, using these structures, in Section 5.5.5.5 we prove that Algorithm Max-SPA-ST-Approx has a performance guarantee of $\frac{3}{2}$.

5.5.5.2 Preliminary definitions

The *underlying graph* G of an SPA-ST instance I consists of sets of student, project and lecturer vertices. Edges exist between a student vertex s_i and a project vertex p_j if s_i finds p_j acceptable. Edges exist between a project vertex p_j and lecturer vertex l_k if l_k offers p_j.

We now introduce the notion of a *mapped graph* G' of the underlying graph G of the SPA-ST instance I. This graph is not created in Algorithm 5.2, but is intended only to make it easier to prove the performance guarantee in Section 5.5.5.5. The *mapped graph* G' is created in the following way. Let all student vertices remain unchanged. Let M be the matching found by Algorithm 5.2 for instance I and let M_{opt} be a maximum stable matching in I. For each lecturer vertex l_k we create multiple *cloned* vertices $l_k^1 \dots l_k^{r_k}$, where $r_k = d_k - |M(l_k) \cap M_{opt}(l_k)|$ and d_k is l_k's capacity. In G there are edges between students and projects, and projects and lecturers, whereas G' contains only edges between students and lecturer clones.

An (s_i, p_j) edge in G corresponds to an (s_i, l_k^r) edge in G', where l_k^r denotes the r^{th} lecturer clone of lecturer l_k. Edges in G' are given by M' and M'_{opt}, defined below. M'_{opt} edges are defined as follows. For each lecturer l_k, if $\bigcup_{p_{j'} \in P_k} \{M_{opt}(p_{j'}) \backslash M(p_{j'})\} = \{s_{i_1}, ..., s_{i_t}\}$ then add $(s_{i_r}, l_k^r), (1 \leq r \leq t)$ to M'_{opt}, the mapped version of M_{opt} in G'. M' edges are then added using Algorithm 5.6. By using this algorithm we ensure that where possible, pairs of edges in M' and M'_{opt} involving the same project are assigned to the same lecturer clone in G'. According to Algorithm 5.6 we do the following. A copy of $M \backslash M_{opt}$ is created and denoted M_0 which intuitively contains the set of student-project pairs that have not yet been mapped. L_0 is a copy of the set of all lecturer clone vertices, and L'_0 is the empty set. Intuitively, L'_0 will collect up any remaining lecturer clones, after pairs of edges in M' and M'_{opt} involving the same project are dealt with. For each lecturer clone $l_k^r \in L_0$, if there is an edge (s_i, l_k^r) in M'_{opt} for some s_i then we let p_j be the project assigned to s_i in M_{opt}. If there is not, then l_k^r is added to L'_0. Assuming $(s_i, l_k^r) \in M'_{opt}$, then we check if there is an edge $(s_{i'}, p_j)$ in M_0 for some student $s_{i'}$. Again, if there is not then l_k^r is added to L'_0. If $(s_{i'}, p_j) \in M_0$ for some student $s_{i'}$ then we add edge $(s_{i'}, l_k^r)$ to M' and remove $(s_{i'}, p_j)$ from M_0. After all lecturer clones have been tested, then for each student-project pair (s_i, p_j) remaining in M_0 we find an unused lecturer clone $l_k^r \in L'_0$, where l_k offers p_j, and add (s_i, l_k^r) to M'. Project vertices and all other edges are ignored in G'.

5.5.5.3 Example mapped graph

In this section we introduce an example to demonstrate the creation of mapped graph G' from underlying graph G and matchings M and M_{opt}. Figure 5.5 shows example instance I_4 of SPA-ST.

Algorithm 5.6 Create-Mapped(M), obtains a set of edges M' for the mapped graph G' corresponding to edges in $M \backslash M_{opt}$.

Require: An instance I of SPA-ST, a stable matching M and maximum stable matching M_{opt} of I and a mapped version M'_{opt} of M_{opt}.
Ensure: Return a mapped version M' of $M \backslash M_{opt}$.

$M_0 \leftarrow M \backslash M_{opt}$ ▷ where M_0 is the working set of student-project pairs in M
Let L_0 be a copy of the set of all lecturer clones
$L'_0 \leftarrow \emptyset$
$M' \leftarrow \emptyset$
while L_0 is non-empty **do**
    Remove a lecturer clone l^r_k from L_0
    if (s_i, l^r_k) is an edge in M'_{opt} for some s_i **then**
        Let p_j be the project assigned to s_i in M_{opt}
        if $(s_{i'}, p_j)$ is in M_0 for some student $s_{i'}$ **then**
            $M' \leftarrow M' \cup \{(s'_i, l^r_k)\}$
            $M_0 \leftarrow M_0 \backslash \{(s'_i, p_j)\}$
        else
            $L'_0 \leftarrow L'_0 \cup \{l^r_k\}$
        end if
    else
        $L'_0 \leftarrow L'_0 \cup \{l^r_k\}$
    end if
end while
while M_0 is non-empty **do**
    Pick some $(s_i, p_j) \in M_0$
    $M_0 \leftarrow M_0 \backslash \{(s_i, p_j)\}$
    Let l^r_k be some lecturer clone in L'_0, where l_k offers $p_j \triangleright l^r_k$ must exist since there are $d_k - |M(l_k) \cap M_{opt}(l_k)|$ clones for l_k
    $L'_0 \leftarrow L'_0 \backslash \{l^r_k\}$
    $M' \leftarrow M' \cup \{(s_i, l^r_k)\}$
end while
return M'

Let $M = \{(s_1, p_1), (s_2, p_2), (s_3, p_3)\}$ and $M_{opt} = \{(s_1, p_1), (s_2, p_1), (s_3, p_2), (s_4, p_3)\}$ be stable matchings in I. Clearly, M_{opt} is also a maximum stable matching as all students are assigned. Figure 5.6a shows the underlying graph G of instance I_4. To create the vertices of G', student vertices are copied, and multiple lecturer cloned vertices are created. For lecturer vertices l_1 and l_2 in G with capacities of 2, we create l^1_1, l^2_1, l^1_2 and l^2_2 in G'. Using the definition of M'_{opt} above, we obtain the edge set $M'_{opt} = \{(s_2, l^1_1), (s_3, l^1_2), (s_4, l^2_2)\}$. Figure 5.6b shows a part built G' with all M'_{opt} edges added.

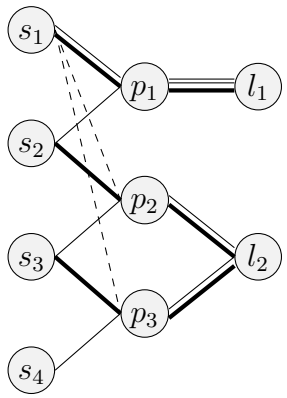
Next M' is calculated using Algorithm 5.6. A copy of $M \backslash M_{opt}$ is created and denoted $M_0 = \{(s_2, p_2), (s_3, p_3)\}$. $L_0 = \{l^1_1, l^2_1, l^1_2, l^2_2\}$ is a copy of the set of all lecturer cloned vertices, and L'_0 is the empty set. We iterate through L_0 as follows.

Student preferences:
s_1: p_1 p_2 p_3
s_2: (p_1 p_2)
s_3: (p_3 p_2)
s_4: p_3

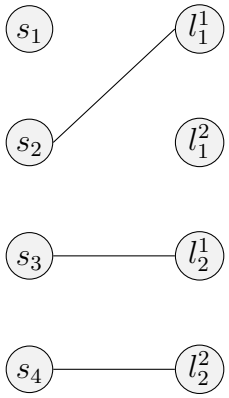
Project details:
p_1: lecturer l_1, $c_1 = 2$
p_2: lecturer l_2, $c_2 = 1$
p_3: lecturer l_2, $c_3 = 1$

Lecturer preferences:
l_1: s_1 s_2 $d_1 = 2$
l_2: (s_2 s_3) s_4 s_1 $d_2 = 2$

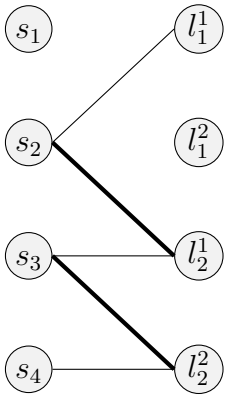
Figure 5.5: SPA-ST instance I_4.



(a) The underlying graph G. M and M_{opt} are shown in bold and non-bold edges respectively. Edges not in $M \cup M_{opt}$ are dashed.



(b) Part-built G'. All student and lecturer clone vertices are added. $M'_{opt} = \{(s_2, l_1^1), (s_3, l_2^1), (s_4, l_2^2)\}$ edges are also shown.



(c) G' with edge set $M' \cup M'_{opt}$, where $M'_{opt} = \{(s_2, l_1^1), (s_3, l_2^1), (s_4, l_2^2)\}$ (non-bold edges) and $M' = \{(s_2, l_2^1), (s_3, l_2^2)\}$ (bold edges), is shown.

Figure 5.6: Example illustrating the underlying graph G and mapped graph G' of instance I_4, relative to two stable matchings M and M_{opt} in G.

- Lecturer clone l_1^1 is removed from L_0. Since there is an edge $(s_2, l_1^1) \in M'_{opt}$ and s_2 is assigned p_1 in M_{opt}, but $(s_{i'}, p_j) \notin M_0$ for each student $s_{i'}$, l_1^1 is added to L'_0;
- Lecturer clone l_1^2 is removed from L_0. As there is no edge $(s_i, l_1^2) \in M'_{opt}$ for any student s_i, l_1^2 is added to L'_0;
- Next lecturer clone l_2^1 is removed from L_0. There is an edge $(s_3, l_2^1) \in M'_{opt}$, s_3 is assigned p_2 in M_{opt} and there is an edge $(s_2, p_2) \in M_0$, hence (s_2, l_2^1) is added to M' and (s_2, p_2) is removed from M_0;
- Using the same reasoning when the final lecturer clone l_2^2 is removed from L_0, (s_3, l_2^2) is also added to M' and (s_3, p_3) is removed from M_0.

As M_0 is now empty, we do not enter the final while loop on Line 19 of Algorithm 5.6 therefore M' is now complete. Figure 5.6c shows the completed mapped graph G' with edge set $M' \cup M'_{opt}$.

5.5.5.4 Components in G'

In this section we define the possible structures that may exist in the mapped graph G'.

An *alternating path* in G' is defined as a path that comprises edges in M_{opt} and in M alternately. A path or alternating path is described as $even$ if there are an even number of edges in the path, *odd* otherwise. Finally, an *alternating cycle* is a sequence of edges in M_{opt} and M alternately, which forms a cycle.

A *component* c in G' is defined as any maximal connected subgraph in G'. Figure 5.7 shows the possible component structures that may be found in G' which are described in more detail below. Let $n_{c,l}$ and $n_{c,s}$ denote the maximum number of lecturer clone vertices and student vertices respectively, in some component c of G', and let $n_c = \max\{n_{c,l}, n_{c,s}\}$. Notation for a lecturer clone in component c is defined as $l^{c,r}$ indicating the r^{th} lecturer clone of component c. Similarly, $s^{c,r}$ indicates the r^{th} student of component c.

Each vertex in G' is incident to at most one M' edge and at most one M'_{opt} edge, meaning every component must be a path or cycle comprising alternating M' and M'_{opt} edges. Therefore the structure of each component must have one of the following forms.

(a) An alternating cycle;

(b) An even length alternating path, with lecturer clone end vertices;

(c) An even length alternating path, with student end vertices;

(d) An odd length alternating path, with end edges in M';

(e) An odd length alternating path, with end edges in M'_{opt}, for $n_c \geq 3$;

(f) An odd length alternating path, with end edges in M'_{opt}, for $n_c = 2$;

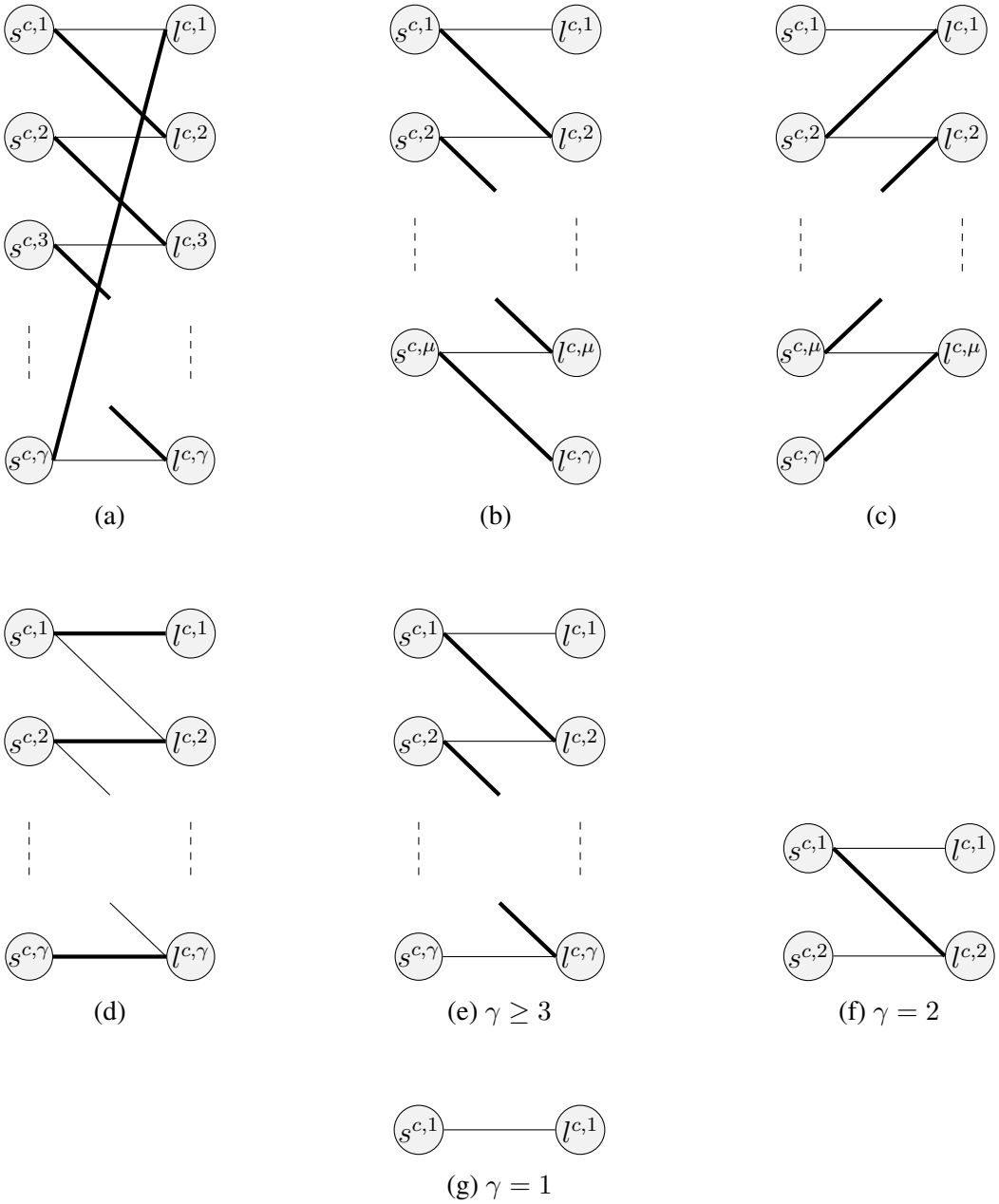
(g) An odd length alternating path, with end edges in M'_{opt}, for $n_c = 1$;

We wish to show that any stable matching found by Algorithm 5.2 must be at least two-thirds of the size of M_{opt}.

5.5.5.5 Proof of the $\frac{3}{2}$ performance guarantee

In this section we prove that any stable matching produced by Algorithm Max-SPA-ST-Approx must be at least two-thirds of the size of a maximum stable matching.

Propositions 5.5.18 and 5.5.19 detail two configurations of components in G' where we may infer that a project is undersubscribed in either M or M_{opt}. The following terminology, used



(a) (b) (c)

(d) (e) $\gamma \geq 3$ (f) $\gamma = 2$

(g) $\gamma = 1$

Figure 5.7: Possible component structures in G' for a component c, where $\gamma = n_c$, the size of the component, and $\mu = \gamma - 1$. M' and M'_{opt} edges are shown in bold and non-bold, respectively.

in these first two propositions, is now introduced. Let $M'_{opt}(p_j)$ denote the set of students who are paired with lecturer clones in M'_{opt} associated with project p_j. Then $M'_{opt}(p_j) = M_{opt}(p_j) \backslash M(p_j)$ by construction. Finally, let M_0^{α} denote the value of $M_0 = M \setminus M_{opt}$ on Line 1 of Algorithm 5.6, i.e. the original set of unmapped student-project pairs.

Proposition 5.5.18. *Let $(s_i, l^{c,r}) \in M'_{opt}$ be an edge in G' where $(s_i, p_j) \in M_{opt}$. If $l^{c,r}$ is unassigned in M' or if there exists an edge $(s_{i'}, l^{c,r}) \in M'$ where $s_{i'}$ is assigned to a project other than p_j in M, then $|M_{opt}(p_j)| > |M(p_j)|$, and hence p_j is undersubscribed in M.*

Proof. Suppose $l^{c,r}$ is unassigned in M' or $(s_{i'}, l^{c,r}) \in M'$ where $s_{i'}$ is assigned a project other than p_j in M. Assume for contradiction that $|M_{opt}(p_j)| \leq |M(p_j)|$. During the execution of Algorithm 5.6, the first while loop iterates over the lecturer clones in G' once. This means that all edges in $M'_{opt}(p_j)$ are iterated over.

Since $|M_{opt}(p_j)| \leq |M(p_j)|$, we know that $|M_{opt}(p_j) \backslash M(p_j)| \leq |M(p_j) \backslash M_{opt}(p_j)|$. But then $|M'_{opt}(p_j)| \leq |M_0^{\alpha}(p_j)|$ and so it must be the case that every lecturer clone in a pair of $M'_{opt}(p_j)$ (including $l^{c,r}$) is paired with a student in $M_0^{\alpha}(p_j)$. This contradicts the fact that $l^{c,r}$ is either unassigned in M' or $(s_{i'}, l^{c,r}) \in M'$ where $s_{i'}$ is assigned to a project other than p_j in M. It follows immediately that p_j is undersubscribed in M. □

Proposition 5.5.19. *Let $(s_i, l^{c,r}) \in M'$ be an edge in G' where $(s_i, p_j) \in M$. If there exists an edge $(s_{i'}, l^{c,r}) \in M'_{opt}$ where $s_{i'}$ is assigned to a project other than p_j in M_{opt}, then $|M(p_j)| > |M_{opt}(p_j)|$, and hence p_j is undersubscribed in M_{opt}.*

Proof. We use a similar proof to Proposition 5.5.18. Suppose there exists an edge $(s_{i'}, l^{c,r}) \in M'_{opt}$ where $s_{i'}$ is assigned a project other than p_j in M_{opt} and assume for contradiction that $|M(p_j)| \leq |M_{opt}(p_j)|$. During the execution of Algorithm 5.6, the first while loop iterates over the lecturer clones in G' once, meaning that all edges in $M'_{opt}(p_j)$ are iterated over.

As $|M(p_j)| \leq |M_{opt}(p_j)|$, we know that $|M(p_j) \backslash M_{opt}(p_j)| \leq |M_{opt}(p_j) \backslash M(p_j)|$. But then $|M_0^{\alpha}(p_j)| \leq |M'_{opt}(p_j)|$ and so each student in $M_0^{\alpha}(p_j)$ is paired with a lecturer clone that exists in a pair of $M'_{opt}(p_j)$. But this contradicts the fact that both $(s_i, l^{c,r}) \in M'$ with $s_i \in M_0^{\alpha}(p_j)$ and $(s'_i, l^{c,r}) \in M'_{opt}$ with $s_i \notin M'_{opt}(p_j)$. Hence $|M(p_j)| > |M_{opt}(p_j)|$ and so p_j is undersubscribed in M_{opt}. □

We now give three proofs of preliminary results that are used to aid Lemma 5.5.23 and Lemma 5.5.24 in showing that it is not possible for a component of the types shown in Figure 5.7f or in Figure 5.7g to exist in G'.

First, Proposition 5.5.20 shows that for a component of the type shown in Figure 5.7f, neither student $s^{c,1}$ nor $s^{c,2}$ can have applied to the project $s^{c,1}$ is assigned to in M_{opt}, during Algorithm 5.2's execution.

Proposition 5.5.20. *Let c be the component of G' in Figure 5.7f. Let $s^{c,1}$ be assigned to project p_j in M_{opt}. Then project p_j is fully available in M and, $s^{c,1}$ and $s^{c,2}$ can never have applied to p_j at any point in Algorithm 5.2's execution.*

Proof. Let lecturer clone $l^{c,1}$ correspond to lecturer l_k. In G', lecturer clone $l^{c,1}$ is unassigned in M', therefore we know that l_k is undersubscribed in M. Since there is no edge in M' incident to $l^{c,1}$, by Proposition 5.5.18, we know p_j is undersubscribed in M. Project p_j is, by definition, fully available in M and, by Proposition 5.5.3, must have been fully available throughout the algorithm's execution.

Now we prove that neither $s^{c,1}$ nor $s^{c,2}$ can have applied to p_j. In Algorithm 5.4, students can only apply to projects that are not fully available by Proposition 5.5.4, hence we only look at the main while loop of Algorithm 5.2. We consider $s^{c,1}$ first. Assume for contradiction that $s^{c,1}$ applied to p_j at some point during the main while loop of Algorithm 5.2's execution. Then $(s^{c,1}, p_j)$ would be added to M as p_j was always fully available. But we know that $(s^{c,1}, p_j)$ is not in the final matching M hence it must have been rejected by l_k at some point. But this can only have happened if p_j was not fully available, which contradicts the fact that p_j is always fully available above. Therefore $s^{c,1}$ can never have applied to p_j at any point. By identical reasoning $s^{c,2}$ can also never have applied to p_j. □

Next, in Proposition 5.5.21, we prove a more general result that if a student applies to a project while they are in phase 2, then that project is neither fully available nor precarious.

Proposition 5.5.21. *Suppose that student s_i applied to project p_j in phase 2 of Algorithm 5.2 and denote this time by T_0. Then at time T_0, p_j is not fully available and is non-precarious.*

Proof. Let l_k be the lecturer offering p_j. Assume for contradiction that p_j is fully available at T_0. Since s_i is applying to p_j in phase 2, l_k must have rejected s_i when s_i was in phase 1. But this can only happen if p_j is not fully available and by Proposition 5.5.3, p_j cannot again become fully available.

Assume then that p_j is precarious at T_0. Then there must exist a precarious pair $(s_{i'}, p_j)$ in the matching for some student $s_{i'}$. We know from Proposition 5.5.5 that when a project is not fully available and non-precarious, it cannot again become precarious. Therefore, when s_i applied in phase 1 to p_j, it was either fully available or precarious (or both), and so (s_i, p_j) must have been added to the matching. But at some point before T_0, since s_i is applying in phase 2, (s_i, p_j) was removed from the matching. This can only happen when p_j is not fully available and either (s_i, p_j) is precarious or is a worst student in $M(p_j)$ (also a worst assignee of $M(l_k)$).

If (s_i, p_j) was precarious then, once removed, s_i would again apply to p_j in phase 1 and must be successfully added for the same reason as before, although this time as a non-precarious

pair (since other fully available projects tied with p_j on s_i's list would be applied to by s_i before p_j). The removal of non-precarious (s_i, p_j) can only happen because p_j is non-precarious which contradicts the assumption that p_j is precarious at T_0 by Proposition 5.5.5, since p_j is also not fully available. Therefore p_j is non-precarious at T_0.

Therefore p_j can be neither fully available nor precarious at T_0. □

Finally, Proposition 5.5.22, shows that if a project is full and non-precarious at some point before the end of the main while loop of Algorithm 5.2, then this project cannot subsequently accept a worse student (according to the lecturer who offers it).

Proposition 5.5.22. *Let T_{end} denote the point in Algorithm 5.2's execution at the end of the main while loop. If a project p_j offered by l_k is full and non-precarious before T_{end}, then a student s_i worse than l_k's worst ranked assignees in $M(p_j)$ cannot subsequently become assigned to p_j.*

Proof. As in Proposition 5.5.8, let T_0 be a point of the algorithm's execution, mentioned in the statement of the propostion, where p_j is full and non-precarious. By Proposition 5.5.8, we know that after T_0 a student s_i worse than l_k's worst ranked assignees in $M(p_j)$ cannot subsequently become assigned to p_j before T_{end}. Hence we concentrate only on changes made by Algorithm 5.4. Additionally, due to this same result, it suffices to show that a student s_i worse than l_k's worst ranked assignees in $M(p_j)$ at T_x (for T_x in the range T_0 to T_{end}) cannot subsequently become assigned to p_j.

Project p_j is either full or undersubscribed at T_{end}. We deal with each case in turn.

- Assume first, that p_j is full at T_{end}. If p_j remains full then no student can become assigned to p_j. Therefore assume that p_j becomes undersubscribed after T_{end} and let T_1 be the first time this occurs. It must be that, just before T_1, a blocking pair $(s_{i'}, p_j)$ exists, for some student $s_{i'}$. By Propositions 5.5.10 and 5.5.11, $s_{i'} \in M(p_j)$ is one of the worst students in $M(l_k)$. Since T_1 is the first time p_j becomes undersubscribed, no students have been removed from $M(p_j)$ since T_{end}, and so $s_{i'}$ must have existed in $M(p_j)$ at T_{end} as well. Since no students are introduced in Algorithm 5.4, $s_{i'} \in M(p_j)$ is also one of the worst students in $M(l_k)$ at T_{end}. Therefore, by Proposition 5.5.8, p_j cannot subsequently be assigned a worse student than exists in $M(p_j)$ at T_{end}.

- Assume now that p_j is undersubscribed at T_{end}. Then there was some point T_2, before T_{end} and after T_0, that p_j became undersubscribed by the removal from M of $(s_{i'}, p_j)$ for some student $s_{i'}$. We know that $(s_{i'}, p_j)$ is non-precarious since p_j is non-precarious after T_0 by Proposition 5.5.5. The removal of a pair $(s_{i'}, p_j)$ at T_2 can only have happened if l_k was full, non-precarious (since we are removing a non-precarious pair)

and l_k meta-preferred the student they are adding, to $s_{i'}$. But then $s_{i'} \in M(p_j)$ was one of the worst students in $M(l_k)$ just before T_2, and so, by Proposition 5.5.8, p_j cannot subsequently be assigned a worse student.

Therefore, a student s_i worse than l_k's worst ranked assignees in $M(p_j)$ at T_0 cannot subsequently become assigned to p_j. □

In Lemma 5.5.23 we prove that it is not possible for a component of the type shown in Figure 5.7f to exist in G'.

Lemma 5.5.23. *Let M be a stable matching found by Algorithm 5.2 for instance I of* SPA-ST*, and let M_{opt} be a maximum stable matching in I. No component of the type given in Figure 5.7f can exist in the mapped graph G'.*

Proof. Assume for contradiction that there is a component c of the type shown in Figure 5.7f in G'. Let p_j be the project assigned to $s^{c,1}$ in M'_{opt} in Figure 5.7f.

We look at the possible configurations in G that could map to c in G'. Lecturer clones $l^{c,1}$ and $l^{c,2}$ may or may not be the same lecturer in G. It may also be the case that $s^{c,1}$ and $s^{c,2}$ are assigned to the same or different projects in M' and M'_{opt}, respecting the fact that projects may only be offered by one lecturer. Let $s^{c,1} = s_i$ and $s^{c,2} = s_{i'}$. Figure 5.8 shows the possible configurations in G relating to c in G'. They are found by noting that all configurations in G must have: 2 students; 1 or 2 lecturers; between 2 and 3 projects; student $s_{i'}$ must be unassigned in M; and $s_{i'}$ must be assigned a project of the same lecturer in M_{opt} as s_i is in M. Note that it is not possible for there to be only one project p_j in the configuration since s_i would be assigned to p_j in both M and M'_{opt}, meaning (s_i, p_j) would not exist in $M_{opt} \backslash M$ or $M \backslash M_{opt}$ and so neither of the edges from $s^{c,1}$ would exist in G', a contradiction.

We now show that none of the subgraphs shown in Figure 5.8 can occur in a matching M with respect to G found using Algorithm 5.2. Assume for contradiction that one does occur. We consider each type of subgraph separately.

(a) Students s_i and $s_{i'}$ are assigned to p_j and $p_{j'}$ in M_{opt} respectively, and s_i is assigned to $p_{j'}$ in M. Lecturer l_k offers both p_j and $p_{j'}$.

There are three sub-cases to consider.

i. *s_i strictly prefers p_j to $p_{j'}$:* If s_i strictly prefers p_j to $M(s_i)$ then s_i must have applied to p_j at least once. But this contradicts Proposition 5.5.20.

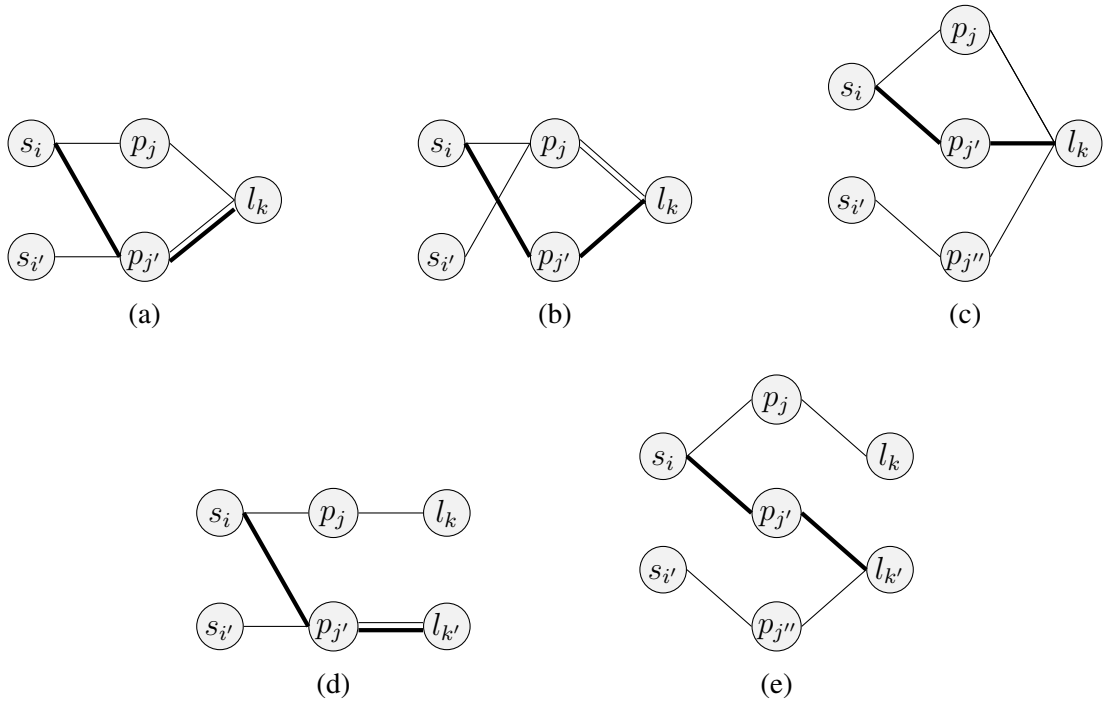


Figure 5.8: Possible configurations in G for an alternating path of size 3 in G' with M'_{opt} end edges. M and M_{opt} are shown in bold and non-bold edges respectively. Any project and lecturer vertices shown may have additional assignments involving other vertices not shown in the graphs.

ii. *p_j and $p_{j'}$ are tied on s_i's preference list:* Project p_j is fully available in the finalised matching M by Proposition 5.5.20 and has always been fully available by Proposition 5.5.3. As there is a fully available project tied with $p_{j'}$ on s_i's list, once edge $(s_i, p_{j'})$ is added to M, as long as it remains, it must be precarious. Pair $(s_i, p_{j'})$ cannot be removed at any stage before the end of the main while loop, since doing so would mean s_i would apply to p_j before again applying to $p_{j'}$ (since p_j is fully available) contradicting Proposition 5.5.20. Also Algorithm 5.4 cannot change the allocations of any precarious lecturer by Proposition 5.5.6. Therefore Algorithm 5.2 must terminate with $(s_i, p_{j'})$ as a precarious pair.

Student $s_{i'}$ must have applied to $p_{j'}$ in phase 2 since they are unassigned in the finalised matching M. By Proposition 5.5.21, at the point of application, $p_{j'}$ is not fully available and is non-precarious. But by Proposition 5.5.5 $p_{j'}$ cannot subsequently become precarious and so the algorithm will terminate with a non-precarious $p_{j'}$, contradicting the above.

iii. *s_i strictly prefers $p_{j'}$ to p_j:* We consider three further sub-cases based on l_k's preference list.

1. *l_k strictly prefers s_i to $s_{i'}$:* We know that s_i strictly prefers $p_{j'}$ to p_j and that l_k strictly prefers s_i to $s_{i'}$. But then $(s_i, p_{j'})$ forms a blocking pair of stable

matching M_{opt}, a contradiction.

2. *$s_{i'}$ and s_i are tied on l_k's preference list:* Project p_j is fully available in M by Proposition 5.5.20 and has always been fully available by Proposition 5.5.3. Student s_i must have been assigned to $p_{j'}$ in phase 1, otherwise s_i would have applied to p_j, contradicting Proposition 5.5.20. Student $s_{i'}$ must have applied to $p_{j'}$ in phase 2 since $s_{i'}$ is unassigned in M. Denote the point at which $s_{i'}$ applies to $p_{j'}$ in phase 2 as T_0. By Proposition 5.5.21, at T_0, $p_{j'}$ is not fully available and is non-precarious. By Proposition 5.5.5, $p_{j'}$ remains non-precarious from time T_0 until the algorithm's termination. Regardless of whether $(s_i, p_{j'})$ exists in the matching at time T_0, we know from time T_0, $(s_i, p_{j'})$ cannot be removed from M, otherwise s_i would remove $p_{j'}$ from their preference list (as $p_{j'}$ is non-precarious) contradicting the fact that s_i assigned to $p_{j'}$ in phase 1.
   We consider the following two possibilities.

   - *$s_{i'}$ applied to $p_{j'}$ in phase 2 before pair $(s_i, p_{j'})$ was added:* Assume first that $s_{i'}$ is unsuccessful in its application at T_0. From above we know that $p_{j'}$ remains non-precarious from T_0 onwards.
     - If $p_{j'}$ is undersubscribed at time T_0, then l_k cannot be precarious (as $s_{i'}$ was rejected) and so l_k does not meta-prefer $s_{i'}$ to its worst assignee $s_w \in M(l_k)$ at T_0. Lecturer l_k must be full at this point since $p_{j'}$ is not fully available and is undersubscribed. As l_k is full and must remain non-precarious by Proposition 5.5.5, after T_0, it is only possible for l_k to improve their allocations, by Proposition 5.5.8. Since l_k meta-prefers $s_{i'}$ to s_i ($s_{i'}$ is in phase 2), when s_i applies to $p_{j'}$, s_i must also be rejected. Project $p_{j'}$ is non-precarious after T_0 and so s_i must remove $p_{j'}$ from their list, contradicting the fact that s_i must be assigned to $p_{j'}$ in phase 1.
     - If $p_{j'}$ is full at T_0 then since it is also non-precarious, we know l_k does not meta-prefer $s_{i'}$ to its worst assignee s_w in $M(p_{j'})$. By Proposition 5.5.8, $p_{j'}$ cannot accept assignments that are worse than or equal to the worst assignee in $M(p_{j'})$ until after the main while loop. Therefore when s_i applies to $p_{j'}$ before the main while loop, as in the previous case, they must also be rejected, a contradiction as above.

     Assume therefore that $s_{i'}$ is successful in their application at T_0. Pair $(s_{i'}, p_{j'})$ must be removed at some point after T_0 since $(s_{i'}, p_{j'}) \notin M$. Denote the time $(s_{i'}, p_{j'})$ is removed as T_1. The removal at T_1 must have occurred before the end of the main while loop since otherwise $s_{i'}$ would be assigned to some project in the finalised matching M (the

same students are assigned when removing blocking pairs of type $(3bi)$), which it is not. We know that $(s_i, p_{j'})$ was added either after T_0 and before T_1 or after T_1. Once added $(s_i, p_{j'})$ cannot be removed from above.

- Assume $(s_i, p_{j'})$ was added before T_1. At T_1 (before the end of the main while loop) pair $(s_{i'}, p_{j'})$ is removed. This must either be because $p_{j'}$ is undersubscribed and l_k is full, or because $p_{j'}$ is full.
  - If the former then l_k is full and cannot be precarious since we are removing a non-precarious pair ($p_{j'}$ is non-precarious after T_0). But this removal can only happen if $s_{i'}$ is the worst student assigned to l_k at T_1. But by the definition of a worst assignee s_i (being in phase 1) would be removed before $s_{i'}$. Therefore, $(s_i, p_{j'})$ must have already been removed from the matching, a contradiction to the fact that $(s_i, p_{j'})$ cannot be removed.
  - Using similar reasoning, if $p_{j'}$ is full at T_1, then (as we are removing a non-precarious pair), $s_{i'}$ must be the worst student assigned in $M(p_{j'})$. But this would mean $(s_i, p_{j'})$ had already been removed, a contradiction.
- Assume $(s_i, p_{j'})$ was added after T_1. Again we consider two subcases.
  - If $p_{j'}$ was undersubscribed at time T_1, then l_k must have been full and must be non-precarious since non-precarious pair $(s_{i'}, p_{j'})$ was removed. This pair was removed as $s_{i'}$ was a worst student in l_k at T_1. By Proposition 5.5.5, l_k remains non-precarious from this point onwards and therefore by Proposition 5.5.8, l_k can only improve their allocations from time T_1. Therefore, as l_k meta-prefers $s_{i'}$ to s_i (as $s_{i'}$ is in phase 2), it must be that s_i, applying after T_1, will be rejected. This would result in the removal of $p_{j'}$ from s_i's list contradicting the fact that s_i assigned to $p_{j'}$ in phase 1.
  - If $p_{j'}$ is full at T_1 then using similar reasoning to above, we know that at T_1, $s_{i'}$ is a worst assignee in $M(p_{j'})$. Since, at T_1, $p_{j'}$ is full and non-precarious, and remains non-precarious, by Proposition 5.5.8, $p_{j'}$ cannot subsequently accept assignments that are worse than or equal to the worst assignee in $M(p_{j'})$ until the end of the main while loop. Therefore s_i will be rejected on application, a contradiction as above.

- *$s_{i'}$ applied to $p_{j'}$ in phase 2 after pair $(s_i, p_{j'})$ was added:* At T_0, since l_k meta-prefers $s_{i'}$ to s_i (s_i is in phase 1 whereas $s_{i'}$ is in phase 2), $(s_{i'}, p_{j'})$

must be added to the matching with some student other than $s_{i'}$ being removed, since s_i cannot be removed from M from time T_0. But now we are in the same position as before where $(s_{i'}, p_{j'})$ must be removed from M, but this can only happen if $(s_i, p_{j'})$ is removed first, a contradiction.

3. l_k *strictly prefers* $s_{i'}$ *to* s_i*:* Since $s_{i'}$ is unassigned in M, $(s_{i'}, p_{j'})$ is a blocking pair of stable matching M, a contradiction.

(b) Students s_i and $s_{i'}$ are both assigned to the same project p_j in M_{opt} and s_i is assigned to project $p_{j'}$ in M. Both p_j and $p_{j'}$ are offered by lecturer l_k. Since $s_{i'}$ is unassigned in M, we know that $s_{i'}$ has to have applied to p_j during the algorithm's execution, but this directly contradicts Proposition 5.5.20.

(c) Students s_i and $s_{i'}$ are assigned to p_j and $p_{j''}$ in M_{opt} respectively, and s_i is assigned to $p_{j'}$ in M. Lecturer l_k offers p_j, $p_{j'}$ and $p_{j''}$. By Proposition 5.5.18, $p_{j''}$ is undersubscribed in M since lecturer clone $l^{c,2}$ in Figure 5.7f is connected to an edge in M'_{opt} corresponding to $p_{j''}$ and an edge in M' corresponding to $p_{j'}$. We know that l_k is undersubscribed in M since $l^{c,1}$ is not assigned in M', and so $p_{j''}$ must be fully available. By Proposition 5.5.3 we know that $p_{j''}$ has always been fully available during the algorithm's execution. Since $s_{i'}$ is unassigned in M they must have applied to $p_{j''}$ during the course of the algorithm. But as $p_{j''}$ has always been fully available this must have been accepted. As we end up with $s_{i'}$ being unassigned it must also be the case that l_k rejects pair $(s_{i'}, p_{j''})$ but we know that $p_{j''}$ is always fully available and so this cannot have happened, a contradiction.

(d) Students s_i and $s_{i'}$ are assigned to p_j and $p_{j'}$ in M_{opt} respectively, and s_i is assigned to $p_{j'}$ in M. Lecturers l_k and $l_{k'}$ offer projects p_j and $p_{j'}$, respectively. Identical arguments to those found in Case (a)i. and (a)ii. can be used to show a contradiction. Similarly, identical arguments to those found in Case (a)iii. can also be used to show a contradiction, but exchanging l_k for $l_{k'}$.

(e) Students s_i and $s_{i'}$ are assigned to p_j and $p_{j''}$ in M_{opt} respectively, and s_i is assigned to $p_{j'}$ in M. Lecturer l_k offers project p_j, whereas $l_{k'}$ offers projects $p_{j'}$ and $p_{j''}$. We consider the following 3 sub-cases.

i. s_i *strictly prefers* p_j *to* $p_{j'}$*:* Identical arguments to those found in Case (a)i. can be used to show a contradiction.

ii. p_j *and* $p_{j'}$ *are tied on* s_i*'s preference list:* Using similar reasoning to Case (a)ii. we know that once edge $(s_i, p_{j'})$ is added to M it is, and remains, a precarious pair and cannot be removed at any stage. Also $p_{j'}$ must have been fully available on application by s_i otherwise fully available p_j at the same rank would have been

applied to by s_i. Therefore $p_{j'}$ is either fully available or precarious throughout the algorithm's execution. Student $s_{i'}$ is not assigned in M and so must have applied to $p_{j''}$ whilst they were in phase 2. Let this time of application be denoted T_0. By Proposition 5.5.21, $p_{j''}$ cannot be precarious at T_0.

- If $p_{j'}$ is fully available at T_0 then $l_{k'}$ is undersubscribed and so $s_{i'}$ would only be rejected if $p_{j''}$ was non-precarious, full and $s_{i'}$ was not meta-preferred by $l_{k'}$ to an student in $M(p_{j''})$.
- If $p_{j'}$ is precarious at T_0 then $l_{k'}$ is also precarious and therefore $s_{i'}$ would again only be rejected if $p_{j''}$ was non-precarious, full and $s_{i'}$ was not meta-preferred by $l_{k'}$ to an student in $M(p_{j''})$.

Therefore we have the following two cases.

1. If $s_{i'}$ was rejected then it must be because $p_{j''}$ was non-precarious, full and $s_{i'}$ was not meta-preferred by $l_{k'}$ to any student in $M(p_{j''})$, by above. But similar to Case (c) we can say that by Proposition 5.5.18, $p_{j''}$ is undersubscribed in the finalised matching M. Therefore, at least one non-precarious pair involved with $p_{j''}$ must be removed (without a pair involving $p_{j''}$ immediately replacing it) before the end of the algorithm. Denote this point in the algorithm's execution as T_1 and the removed pair $(s_{i''}, p_{j''})$ for some student $s_{i''}$. Note T_1 may either be before of after the end of the main while loop. This type of removal can only happen when $l_{k'}$ is full (this is clear before the main while loop, and is true after the main while loop by Proposition 5.5.2), and once a lecturer is full they remain full (since any pair deletion involving a project of $l_{k'}$ can only occur with a pair addition involving a project of $l_{k'}$). But $(s_i, p_{j'})$ was assigned when $p_{j'}$ was fully available and so $(s_i, p_{j'})$ was assigned before T_1. If T_1 occurs after the end of the main while loop then $l_{k'}$ must be non-precarious at T_1 by Proposition 5.5.6. But this means before T_1, pair $(s_i, p_{j'})$ is either in the matching but no longer precarious, or has been removed from the matching, a contradiction to the fact that $(s_i, p_{j'})$, once added, remains a precarious pair that can never be removed. Therefore the removal of $(s_{i''}, p_{j''})$ at T_1 must have occurred before the end of the main while loop. But, since $(s_i, p_{j'})$ is precarious, it would be removed before non-precarious $(s_{i''}, p_{j''})$, a contradiction.
2. If $s_{i'}$ was accepted then pair $(s_{i'}, p_{j''})$ would need to be removed before the algorithm terminated (since $(s_{i'}, p_{j''}) \notin M$). We know from before that $p_{j''}$ is non-precarious at the point of application and further that pair $(s_{i'}, p_{j''})$ must remain non-precarious by definition since $s_{i'}$ applied in phase 2. Therefore, we need to remove non-precarious pair $(s_{i'}, p_{j''})$ from the matching which can only happen if either $p_{j''}$ is full or $l_{k'}$ is full. Firstly assume that $p_{j''}$

is full and pair $(s_{i'}, p_{j''})$ is replaced with a meta-preferred student assigned to $p_{j''}$ ($p_{j''}$ must be non-precarious since we are removing a non-precarious pair). Since $p_{j''}$ needs to be undersubscribed in the finalised matching M we are in the same position and contradiction as the previous case. Secondly, $l_{k'}$ is full (and therefore remains full), in which case pair $(s_i, p_{j'})$ must already be in M (since it was added to M when $p_{j'}$ was fully available) and we can use a similar reasoning to the latter half of the previous case to show a contradiction.

iii. *s_i strictly prefers $p_{j'}$ to p_j:* We now consider three sub-cases based on $l_{k'}$'s preference list.

1. *$l_{k'}$ strictly prefers s_i to $s_{i'}$:* We know that $p_{j'}$ is undersubscribed in M_{opt} by Proposition 5.5.19. Either $l_{k'}$ is undersubscribed (in which case $p_{j'}$ is fully available), or $l_{k'}$ is full (and strictly prefers s_i to $s_{i'}$). In either case, $(s_i, p_{j'})$ is a blocking pair of stable M_{opt}, a contradiction.
2. *$s_{i'}$ and s_i are tied on $l_{k'}$'s preference list:* For this initial paragraph we use some similar reasoning to Case (a)iii.2.. Student s_i must have assigned to $p_{j'}$ in phase 1, otherwise s_i would have applied to p_j a contradiction to Proposition 5.5.20. Unlike Case (a)iii.2., $p_{j'}$ may be precarious at this point of application. Also, student $s_{i'}$, not being assigned in M, must have applied to $p_{j''}$ whilst in phase 2. Denote the point at which $s_{i'}$ applies to $p_{j''}$ in phase 2 as T_0. At T_0 we know that $p_{j''}$ is not fully available and is non-precarious by Proposition 5.5.21 and that $p_{j''}$ remains non-precarious from this point onwards by Proposition 5.5.5.

   We look at two possibilities:

   A. *$s_{i'}$ was rejected at T_0:* There would be two possible reasons for the rejection. Firstly, that $l_{k'}$ is non-precarious, full and $l_{k'}$ does not meta-prefer $s_{i'}$ to any student in $M(l_{k'})$. Secondly, that $p_{j''}$ is non-precarious, full and $l_{k'}$ does not meta-prefer $s_{i'}$ to any student in $M(p_{j''})$. We can rule out the first option as follows. If $l_{k'}$ is non-precarious and full at T_0 then by Proposition 5.5.8, $l_{k'}$ cannot accept a worse student than currently exists in $M(l_{k'})$ for the remainder of the algorithm. Since $s_{i'}$ was rejected we can conclude that no worse student than $s_{i'}$ can exist in $M(l_{k'})$ at T_0 and cannot exist in $M(l_{k'})$ from T_0 onwards. But $s_{i'}$ being in phase 2 is meta-preferred to s_i in phase 1. This contradicts the fact that $(s_i, p_{j'}) \in M$. Therefore, $s_{i'}$ was rejected because $p_{j''}$ is non-precarious, full and $l_{k'}$ does not meta-prefer $s_{i'}$ to any student in $M(p_{j''})$.

   Using a similar strategy to Case (e)ii., we know that $p_{j''}$ is undersubscribed in the finalised matching M by Proposition 5.5.18, therefore

before the algorithm terminates a pair $(s_{i''}, p_{j''})$ involving $p_{j''}$ must be removed without being immediately replaced with another pair involving $p_{j''}$. Denote the first such occurrence as happening at time T_1, where T_1 occurs after T_0. Note that T_1 may be either before or after the end of the main while loop.

- Assume T_1 occurs before the end of the main while loop. We know any removal of the type occurring at T_1 must be due to $l_{k'}$ being full and non-precarious (since $(s_{i''}, p_{j''})$ is removed as a non-precarious pair). By Proposition 5.5.8, we know $l_{k'}$ cannot subsequently be assigned in M a student worse than or equal to a worst student in $M(l_{k'})$ at T_1. Using the same proposition we know that $p_{j''}$ cannot be assigned in M a worse student until the end of the main while loop than exists in the matching at T_0.

  Since a pair involving $p_{j''}$ was removed at T_1, a worst assignee in $M(l_{k'})$ at T_1 can be no worse than a worst assignee in $M(p_{j''})$ at T_0. Finally, this means that no student assigned to $l_{k'}$ from T_1 onwards can be worse than $s_{i'}$ rejected at T_0, but s_i being in phase 1 is worse than $s_{i'}$ in phase 2 according to $l_{k'}$, a contradiction to the fact that $(s_i, p_{j'}) \in M$.
- Assume therefore that T_1 occurs after the end of the main while loop. Then s_i has to be assigned to $p_{j'}$ at this point. Since $p_{j''}$ becomes undersubscribed at T_1, pair $(s_{i''}, p_{j'''})$ must be a blocking pair of type $(3bi)$, where $p_{j'''}$ is an undersubscribed project of $l_{k'}$, and s_i strictly prefers $p_{j'''}$ to $p_{j''}$. By Propositions 5.5.10 and 5.5.11, $s_{i''}$ must be a worst student in $M(l_{k'})$ and therefore $M(p_{j''})$ at T_1. But, we also know that at T_0 when $s_{i'}$ was rejected, $l_{k'}$ could not subsequently accept a student to project $p_{j''}$ that is worse than a worst student existing in $M(p_{j''})$ for the remainder of the algorithm, by Proposition 5.5.22. Also, since $s_{i'}$ was rejected, no worse student can exist in $p_{j''}$ at T_0. Therefore, since $(s_{i''}, p_{j''})$ exists in M just before T_1, $s_{i''}$ cannot be worse than $s_{i'}$ according to $l_{k'}$ (and so must either be of equal rank and in phase 2 or of higher rank). Recall that $(s_i, p_{j'})$ was assigned in phase 1 and so must exist in the matching at T_1. This means that $s_{i''}$ is either at an equal rank to s_i (as the rank of s_i and $s_{i'}$ are equal) but is in phase 2 with s_i being in phase 1, or $s_{i''}$ is at a higher rank than s_i. In either case this contradicts the fact that $s_{i''}$ is a worst student in $M(l_{k'})$ at T_1.

B. *$s_{i'}$ application to $p_{j''}$ was accepted at T_0:* Pair $(s_{i'}, p_{j''})$ does not exist

in the finalised matching M and therefore must be removed sometime after T_0. We know $(s_{i'}, p_{j''})$ is always non-precarious by definition (as $s_{i'}$ applied in phase 2 and hence is removed as a non-precarious pair. Since $s_{i'}$ is unassigned in the finalised matching M and Algorithm 5.4 cannot change which students are assigned, $(s_{i'}, p_{j''})$ must be removed before the end of the main while loop. Denote this time as T_2.
This can only happen if either $l_{k'}$ is full and $s_{i'}$ is a worst assignee in $M(l_{k'})$ or $p_{j''}$ is full and $s_{i'}$ is a worst assignee in $M(p_{j''})$. If the former, then by Proposition 5.5.8, $l_{k'}$ cannot accept a worse student than a current worst student in $M(l_{k'})$. This worst student cannot be worse than $s_{i'}$, since $s_{i'}$ was just removed, hence s_i (of equal rank to $s_{i'}$ and in phase 1) cannot be assigned a project of $l_{k'}$'s in the finalised matching M, a contradiction. Therefore at T_2, $p_{j''}$ is full and $s_{i'}$ is removed as a worst assignee in $M(p_{j''})$.
But since $p_{j''}$ must be undersubscribed in the finalised matching M by Proposition 5.5.18, we can now use almost identical arguments as in case (e)iii.2.2.A to show a contradiction, noting that T_2 replaces T_0 and $s_{i'}$ was removed rather than rejected.

3. $l_{k'}$ *strictly prefers* $s_{i'}$ *to* s_i*:* Project $p_{j''}$ is undersubscribed in M by Proposition 5.5.18. Either $l_{k'}$ is undersubscribed (in which case $p_{j''}$ is fully available), or $l_{k'}$ is full (and strictly prefers $s_{i'}$ to s_i). In either case, $(s_{i'}, p_{j''})$ is a blocking pair of stable M, a contradiction.

Therefore it is not possible for a component structured as in Figure 5.7f to exist in G'. □

Similar to our previous lemma, we prove in Lemma 5.5.24 that it is not possible for a component of the type shown in Figure 5.7g to exist in G'.

Lemma 5.5.24. *Let M be a stable matching found by Algorithm 5.2 for instance I of* SPA-ST*, and let M_{opt} be a maximum stable matching in I. No component of the type given in Figure 5.7g can exist in the mapped graph G'.*

Proof. Let p_j be the project that student $s^{c,1}$ is assigned to in M_{opt} and let lecturer clone $l^{c,1}$ correspond to lecturer l_k in G. l_k must be undersubscribed in M as there is a lecturer clone $l^{c,1}$ unassigned in M'. Also, by Proposition 5.5.18, p_j must also be undersubscribed in M. Therefore, p_j is fully available at the end of the algorithm's execution and must always have been fully available by Proposition 5.5.3.

Student s_i is unassigned in M and so we know that s_i has to have applied to p_j during the algorithm's execution. Since p_j has always been fully available this had to have been

accepted. Now, (s_i, p_j) is not in the finalised matching M and so it must have been removed and this could only have happened if p_j or l_k were full. But this contradicts the fact that p_j has always been fully available. Therefore, no component of type (g) can exist in the mapped graph G'. □

Finally, Theorem 5.5.25 proves that Algorithm Max-SPA-ST-Approx is a $\frac{3}{2}$-approximation algorithm for MAX-SPA-ST.

Theorem 5.5.25. *Let M be a stable matching found by Algorithm 5.2 for instance I of* SPA-ST*, and let M_{opt} be a maximum stable matching in I. Then $|M| \geq \frac{2}{3}|M_{opt}|$.*

Proof. Let G' be the mapped graph constructed from the underlying graph of the instance G. Components in G' may only exist in the forms shown in Figure 5.7. Therefore we need only show that no component in G' can exist where the number of M' edges is less than two-thirds of the number of M'_{opt} edges. We run through each component of Figure 5.7 in turn. Let the current component be denoted c, where $M'(c)$ and $M'_{opt}(c)$ denote the set of edges in M' and M'_{opt} involved in c, respectively.

For Case 5.7a, an alternating cycle, and Cases 5.7b and 5.7c, alternating paths of even length, it is clear that $|M'(c)| = |M'_{opt}(c)|$. Case 5.7d involves an odd length alternating path with end edges in M'. It must be the case therefore that $|M'(c)| > |M'_{opt}(c)|$ for components of this type. Case 5.7e shows an odd length alternating path with end edges in M'_{opt}, but for path sizes greater than 5. Therefore, $|M'(c)| \geq \frac{2}{3}|M'_{opt}(c)|$ as required. Neither Case 5.7f nor 5.7g can exist in G' by Lemmas 5.5.23 and 5.5.24 respectively.

Hence it is not possible for the mapped graph G' to contain components in which $|M'(c)| < \frac{2}{3}|M'_{opt}(c)|$. Algorithm 5.2 is therefore a $\frac{3}{2}$-approximation algorithm for the problem of finding a maximum stable matching in I. □

5.5.6 Lower bound for the algorithm

Figure 5.9 shows instance I_5 of SPA-ST. A maximum stable matching M' in I is given by $M' = \{(s_1, p_2), (s_2, p_3), (s_3, p_1)\}$. The only possible blocking pairs for this matching are (s_3, p_3) and (s_3, p_2). However, neither pair can be a blocking pair since l_2 prefers both of their current assignees to s_3.

A trace is given as Table 5.3 which shows the execution run of Algorithm 5.2 over instance I_5. The algorithm outputs stable matching $M = \{(s_1, p_3), (s_3, p_2)\}$. The possible blocking pairs of this matching are (s_2, p_3) and (s_3, p_3). Neither can be a blocking pair since l_2 prefers s_1 to both s_2 and s_3.

Student preferences:	Project details:	Lecturer preferences:	
s_1: (p_3 p_2)	p_1: lecturer l_1, $c_1 = 2$	l_1: s_1 s_3	$d_1 = 2$
s_2: p_3	p_2: lecturer l_1, $c_2 = 1$	l_2: s_1 s_2 s_3	$d_2 = 1$
s_3: p_3 p_2 p_1	p_3: lecturer l_2, $c_3 = 1$		

Figure 5.9: SPA-ST instance I_5 in which Algorithm Max-SPA-ST-Approx finds a stable matching two-thirds of the size of optimal.

	Action	s_1	s_2	s_3
1	s_1 applies to p_3, accepted	p_3		
2	s_2 applies to p_3, accepted		p_3	
3	s_3 applies to p_3, rejected, s_3 removes p_3		p_3	
4	s_3 applies to p_2, accepted		p_3	p_2
5	s_1 applies to p_3, accepted, s_2 removes p_3	p_3		p_2
6	s_2 moves to phase 2	p_3		p_2
7	s_2 applies to p_3, rejected, s_2 removes p_3	p_3		p_2
8	s_2 moves to phase 3	p_3		p_2

Table 5.3: Trace of running Algorithm Max-SPA-ST-Approx for instance I_5 in Figure 5.9. In this table, the phrase "s_i removes p_j" indicates that student s_i removes project p_j from their preference list.

Therefore, Algorithm Max-SPA-ST-Approx has found a stable matching that is *exactly* two-thirds of the size of the maximum stable matching, thus the algorithm cannot guarantee a better bound than $\frac{3}{2}$.

We further note that this result holds for the arbitrarily large family of instances generated by copying instance I_5 a constant number of times, such that in each copy, indices of students and projects increase by 3 and indices of lecturers increase by 2.

5.6 Conclusions and future work

This chapter has described a $\frac{3}{2}$-approximation algorithm for MAX-SPA-ST. Furthermore, we gave an example to show that the $\frac{3}{2}$ bound is tight. It is of interest to determine how likely it is that such a worst-case example would arise in practice. In other words, can we expect the algorithm to produce stable matchings that are much closer to optimal in general? In Chapter 6, we present a new IP model for MAX-SPA-ST and use this to determine whether our $\frac{3}{2}$-approximation algorithm might in general produce matchings that are closer to optimal than the worst case bound of $\frac{3}{2}$. It remains open to describe an approximation algorithm that has a better performance guarantee, and/or to prove a stronger lower bound on the inapproximability of the problem than the current best bound of $\frac{33}{29}$ [74].

The work in this chapter has mainly focused on the size of stable matchings. However, it is

possible for a stable matching to admit a *blocking coalition*, where a permutation of student assignments could improve the allocations of the students and lecturers involved without harming anyone else. Since permutations of this kind cannot change the size of the matching they are not studied further here, but would be of interest for future work.

Chapter 6

Experiments and IP models for SPA-ST and lecturer load balancing for SPA-STL

6.1 Introduction

6.1.1 Background

In this chapter we continue our work in SPA-ST by first evaluating the performance of our $\frac{3}{2}$-approximation algorithm from Chapter 5 (Algorithm Max-SPA-ST-Approx), and second, investigating lecturer load balancing in SPA-STL. An introduction to SPA-ST was given in Section 2.5.2.2. The extension to SPA-ST, in which each lecturer has a target number of assignees, known as SPA-STL, was introduced in Section 2.5.2.3. In SPA-STL, it is desirable to find a matching that brings the number of assignees of each lecturer as close to their target as possible. There are several ways this may be done and three natural definitions are given below. Let I be an instance of SPA-STL.

- A matching M is *load-max-balanced* if the maximum absolute difference between lecturer targets and lecturer allocations is minimised over all matchings. We define the *load-max-balanced score* $r_m(M)$ as $\max_{l_k \in L} ||M(l_k)| - t_k|$. Thus a *load-max-balanced matching* is a matching M such that $r_m(M) = \min_{M' \in \mathcal{M}} \max_{l_k \in L} ||M(l_k)| - t_k|$.
- A matching M is *load-sum-balanced* if the sum of absolute differences between lecturer targets and lecturer allocations is minimised over all matchings. We define the *load-sum-balanced score* $r_s(M)$ as $\sum_{l_k \in L} ||M(l_k)| - t_k|$. Then a *load-sum-balanced matching* is a matching M such that $r_s(M) = \min_{M' \in \mathcal{M}} \sum_{l_k \in L} ||M(l_k)| - t_k|$.

Finally, a matching M is *load-balanced* if it is both load-max-balanced and load-sum-balanced.

6.1.2 Motivation

Some motivation for finding large stable matchings in SPA-ST was presented in Section 5.1.2 of Chapter 5. Recall that MAX-SPA-ST is the problem of finding a maximum stable matching in an instance of SPA-ST. We are interested in determining how large the matchings that are produced by the approximation algorithm (developed in Chapter 5) are, in practice. In particular, understanding whether the algorithm tends to produce a stable matching with size close to a maximum stable matching or close to either the $\frac{3}{2}$ bound or a minimum-sized stable matching (hereafter *minimum stable matching*) will help evaluate its performance.

To motivate our work in SPA-STL, we further discuss the example introduced in Section 5.1.2, which comprised students assigning to projects at a university. In this setting, each lecturer offers a range of projects. Students have preferences over projects, and lecturers have preferences over the students who rank their projects. We may wish to find a matching that provides fair lecturer allocations, so that the number of students assigned to each lecturer is as balanced as possible. This may be modelled by giving each lecturer a target number of assignees. Since teaching and administration responsibilities as well as the number of hours worked per week may differ among lecturers, it makes sense that each lecturer has an individual target that is calculated based on these factors. It is then desirable to find a matching that balances supervision loads fairly among lecturers by bringing each lecturer as close to their target as possible. Further motivation for studying load-balanced matchings as opposed to only load-max-balanced and load-sum-balanced matchings is presented in Section 6.3.2.

6.1.3 Contribution

In this chapter we present two new contributions relating to SPA-ST. Firstly, in Section 6.2, we evaluate our $\frac{3}{2}$-approximation algorithm for MAX-SPA-ST, by comparing output of this algorithm against a new IP model for MAX-SPA-ST, using randomly-generated data. We find that the performance of the approximation algorithm easily surpassed the $\frac{3}{2}$ bound, constructing a stable matching within 92% of optimal in all cases, with the percentage being far higher for many instances. Secondly, in Section 6.3, we consider lecturer load balancing in SPA-STL. We give polynomial-time algorithms to find optimal matchings of the three types of matching defined in Section 6.1.1. Additionally we show that when combined with stability, finding optimal matchings of these types becomes NP-hard. We additionally present an IP model to find a load balanced stable matching and prove its correctness.

6.1.4 Structure of the chapter

Section 6.2 presents the experiments evaluating the performance of the $\frac{3}{2}$-approximation algorithm for MAX-SPA-ST. Within this section, the IP model for MAX-SPA-ST and experimental evaluation are given in Sections 6.2.2 and 6.2.3 respectively. Section 6.3 describes the work relating to lecturer load balancing in SPA-STL. In Section 6.3.3 we present algorithms to find a load-balanced matching and a maximum-sized load-max-balanced matching (henceforth *maximum load-max-balanced matching*). Complexity results for finding an optimal stable matching with respect to various lecturer load balancing objectives are presented in Section 6.3.4. Section 6.3.5 gives an IP model for finding a load-balanced stable matching. Finally, conclusions and future work are discussed in Section 6.4.

6.2 IP model and experiments for SPA-ST

6.2.1 Introduction

In this section we present a new IP model for MAX-SPA-ST, and use it to evaluate the $\frac{3}{2}$-approximation algorithm for MAX-SPA-ST from Chapter 5. We additionally present experiments investigating how matching properties differ with changing instance parameter values such as instance size, probability of ties and preference list lengths.

6.2.2 IP model for MAX-SPA-ST

6.2.2.1 Stability definition

For the stability constraints in the model, it is advantageous to use an equivalent condition for stability, as given by the following lemma.

Lemma 6.2.1. *Let I be an instance of* SPA-ST *and let M be a matching in I. Then M is stable if and only if the following condition, referred to as condition (*) holds: For each student $s_i \in S$ and project $p_j \in P$, if s_i is unassigned in M and finds p_j acceptable, or s_i prefers p_j to $M(s_i)$, then either:*

- *l_k is full, $s_i \notin M(l_k)$ and l_k prefers the worst student in $M(l_k)$ to s_i or is indifferent between them, or;*
- *p_j is full and l_k prefers the worst student in $M(p_j)$ to s_i or is indifferent between them, where l_k is the lecturer offering p_j.*

Proof. Suppose M is stable. Assume for contradiction that condition (*) is not satisfied. Then there exists a student $s_i \in S$ and a project $p_j \in P$ such that s_i is unassigned in M and finds p_j acceptable, or s_i prefers p_j to $M(s_i)$, and one of the following four cases arises:

1. p_j and l_k are both undersubscribed;
2. p_j is undersubscribed, l_k is full and $s_i \in M(l_k)$;
3. p_j is undersubscribed, l_k is full and l_k prefers s_i to the worst student in $M(l_k)$;
4. p_j is full and l_k prefers s_i to the worst student in $M(p_j)$.

Each of these scenarios clearly describes a blocking pair as in Section 2.5.2.1, hence we have a contradiction to the stability of M.

Conversely, assume M satisfies condition (*). Suppose for contradiction that M is not stable. Then, there exists a blocking pair (s_i, p_j), implying that s_i is unassigned in M and finds p_j acceptable, or s_i prefers p_j to $M(s_i)$, and one of the above four cases will be true.

Whichever one of these cases holds, we then obtain a contradiction to the fact that the conditions given in the statement of the theorem holds. Thus M is stable. □

6.2.2.2 Description of variables and constraints

The key variables in the model are binary-valued variables x_{ij}, defined for each $s_i \in S$ and $p_j \in P$, where $x_{ij} = 1$ if and only if student s_i is assigned to project p_j. Additionally, we have binary-valued variables α_{ij} and β_{ij} for each $s_i \in S$ and $p_j \in P$. These variables allow us to more easily describe the stability constraints below. For each $s_i \in S$ and $l_k \in L$, let

$$T_{ik} = \{s_u \in S : \text{rank}(l_k, s_u) \leq \text{rank}(l_k, s_i) \wedge s_u \neq s_i\}.$$

That is, T_{ik} is the set of students ranked at least as highly as student s_i in lecturer l_k's preference list not including s_i. Also, for each $p_j \in P$, let

$$T_{ijk} = \{s_u \in S : \text{rank}(l_k, s_u) \leq \text{rank}(l_k, s_i) \wedge s_u \neq s_i \wedge p_j \in A(s_u)\}.$$

That is, T_{ijk} is the set of students s_u ranked at least as highly as student s_i in lecturer l_k's preference list, such that project p_j is acceptable to s_u, not including s_i. Finally, let $S_{ij} = \{p_r \in P : \text{rank}(s_i, p_r) \leq \text{rank}(s_i, p_j)\}$, that is, S_{ij} is the set of projects ranked at least as highly as project p_j in student s_i's preference list, including p_j. Figure 6.1 shows the IP model for MAX-SPA-ST.

maximise: $\sum_{s_i \in S} \sum_{p_j \in P} x_{ij}$

subject to:

1.	$x_{ij} \leq 0$	$\forall s_i \in S \ \forall p_j \in P,\ p_j \notin A(s_i)$
2.	$\sum_{p_j \in P} x_{ij} \leq 1$	$\forall s_i \in S$
3.	$\sum_{s_i \in S} x_{ij} \leq c_j$	$\forall p_j \in P$
4.	$\sum_{s_i \in S} \sum_{p_j \in P_k} x_{ij} \leq d_k$	$\forall l_k \in L$
5.	$1 - \sum_{p_r \in S_{ij}} x_{ir} \leq \alpha_{ij} + \beta_{ij}$	$\forall s_i \in S \ \forall p_j \in P$
6.	$\sum_{s_u \in T_{ik}} \sum_{p_r \in P_k} x_{ur} \geq d_k \alpha_{ij}$	$\forall s_i \in S \ \forall p_j \in P$
7.	$\sum_{s_u \in T_{ijk}} x_{uj} \geq c_j \beta_{ij}$	$\forall s_i \in S \ \forall p_j \in P$

$x_{ij} \in \{0,1\}$	$\forall s_i \in S \ \forall p_j \in P$
$\alpha_{ij} \in \{0,1\}$	$\forall s_i \in S \ \forall p_j \in P$
$\beta_{ij} \in \{0,1\}$	$\forall s_i \in S \ \forall p_j \in P$

Figure 6.1: IP model for MAX-SPA-ST.

Constraint 1 enforces $x_{ij} = 0$ if s_i finds p_j unacceptable (since $x_{ij} \in \{0, 1\}$). Constraint 2 ensures that a student may be assigned to a maximum of one project. Constraints 3 and 4 ensure that project and lecturer capacities are enforced. In the left hand side of the inequality of Constraint 5, if $1 - \sum_{p_r \in S_{ij}} x_{ir} = 1$, then either s_i is unassigned or s_i prefers p_j to $M(s_i)$. This ensures that if s_i is unassigned or s_i prefers p_j to $M(s_i)$ then either $\alpha_{ij} = 1$ or $\beta_{ij} = 1$ (or both), where α_{ij} and β_{ij} are described in Constraints 6 and 7. Constraint 6 ensures that, if $\alpha_{ij} = 1$, then the number of students assigned to l_k who are ranked at least as highly as student s_i by l_k (not including s_i) must be at least l_k's capacity d_k. Constraint 7 ensures that, if $\beta_{ij} = 1$, then the number of students assigned to p_j who are ranked at least as highly as student s_i by l_k (not including s_i) must be at least p_j's capacity c_j.

Finally, for our optimisation we maximise the sum of all x_{ij} variables in order to maximise the number of students assigned. The following section establishes the correctness of the IP model.

6.2.2.3 Proof of correctness

Theorem 6.2.2. *Given an instance I of* SPA-ST*, let J be the IP model as defined in Figure 6.1. A stable matching in I corresponds to a feasible solution in J and vice versa.*

Proof. Assume instance I of SPA-ST contains a stable matching M. We construct a feasible solution to J involving the variables $\mathbf{x}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ as follows.

The variables $\mathbf{x}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are constructed as follows. For each student $s_i \in S$ and for each project $p_j \in P$, if s_i is assigned to p_j in M then we set variable $x_{ij} = 1$, otherwise $x_{ij} = 0$. Let lecturer l_k be the proposer of project p_j. Let variable $\alpha_{ij} = 1$ if the following two conditions hold: *i)* student s_i is not assigned to lecturer l_k, and *ii)* lecturer l_k is full and prefers their worst ranked assignee to s_i, or is indifferent between them. Else let $\alpha_{ij} = 0$. Let variable $\beta_{ij} = 1$ if the following two conditions hold: *i)* student s_i is not assigned to project p_j, and *ii)* project p_j is full, and l_k prefers p_j's worst assignee to s_i, or is indifferent between them. Else, set $\beta_{ij} = 0$.

Now it must be shown that all constraints described in Figure 6.1 are satisfied.

1. *Constraints 1 - 4.* It is clear by the construction of J that Constraints 1-4 are satisfied.

2. *Constraint 5.* Recall $S_{ij} = \{p_r \in P : \text{rank}(s_i, p_r) \leq \text{rank}(s_i, p_j)\}$ is the set of projects ranked at least as highly as p_j in s_i's preference list. Let $\gamma_{ij} = 1 - \sum_{p_r \in S_{ij}} x_{ir}$. We must show that whenever $\gamma_{ij} = 1$, $\alpha_{ij} + \beta_{ij} \geq 1$. Assume $\gamma_{ij} = 1$, that is s_i is unassigned or would prefer to be assigned to p_j than to $M(p_j)$. As M is stable we know that condition (*) of Lemma 6.2.1 is satisfied. Therefore $\alpha_{ij} + \beta_{ij} \geq 1$ by construction. This directly satisfies Constraint 5.

3. *Constraint 6.* Recall that for student s_i and lecturer l_k, T_{ik} is the set of students ranked at least as highly as student s_i in lecturer l_k's preference list, not including s_i. Assume $\alpha_{ij} = 1$. Then, by definition, we know that lecturer l_k is full and prefers their worst ranked assignee to s_i, or is indifferent between them. Therefore, the LHS of the inequality must equal d_k and so this constraint is satisfied.

4. *Constraint 7.* Recall that T_{ijk} is the set of students ranked at least as highly as student s_i in lecturer l_k's preference list, such that the project p_j is acceptable to each student. Similar to above, assume $\beta_{ij} = 1$. Then, by definition, we know that project p_j is full and l_k prefers their worst ranked assignee in $M(p_j)$ to s_i, or is indifferent between them. Therefore, the LHS of the inequality must equal c_j and so this constraint is also satisfied.

We have shown that the assignment of values to $\mathbf{x}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ satisfy all the constraints in J, thus if there is a stable matching M in I, then there is a feasible solution of J.

Conversely, we now show that a feasible solution of J corresponds to a stable matching M in I. Let $\mathbf{x}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ be a feasible solution of J. For each x_{ij} variable in J, if $x_{ij} = 1$ then add pair (s_i, p_j) to M in I. It is now shown that this assignment of students to projects satisfies the definition of a stable matching M in I.

1. The following constraints are clearly satisfied by Constraints 1-4:

   - *A student s_i may be assigned to a maximum of 1 project;*
   - *A student s_i may only be assigned to a project that they find acceptable;*
   - *The number of students assigned to project p_j is less than or equal to c_j;*
   - *The number of students assigned to projects offered by lecturer l_k is less than or equal to d_k.*

2. *M is stable.* Assume for contradiction that there exists a blocking pair $(s_i, p_j) \in M$. Then by Lemma 6.2.1, neither of the sub-conditions of condition (*) can be true. Both of these sub-conditions being false imply that, as Constraints 6 and 7 must be satisfied, $\alpha_{ij} = 0$ and $\beta_{ij} = 0$.

   Recall $\gamma_{ij} = 1 - \sum_{p_r \in S_{ij}} x_{ir}$. $\sum_{p_r \in S_{ij}} x_{ir}$ is the number of projects that student s_i is assigned to at a higher or equal ranking than p_j in s_i's preference list (including p_j). Since (s_i, p_j) is a blocking pair, then it must be the case that $\sum_{p_r \in S_{ij}} x_{ir} = 0$. But this forces $\gamma_{ij} = 1$, and we know that α_{ij} and β_{ij} are equal to 0 so Constraint 5 is contradicted.

We have shown that if there is a feasible solution of $\mathbf{x}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ of J, then there is a stable matching M in I. This completes the proof. □

Corollary 6.2.3. *Given an instance I of* SPA-ST*, let J be the IP model as defined in Figure 6.1. A maximum stable matching in I corresponds to an optimal solution in J and vice versa.*

Proof. Assume M is a maximum stable matching in I. Let $f = \langle \mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta} \rangle$ be the solution in J constructed according to the description in Theorem 6.2.2. We must show that f forms an optimal solution of J. Firstly, since M is stable, we know by Theorem 6.2.2 that f is a feasible solution of J. Suppose for contradiction that f is not optimal. Then there is some solution $g = \langle \mathbf{x}', \boldsymbol{\alpha}', \boldsymbol{\beta}' \rangle$ of J in which $\text{obj}(g) > \text{obj}(f)$, where $\text{obj}(f')$ gives the objective value of f'. But by construction, g would translate into a stable matching M' such that $|M'| = \text{obj}(g) > \text{obj}(f) = |M|$ in I contradicting the fact that M is maximum.

Conversely, assume $f = \langle \mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta} \rangle$ is an optimal solution in J, and let M be the stable matching in I constructed according to the description in Theorem 6.2.2. Suppose for contradiction that there is some stable matching M' in I such that $|M'| > |M|$. Then by construction, there must be some corresponding solution $g = \langle \mathbf{x}', \boldsymbol{\alpha}', \boldsymbol{\beta}' \rangle$ of J such that $\text{obj}(g) = |M'| > |M| = \text{obj}(f)$, giving the required contradiction. □

6.2.3 Experimental evaluation

6.2.3.1 Methodology

Experiments were conducted on our $\frac{3}{2}$-approximation algorithm and an implementation of the IP model in Figure 6.1 using randomly-generated data, in order to measure the effects on matching properties when changing different parameter values (including instance size, probability of ties in preference lists and preference list lengths). Two further experiments explored the scalability of instance size and preference list lengths. Instances were generated using both existing and new software. The existing software is known as the *Matching Algorithm Toolkit* and is a collaborative project developed by students and staff at the University of Glasgow. A web application of the Matching Algorithm Toolkit, which allows access to many of its functions was developed by Lazarov [43].

For a given SPA-ST instance, let the total project and lecturer capacities be denoted by c_P and d_L, respectively. Capacities were distributed randomly over the sets of projects and lecturers, subject to there being a maximum difference of 1 between the capacities of any two projects or any two lecturers. The minimum and maximum size of student preference lists is given by l_{min} and l_{max}, and t_s represents the probability that a project on a student's preference list is tied with the next project. Lecturer preference lists were generated initially from the student preference lists, where a lecturer l_k must rank a student if a student ranks a project offered by l_k. These lists were randomly shuffled and t_l denotes the ties probability for lecturer preference lists. A linear distribution was used to make some projects more popular

than others and in all experiments the most popular project is around 5 times more popular than the least. This distribution influenced the likelihood of a student finding a given project acceptable. Parameter details for each experiment are given below.

1. *Increasing instance size:* 10 sets of $10,000$ instances were created (labelled SIZE1, ..., SIZE10). The number of students n_1 increased from 100 to 1000 in steps of 100, with $n_2 = 0.6n_1$, $n_3 = 0.4n_1$, $c_P = 1.4n_1$ and $d_L = 1.2n_1$. The probabilities of ties in preference lists were $t_s = t_l = 0.2$ throughout all instance sets. Student preference list lengths, bound by $l_{min} = 3$ and $l_{max} = 5$, also remained the same and were kept low to ensure a wide variability in stable matching size per instance.

2. *Increasing probability of ties:* 11 sets of $10,000$ instances were created (labelled TIES1, ..., TIES11). Throughout all instance sets $n_1 = 300$, $n_2 = 250$, $n_3 = 120$, $c_P = 420$, $d_L = 360$, $l_{min} = 3$ and $l_{max} = 5$. The probabilities of ties in student and lecturer preference lists increased from $t_s = t_l = 0.0$ to $t_s = t_l = 0.5$ in steps of 0.05.

3. *Increasing preference list lengths:* 10 sets of $10,000$ instances were generated (labelled PREF1, ..., PREF10). Similar to the TIES cases, throughout all instance sets $n_1 = 300$, $n_2 = 250$, $n_3 = 120$, $c_P = 420$ and $d_L = 360$. Additionally, $t_s = t_l = 0.2$. Student preference list lengths increased from $l_{min} = l_{max} = 1$ to $l_{min} = l_{max} = 10$ in steps of 1.

4. *Instance size scalability:* 5 sets of 10 instances were generated (labelled SCALS1, ..., SCALS5). All instance sets in this experiment used the same parameter values as the SIZE experiment, except the number of students n_1 increased from $10,000$ to $50,000$ in steps of $10,000$.

5. *Preference list scalability:* Finally, 6 sets of 10 instances were created (labelled SCALP1, ..., SCALP6). Throughout all instance sets $n_1 = 500$ with the same values for other parameters as in the SIZE experiment. However in this case ties were fixed at $t_s = t_l = 0.4$, and student preference list lengths increased from $l_{min} = l_{max} = 25$ to $l_{min} = l_{max} = 150$ in steps of 25.

For each generated instance, we ran our $\frac{3}{2}$-approximation algorithm and then used the IP model to find a maximum stable matching. Additionally, we computed a minimum stable matching using a simple adaptation of our IP model for MAX-SPA-ST. A timeout of 1800 seconds (30 minutes) was imposed on each algorithm, for each instance run, with all instances of Experiments 1, 2 and 3 completing within the timeout time. All experiments were conducted on the machine described in Chapter 1. The operating system was Ubuntu version 17.04 with all code compiled in Java version 1.8, where the IP models were solved using Gurobi

version 7.5.2. Each approximation algorithm instance was run on a single thread while each IP instance was run on two threads. No attempt was made to parallelise Java garbage collection. The statistics collection program, and plot and table generation program, were written in Python and run using Python version 3.6.1. Repositories for the software and data used in these experiments can be found at `https://doi.org/10.5281/zenodo.1183221` and `https://doi.org/10.5281/zenodo.1186823` respectively.

Correctness testing was conducted over all generated instances. This consisted of checking that the matchings produced by the approximation and IP-based algorithms adhered to (1) *capacity*: each student is assigned to a maximum of 1 project, and each project and lecturer is not assigned to more students than their given capacity; and (2) *stability*: no blocking pair exists. Additionally, it was checked that each matching produced by the approximation algorithm was at least two-thirds the size of a maximum stable matching. All correctness tests passed successfully.

6.2.3.2 Experimental results

Experimental results can be seen in Figures 6.2, 6.3 and 6.4, and in Tables D.1, D.2, D.3, D.4, D.5, D.6 and D.7 of Appendix D.

Figures 6.2, 6.3 and 6.4 (with associated Tables D.1, D.2 and D.3) show plots comparing the size of matching returned by the approximation algorithm with the sizes of minimum and maximum stable matchings and a $\frac{3}{2}$ bound, for Experiments 1, 2 and 3 respectively. In each of these Figures, the median values of the size of stable matchings are plotted and a 90% confidence interval is displayed using the 5th and 95th percentile measurements.

Tables D.4, D.5 and D.6 show additional results for Experiments 1, 2 and 3 respectively. From this point onwards an *optimal* matching refers to a maximum stable matching. In these tables, 'A' represents statistics relating to the approximation algorithm, and 'Min' and 'Max' represent statistics relating to the IP models to find a minimum and maximum stable matching, respectively. Column 'Minimum A/Max' gives the minimum ratio of approximation algorithm matching size to optimal matching size that occurred, '% A=Max' displays the percentage of times the approximation algorithm achieved an optimal result, and '% A≥ 0.98Max' shows the percentage of times the approximation algorithm achieved a result at least 98% of optimal. The 'Mean size' columns are somewhat self explanatory, with subcolumns 'A/Max' and 'Min/Max' showing the mean approximation algorithm matching size and minimum stable matching size as a fraction of optimal. Finally, the mean time taken in milliseconds to run each algorithm per instance is given in the last three columns. Table D.7 shows the scalability results for increasing instance sizes (Experiment 4) and increasing preference list lengths (Experiment 5). The 'Instances completed' column indicates the number of instances that completed before timeout occurred. The mean time taken is also shown,

where instances that did not complete within the timeout time were said to have taken the maximum time of 30 minutes.

The main findings are summarised below.

- *The approximation algorithm consistently far exceeds its $\frac{3}{2}$ bound.* Considering the column labelled 'Minimum A/Max' in Tables D.4, D.5 and D.6, we see that the smallest value was within the SIZE1 instance set with a ratio of 0.9286. This is well above the required bound of $\frac{2}{3}$.

- *Approximation algorithm matchings are closer in size to the maximum stable matchings than to the minimum stable matchings.* The columns 'A/Max' and 'Min/Max' of Tables D.4, D.5 and D.6 show that, on average, for each instance set, the approximation algorithm produces a solution that is within 98% of maximum and far closer to the maximum size than to the minimum size. This may also be seen each of the Figures 6.2, 6.3 and 6.4, where in general, the size of a stable matching output by the approximation algorithm, indicated in blue, is far closer to the size of a maximum stable matching (red) than the size of a minimum stable matching (green), in all cases.

- *Divergence in sizes of minimum and maximum stable matchings when increasing the probability of ties.* Unlike in Figures 6.2 and 6.4 which show similar trends for mean matching sizes produced by the approximation algorithm, and the minimum and maximum sized stable matchings, Figure 6.3 shows a marked divergence of the minimum stable matching size as the probability of ties increases. With no ties in the preference lists, all matchings are the same size (namely 284), as expected from the theory. As the probability of ties increases, the maximum stable matching size and size of matching from the approximation algorithm increase steadily to 294.8 and 299.5 respectively. However, the minimum stable matching size decreases steadily to 254.8. This behaviour may be explained by noticing that the higher the probability of ties, the larger the number of stable matchings that will exist. Hence there is a higher probability that the sizes of minimum and maximum stable matchings will diverge. Interestingly however, the decrease in size of minimum stable matchings does not have a noticeable effect on the approximation algorithm output, as matching sizes from this algorithm closely align with optimal.

- In Table D.7, Experiment 4 (SCALS) shows the number of instances solved within the 30-minute timeout reduced from 10 to 0 for the IP-based algorithm for MAX-SPA-ST. However, even for the largest instance set sizes the approximation algorithm was able to solve each instance on average within 21 seconds. For Experiment 5 (SCALP), with a higher probability of ties and increasing preference list lengths, the IP-based algorithm to find a maximum stable matching was only able to solve all the instances of one

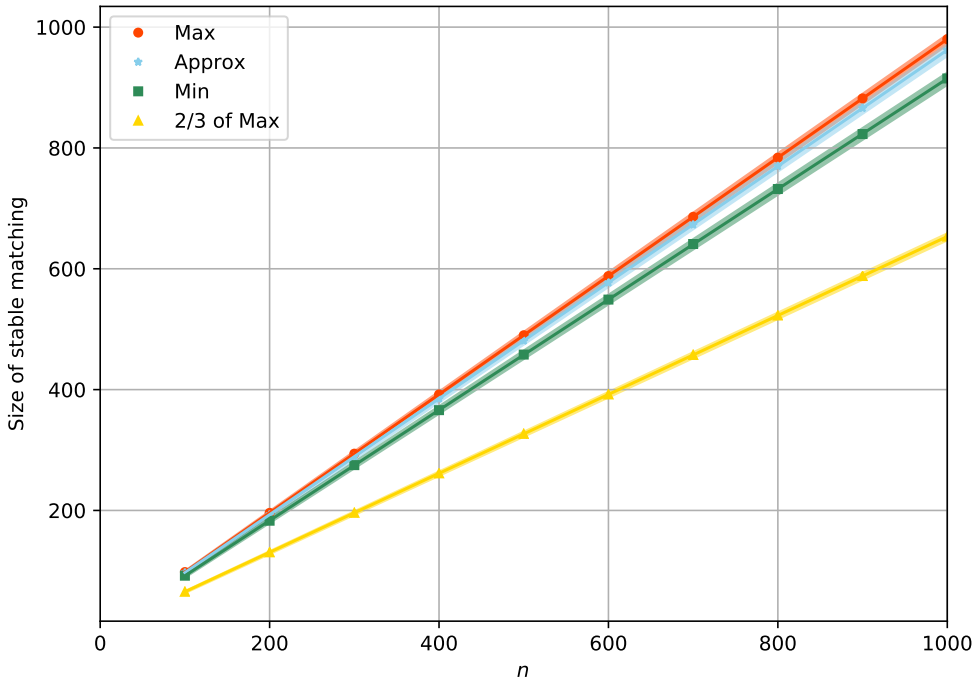


Figure 6.2: Plot of the size of stable matching returned by the approximation algorithm, and the minimum and maximum stable matching sizes, with increasing n, where n is the number of men or women. A second-order polynomial has been assumed for all best-fit lines.

instance set (SCALP2) within 30 minutes each, however the approximation algorithm took less than 0.3 seconds on average to return a solution for each instance. This shows that the approximation algorithm is useful for either larger or more complex instances than the IP-based algorithm can handle, motivating its use for real world scenarios.

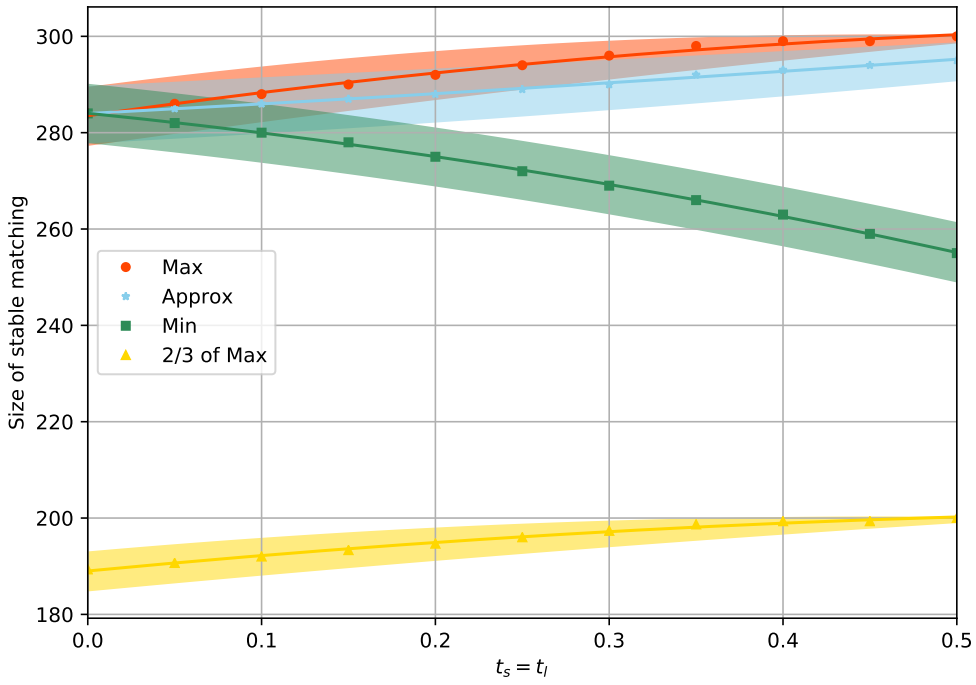


Figure 6.3: Plot of the size of stable matching returned by the approximation algorithm, and the minimum and maximum stable matching sizes, with increasing probability of ties. A second-order polynomial has been assumed for all best-fit lines.

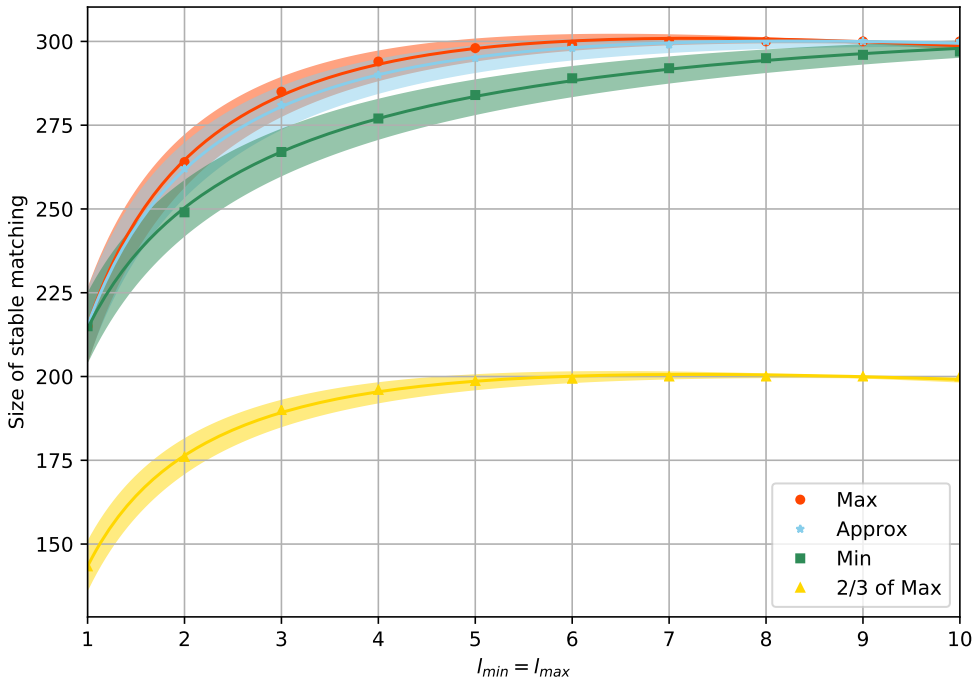


Figure 6.4: Plot of the size of stable matching returned by the approximation algorithm, and the minimum and maximum stable matching sizes, with increasing student preference list length. A second-order polynomial in $\log(l_{min})$ has been assumed for all best-fit lines.

6.3 Load balancing in SPA-STL

6.3.1 Introduction

We now move on to examining lecturer load-balancing in SPA-STL. The definitions of a load-max-balanced and load-sum-balanced matching were given in Section 6.1.1. Some motivation for studying lecture load balancing was given in Section 6.1.2 and we provide further motivation for studying load-balanced matchings (as opposed to only load-max-balanced or load-sum-balanced matchings) in the next section.

6.3.2 Motivation for studying load-balanced matchings

We now show, using examples, that a load-max-balanced matching may have an arbitrarily large load-sum-balanced score and vice versa. Let I_0 and I_1 be two instances of SPA-STL, such that all lecturers have a target and capacity of c, where c is even. For both instances, there are n_3 projects and lecturers, such that each lecturer supervises one project, where n_3 is also even. In I_0, let there be n_3c students, each of whom ranks every project, except for the project of lecturer l_1. Similarly, in I_1, let there be $\frac{n_3c}{2}$ students, each of whom ranks every project. In both instances, lecturers rank all students who rank them. Table 6.1 gives examples of lecturer allocations for load-max-balanced matchings in I_0, and load-sum-balanced matchings in I_1.

In I_0, it is not possible for lecturer l_1 to gain any allocations, and so the minimum load-max-balanced score over all matchings in $\mathcal{M}$ is c. The lecturer allocations for matchings M_1 and M_2 in I_0 are shown in Table 6.1a. Matching M_1 has zero allocations for all lecturers and matching M_2 has zero allocations for lecturer l_1, with all other lecturers reaching their targets. Hence, both M_1 and M_2 are load-max-balanced matchings. Despite this, M_1 has a significantly worse load-sum-balanced score of $r_s(M_1) = n_3c$ compared to that of M_2, which is $r_s(M_2) = c$.

In I_1, any matching M that allocates all $\frac{n_3c}{2}$ students to projects will be a load-sum-balanced matching. This is because, no matter how the students are distributed, the load-sum-balanced

Lecturer	$d_k = t_k$	$\lvert M_1(l_k)\rvert$	$\lvert M_2(l_k)\rvert$
l_1	c	0	0
l_2	c	0	c
...			
l_{n_3}	c	0	c
	$r_m(M)$:	c	c
	$r_s(M)$:	n_3c	c

(a) Possible load-max-balanced scores for two different matchings M_1 and M_2 in I_0.

Lecturer	$d_k = t_k$	$\lvert M_3(l_k)\rvert$	$\lvert M_4(l_k)\rvert$
l_1	c	c	$c/2$
...			
$l_{n_3/2}$	c	c	$c/2$
$l_{n_3/2+1}$	c	0	$c/2$
...			
l_{n_3}	c	0	$c/2$
	$r_m(M)$:	c	$c/2$
	$r_s(M)$:	$n_3c/2$	$n_3c/2$

(b) Possible load-sum-balanced scores for two different matchings M_3 and M_4 in I_1.

Table 6.1: Examples of lecturer allocations in I_0 and I_1.

score of such a matching will always be

$$\begin{aligned}\sum_{l_k \in L} (t_k - |M(l_k)|) &= \sum_{l_k \in L} t_k - \sum_{l_k \in L} |M(l_k)| \\ &= \sum_{l_k \in L} (t_k) - |M| \\ &\geq n_3 c - \frac{n_3 c}{2} \\ &= \frac{n_3 c}{2}.\end{aligned}$$

Table 6.1b shows the lecturer allocations for matchings M_3 and M_4 in I_1. Matching M_3 has c allocations for half of the lecturers and 0 allocations for the other half. Matching M_4 has $c/2$ allocations for all of the lecturers. Since all $\frac{n_3 c}{2}$ students are assigned in both matchings, M_3 and M_4 are load-sum-balanced. However, the load-max balanced score of M_3 is twice as large as the load-max-balanced score of M_4 ($r_m(M_3) = c$ compared to $r_m(M_4) = \frac{c}{2}$).

This motivates the study of a matching in which both notions of optimality are adhered to and is defined formally as follows. A matching M is *load-balanced* if it is both a load-max-balanced matching and a load-sum-balanced matching. Thus, a *load-balanced matching* is a matching M such that $r_m(M) = \min_{M' \in \mathcal{M}} \max_{l_k \in L} ||M(l_k)| - t_k|$ and $r_s(M) =$

$\min_{M' \in \mathcal{M}} \sum_{l_k \in L} ||M(l_k)| - t_k|$. We show in Section 6.3.3 that a load-balanced matching must exist, and present a polynomial-time algorithm for returning such a matching in an instance of SPA-STL. Additionally, we show that with a small change to the algorithm we are able to return a maximum load-max-balanced matching in polynomial time.

Note that in all of the load balancing optimality definitions, the position of a project on a student's preference list and the position of a student on a lecturer's preference list are irrelevant. However, in Section 6.3.4 we look at the complexity of finding optimal matchings of these types over the set of all stable matchings $\mathcal{M}_S$. Hence we define these objectives in the SPA-STL setting which includes ordered preference lists for both students and lecturers.

Due to the irrelevance of the preference list orderings to the results we present in Section 6.3.3, these results are also applicable to SPA-STL instances in which both students and lecturers have dichotomous preferences (i.e., students (lecturers) rank projects (students) as either *acceptable* or *unacceptable* and all acceptable agents are essentially tied). However, since the number of students allocated by these algorithms is not necessarily maximised, even in this reduced instance, it is not true to say that a resultant matching is optimal for students.

6.3.3 Load balancing algorithms

6.3.3.1 Introduction

In this section we first study the simpler problem of finding a load-sum-balanced matching. Then, we extend our algorithm for finding such a matching to the problem of finding a load-balanced matching and a maximum load-max-balanced matching. Let I be an instance of SPA-STL.

6.3.3.2 Load-sum-balanced matchings

We show that the problem of finding a load-sum-balanced matching is solvable, using a network flow approach, in $O((n_1+m+n_2+n_3)\min\{n_1, t_{sum}\})$ time where $t_{sum} = \sum_{l_k \in L} t_k$.

First, we show that no lecturer gains more than their target number of allocations in any load-sum-balanced matching.

Lemma 6.3.1. *For instance I, let M be a load-sum-balanced matching. Then, there is no lecturer $l_k \in L$ such that $|M(l_k)| > t_k$.*

Proof. Assume for contradiction that in M there is some lecturer l_k with $|M(l_k)| > t_k$. Construct matching M' from M by removing any $|M(l_k)| - t_k$ students from projects that

l_k offers. Since we are only removing student-project pairs from M to reach M', it is not possible to break capacity constraints of any agents, therefore M' is also a valid matching. Additionally, due to the removals from the matching, the load-sum-balanced score has reduced from $r_s(M)$ to $r_s(M') = r_s(M) - (t_k - |M(l_k)|)$. But then $r_s(M') < r_s(M)$ meaning M is not a load-sum-balanced matching, a contradiction. □

Recall the definitions of a network and flow given in Chapter 4. We convert an instance I of SPA-STL into a network $N(I)$ as follows. First, each student, project and lecturer forms a vertex. Let v_{s_i}, v_{p_j} and v_{l_k} denote the vertices of student s_i, project p_j and lecturer l_k. Two additional vertices are added, namely, a *source* vertex s and a *sink* vertex t. The following edges are then added to $N(I)$.

- Edge (s, v_{s_i}) with capacity $c(s, v_{s_i}) = 1$ for all $s_i \in S$;
- Edge (v_{s_i}, v_{p_j}) with capacity $c(v_{s_i}, v_{p_j}) = 1$ for all $s_i \in S$ and $p_j \in P$ such that s_i finds p_j acceptable;
- Edge (v_{p_j}, v_{l_k}) with capacity $c(v_{p_j}, v_{l_k}) = c_j$ for all $p_j \in P$ such that p_j is offered by lecturer l_k;
- Edge (v_{l_k}, t) with capacity $c(v_{l_k}, t) = t_k$ for all $l_k \in L$.

The flow across each edge is initially set to 0. An example of this transformation is given as instance I_2 of SPA-STL and its associated network $N(I_2)$ in Figure 6.5. As in the network diagrams of Chapter 4, in Figure 6.5, each edge e has a pair of associated integers e_1/e_2 where e_1 is the flow over e and e_2 is the capacity of e.

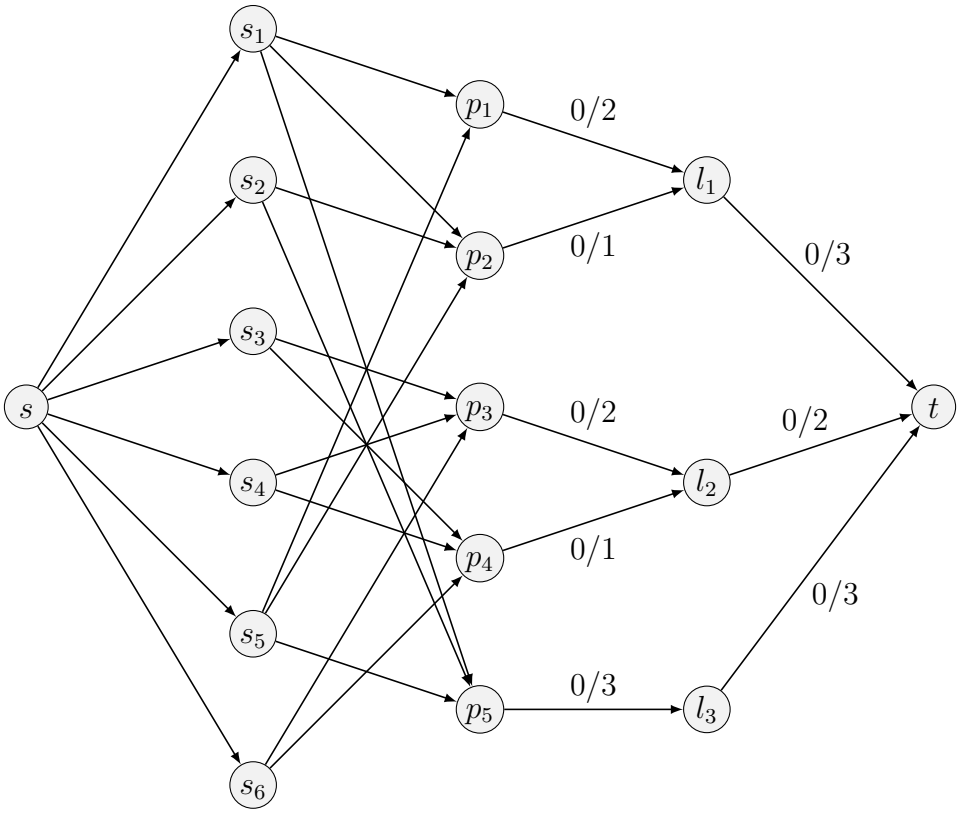
The transformation of a flow in $N(I)$ to a load-sum-balanced matching in I and vice versa, is now described. A flow f in $N(I)$ may be converted to a load-sum-balanced matching M in I by adding (s_i, p_j) to M for each $(s_i, p_j) \in S \times P$ where $f(v_{s_i}, v_{p_j}) = 1$. Let M be a load-sum-balanced matching in I. M may be converted into a flow f by adding a flow of 1 to edges (s, v_{s_i}), (v_{s_i}, v_{p_j}), (v_{p_j}, v_{l_k}) and (v_{l_k}, t) in $N(I)$ for each $(s_i, p_j) \in M$ where lecturer l_k supervises project p_j. Note that this cannot contravene the capacities on (v_{l_k}, t) edges since no lecturer in M can have a more allocations than their target, by Lemma 6.3.1.

We now show that we are able to find a load-sum-balanced matching in $O((n_1 + m + n_2 + n_3) \min\{n_1, t_{sum}\})$ time where $t_{sum} = \sum_{l_k \in L} t_k$.

The algorithm to find a load-sum-balanced matching (Algorithm Load-Sum-Bal) works as follows. First, create the network $N(I)$, as described above. Then, find the max flow f through $N(I)$, using a Max Flow Algorithm, and return a matching M created from f as described above. Theorem 6.3.2 shows that Algorithm Load-Sum-Bal produces a load-sum-balanced matching in $O((n_1 + m + n_2 + n_3) \min\{n_1, t_{sum}\})$ time.

Student preferences:	Project details:	Lecturer preferences:	
s_1: p_2 (p_1 p_5)	p_1: lecturer l_1, $c_1 = 2$	l_1: (s_3 s_2) s_5	$t_1 = d_1 = 3$
s_2: p_2 p_5	p_2: lecturer l_1, $c_2 = 1$	l_2: s_2	$t_2 = 2, d_2 = 3$
s_3: p_3 p_4	p_3: lecturer l_2, $c_3 = 2$	l_3: s_2 s_4 s_5	$t_3 = d_3 = 3$
s_4: (p_3 p_4)	p_4: lecturer l_3, $c_3 = 1$		
s_5: p_5 p_2 p_1	p_5: lecturer l_3, $c_3 = 3$		
s_6: p_4 p_3			

(a) SPA-STL instance I_2.



(b) Network $N(I_2)$ created from instance I_2 of SPA-STL. All source-student and student-project edges are assumed to have label '$0/1$'.

Figure 6.5: SPA-STL instance I_2 and associated network $N(I_2)$.

Theorem 6.3.2. *For instance I, Algorithm* Load-Sum-Bal *finds a load-sum-balanced matching M in $O((n_1 + m + n_2 + n_3)\min\{n_1, t_{sum}\})$ time where $t_{sum} = \sum_{l_k \in L} t_k$.*

Proof. Let $N(I)$ be the network created during Algorithm Load-Sum-Bal's execution, and let f be the final max flow in $N(I)$ output by the Ford-Fulkerson Algorithm [17]. Let f correspond to matching M where $r_s(M) = \sum_{l_k \in L} t_k - \text{val}(f)$. Suppose for a contradiction that M is not a load-sum-balanced matching. Then there exists some other load-sum-balanced matching M_{opt} with $r_s(M_{opt}) < r_s(M)$. From Lemma 6.3.1, no lecturer can have an allocation greater than their target in M_{opt}, and so by the process outlined above, let f_{opt} be the flow corresponding to M_{opt} in $N(I)$. Then, by construction, $r_s(M_{opt}) = \sum_{l_k \in L} t_k - \text{val}(f_{opt})$. But then, $r_s(M_{opt}) < r_s(M)$, which implies that

$$\sum_{l_k \in L} t_k - \text{val}(f_{opt}) < \sum_{l_k \in L} t_k - \text{val}(f)$$

and hence $\text{val}(f_{opt}) > \text{val}(f)$. Thus f is not a max flow in $N(I)$ and M would not have been returned by Algorithm Load-Sum-Bal, a contradiction.

Algorithm Load-Sum-Bal requires only one execution of the Ford-Fulkerson Algorithm, taking $O(|E|f_{max})$ time, where f_{max} is the value of the maximum flow through the network. Since there are n_1 edges from the source to each student vertex, m edges between student and project vertices, n_2 edges between project and lecturer vertices, and n_3 edges between lecturer vertices to the sink, we have a total of $|E| = n_1 + m + n_2 + n_3$ edges in $N(I)$. Hence the total time complexity is given by $O((n_1 + m + n_2 + n_3)\min\{n_1, t_{sum}\})$ time where $t_{sum} = \sum_{l_k \in L} t_k$. □

6.3.3.3 Load-balanced matchings

In this section, we show how to extend Algorithm Load-Sum-Bal to find a load-balanced matching in $O(t_{max}n_3 + (n_1 + m + n_2 + n_3)\min\{n_1, t_{sum}\})$ time where $t_{max} = \max\{t_k : l_k \in L\}$ and $t_{sum} = \sum_{l_k \in L} t_k$. In the next section, we show how to find a maximum load-max-balanced matching in the same time complexity.

Algorithm Load-Bal to find a load-balanced matching proceeds as follows. First we create the network $N(I)$ as detailed in the previous section, compute the highest target t_{max} among all lecturers and set a flow f to have value zero over the network $N(I)$. A variable r_z is initialised to 0. Over t_{max} rounds (Line 5 of Algorithm 6.1) we update the capacity of the lecturer-sink edges as follows. In round r we try to find a saturating flow that would correspond to a matching with a load-max-balanced score of $t_{max} - r$ for $r \in \{1, ..., t_{max}\}$. For lecturer l_k in round r, edge (v_{l_k}, t) is set to have a capacity of $t_k - (t_{max} - r)$ when $0 \leq t_k - (t_{max} - r)$ and a capacity of 0 when $t_k - (t_{max} - r) < 0$. The variable r_z is updated

to r on each iteration, which will either equal the value of r when we break out of the for loop or t_{max} if the for loop terminates naturally. Using the Ford-Fulkerson Algorithm [17] we then augment f with respect to an $N(I)$ updated with the new edge bounds described above, starting from our existing flow. If we are able to reach a *saturating flow* f, where all lecturer-sink edges are saturated, then we continue to the next round. If no such flow is found, then we break out of the for loop, and perform one final augmentation of f with respect to $N(I)$ (as long as $r_z \neq t_{max}$) such that the capacity of each (v_{l_k}, t) edge is t_k. Finally, we return matching M created from f on Lines 21 and 22.

Algorithm 6.1 Load-Bal(I), returns a load-balanced matching for an instance of SPA-STL.

Require: An instance I of SPA-STL.
Ensure: Returns a load-balanced matching M of I.

```
1:  Create network N(I)                          ▷ Described in the accompanying text.
2:  t_max ← max{t_k : l_k ∈ L}
3:  f ← zero flow through N(I)
4:  r_z ← 0
5:  for r in {1, ..., t_max} do
6:      for (v_{l_k}, t) ∈ N(I) do
7:          c(v_{l_k}, t) ← max{0, t_k − (t_max − r)}
8:      end for
9:      r_z ← r
10:     f ← augment(f, N(I))                     ▷ Augment f to a maximum flow in N(I).
11:     if Σ_{l_k∈L} f(v_{l_k}, t) ≠ Σ_{l_k∈L} c(v_{l_k}, t) then   ▷ If f is not a saturating flow.
12:         break
13:     end if
14: end for
15: if r_z ≠ t_max then
16:     for (v_{l_k}, t) ∈ N(I) do
17:         c(v_{l_k}, t) ← t_k
18:     end for
19:     f ← augment(f, N(I))                     ▷ Augment f to a maximum flow in N(I).
20: end if
21: Create M from f                              ▷ Described in the accompanying text.
22: return M
```

Figure 6.6 shows the state of $N(I_2)$ after the Max Flow algorithm has been run for round 2, and the final augmentation of f on Line 19. In these figures, a bold line indicates a non-zero flow over an edge. Since all source-student and student-project edges have capacity 1, thin and bold lines over source-student and student-project edges represent a flow of value 0 and 1 respectively. In Figure 6.6a we can see that in round 2, no saturating flow has been found. Hence, at this stage of Algorithm Load-Bal, we break out of the for loop on Line 12. Since $r_z = 2$, we have $r_z \neq t_{max}$, and so the final augmentation on Line 19 is executed. Figure 6.6b shows the state of $N(I)$ after this final augmentation, when each lecturer-sink edge (v_{l_k}, t) has a capacity of t_k. Matching M associated with the final flow f would be created

on Line 21, giving $M = \{(s_1, p_1), (s_2, p_2), (s_3, p_3), (s_4, p_3), (s_5, p_5)\}$ with $r_s(M) = 3$ and $r_m(M) = 2$.

Theorems 6.3.6 and 6.3.8 shows that Algorithm Load-Bal produces a load-balanced matching and runs in $O(t_{max}n_3 + (n_1 + m + n_2 + n_3)\min\{n_1, t_{sum}\})$ time where $t_{max} = \max\{t_k : l_k \in L\}$ and $t_{sum} = \sum_{l_k \in L} t_k$. In order to prove this we first define a *reduced load-max-balanced matching*, that is, a load-max-balanced matching in which each lecturer gains $\max\{0, t_k - r_m(M_{opt})\}$ allocations (where M_{opt} is a load-max-balanced matching), and show that it may be constructed from a load-max-balanced matching in polynomial time in Lemma 6.3.4. Next, in Lemma 6.3.5, we show that there is a 1-1 correspondence between reduced load-max-balanced matchings in I and saturating flows of $N(I)$ where each (v_{l_k}, t) edge has capacity $\max\{0, t_k - r_m(M_{opt})\}$.

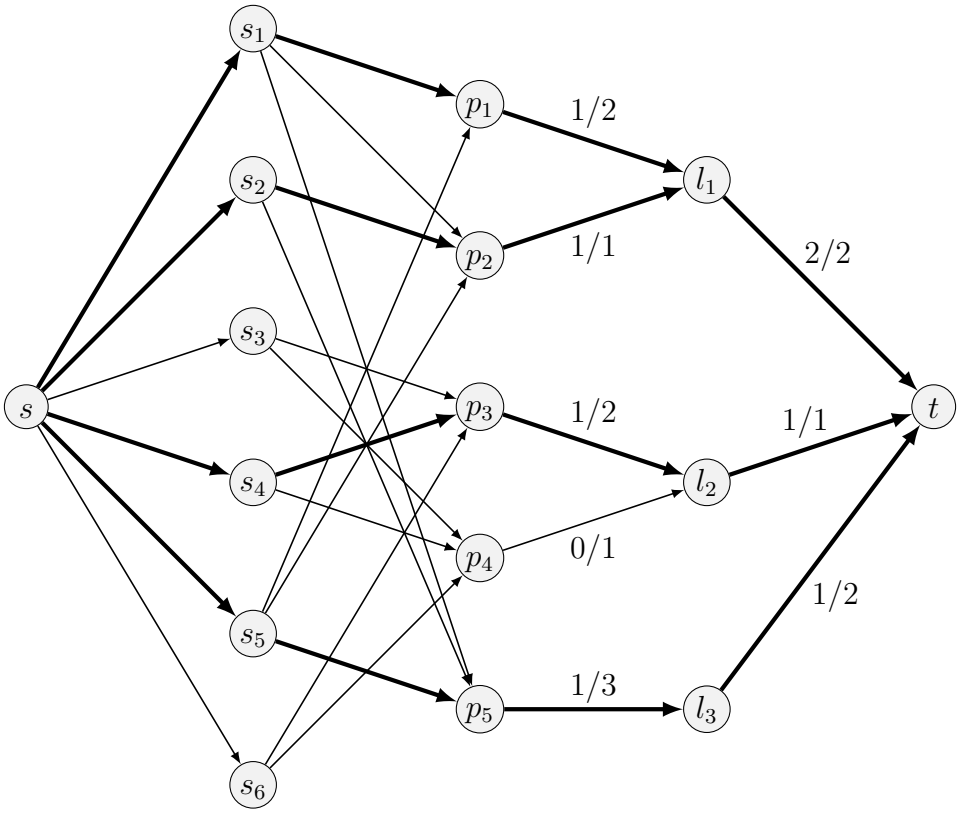
Definition 6.3.3. *A* reduced load-max-balanced matching *is a load-max-balanced matching such that* $|M(l_k)| = \max\{0, t_k - r_m(M_{opt})\}$ *for all* $l_k \in L$*, where* M_{opt} *is a load-max-balanced matching.*

Lemma 6.3.4. *Given a load-max balanced matching* M_{opt}*, we may construct in linear time, a reduced load-max balanced matching* M'_{opt} *from* M_{opt} *such that* $r_m(M'_{opt}) = r_m(M_{opt})$*.*

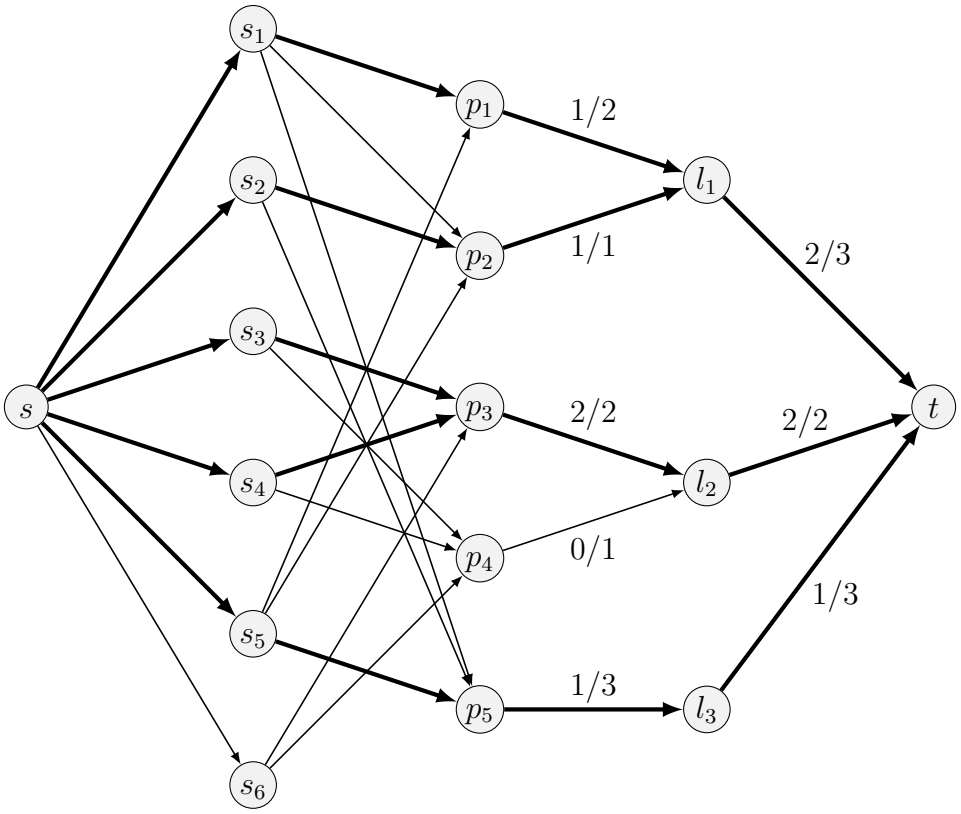
Proof. We obtain M'_{opt} from M_{opt} as follows. For each lecturer l_k in M_{opt}, remove any students from projects supervised by l_k until l_k has $\max\{0, t_k - r_m(M_{opt})\}$ allocations. Since we have only removed student-project pairs from a valid matching, no capacity constraints can be contravened and therefore M'_{opt} is also a valid matching. Additionally, the number of allocations for each lecturer is within $r_m(M_{opt})$ of their target, hence M'_{opt} is also a load-max-balanced matching. The process of constructing M'_{opt} requires only a single pass through the student-project allocations of each lecturer. Therefore we can construct M'_{opt} in linear time. □

Lemma 6.3.5. *Let* M_{opt} *be a load-max-balanced matching in* I*. There is a* 1-1 *correspondence between reduced load-max-balanced matchings in* I *and saturating flows of* $N(I)$ *where for each lecturer* l_k*, the edge* (v_{l_k}, t) *has capacity* $\max\{0, t_k - r_m(M_{opt})\}$*.*

Proof. Let M be a reduced load-max-balanced matching in I. We construct f as follows. For each (s_i, p_j) pair in M, where p_j is supervised by lecturer l_k, we add a flow of 1 to edges (s, v_{s_i}), (v_{s_i}, v_{p_j}), (v_{p_j}, v_{l_k}) and (v_{l_k}, t). Clearly, for each such pair, this transfers one unit of flow into and out of vertices v_{s_i}, v_{p_j} and v_{l_k} meaning that flow conservation is maintained. Capacities of students and projects in M must be respected and correspond directly to the capacities of student-project and project-lecturer edges in $N(I)$. Finally, since M is a reduced load-max-balanced matching, each lecturer l_k has $|M(l_k)| = \max\{0, t_k - r_m(M)\}$ assignees, where $r_m(M) = r_m(M_{opt})$. This corresponds exactly to a flow of $\max\{0, t_k - r_m(M)\}$ through each (v_{l_k}, t) edge with capacity $\max\{0, t_k - r_m(M)\}$. Hence f is a saturating flow.



(a) Flow through $N(I_2)$ after round 2.



(b) Flows through $N(I_2)$ after the final augmentation on Line 19 of Algorithm Load-Bal.

Figure 6.6: Flows through $N(I_2)$ during the execution of Algorithm Load-Bal.

Now, let f be a saturating flow of $N(I)$. M is constructed by adding (s_i, p_j) to M for each $(s_i, p_j) \in S \times P$ where $f(v_{s_i}, v_{p_j}) = 1$. Since f is a saturating flow, each of the lecturer-sink edges must have a flow equal to their capacity. But then by construction, the corresponding (s_i, p_j) pairs will produce a matching M where each lecturer $l_k \in L$ has exactly $\max\{0, t_k - r_m(M)\}$ allocations. Additionally, capacities of students and projects will have been enforced by the network and so M is a reduced load-max-balanced matching. □

Theorem 6.3.6. *Algorithm* Load-Bal *always produces a load-balanced matching.*

Proof. There is one point in Algorithm 6.1's execution where a matching may be returned, namely Line 22. We show that this matching must be both a load-max-balanced matching and a load-sum-balanced matching.

If $t_{max} = 0$, then clearly the algorithm performs no augmentations of f, and we exit with a matching M corresponding to a flow with value 0, which is optimal. From this point on, assume that $t_{max} > 0$.

Let M be the matching returned on Line 22. Then one of the following cases must be true.

- $r_z = t_{max}$ and the final augmentation of f on Line 10 for round $r_z = t_{max}$ produced a saturating flow;
- $r_z = t_{max}$ and the final augmentation of f on Line 10 for round $r_z = t_{max}$ produces a non-saturating flow;
- $r_z \neq t_{max}$ and the final augmentation of f on Line 10 for round r_z produced a non-saturating flow;

In the first two cases, the final max flow is calculated on Line 10 since $r_z = t_{max}$ and so we cannot enter the if statement on Line 15. In the third case, the final max flow is calculated on Line 19. Note that, for the third case, any further flow augmentation on Line 19 may only improve lecturer outcomes up to their target number of allocations. It also cannot increase the flow through all lecturer-sink edges, or the most recent augmentation on Line 10 would have found a saturating flow. Therefore, the load-max-balanced score of the associated matching M is unchanged by the augmentation on Line 19.

In the first case, we have a saturating flow f corresponding to a matching M where $r_m(M) = 0$. But then it is also be the case that $r_s(M) = 0$, and so we would have returned a load-balanced matching.

In the second and third case, we have augmented to a non-saturating flow f in round r_z on Line 10. Hence, no saturating flow was found in $N(I)$ when lecturer-sink capacities are given by $\max\{0, t_k - (t_{max} - r)\} = \max\{0, t_k - r_m(M) + 1\}$ for each lecturer $l_k \in L$.

Assume for contradiction that there exists a load-max-balanced matching M_{opt} with $r_m(M_{opt}) < r_m(M)$. By Lemma 6.3.4, we know that there must exist a reduced load-max-balanced matching M'_{opt} with $r_m(M'_{opt}) = r_m(M_{opt})$. Then, by Lemma 6.3.5, M'_{opt} corresponds to a saturating flow f' of $N(I)$ where lecturer-sink capacities are given by $\max\{0, t_k - r_m(M_{opt})\}$ for each lecturer $l_k \in L$. Since saturating flow f' exists, there must also be a saturating flow for any $N(I)$ with lecturer-sink capacities less than or equal to $\max\{0, t_k - r_m(M_{opt})\}$ for each lecturer $l_k \in L$. Notice that for all $l_k \in L$, since $r_m(M_{opt}) < r_m(M)$, we have $t_k - r_m(M_{opt}) > t_k - r_m(M)$, implying that $t_k - r_m(M_{opt}) \geq t_k - r_m(M) + 1$. But then, network $N(I)$ in round r_z has lecturer-sink capacities less than or equal to $\max\{0, t_k - r_m(M_{opt})\}$, contradicting the fact that no saturating flow was found in round r_z. Since the augmentation on Line 19 does not change the load-max-balanced score of the output matching M, Algorithm Load-Bal always produces a load-max-balanced matching.

It remains to show that M is a load-sum-balanced matching. In all cases above, we always augment f to a maximum flow over $N(I)$ such that each lecturer-sink edge (v_{l_k}, t) has capacity t_k. This happens on Line 10 for the first and second case, and Line 19 for the third case. Since Algorithm Load-Sum-Bal comprises finding a max flow over $N(I)$ with precisely these capacities, M is also a load-sum-balanced matching, by Theorem 6.3.2.

Therefore Algorithm Load-Bal always produces a load-balanced matching. □

Corollary 6.3.7. *A load-balanced matching always exists. Furthermore, a matching M that has minimum load-sum-balanced score taken over all load-max-balanced matchings is a load-balanced matching. Furthermore, a matching M that has minimum load-max-balanced score taken over all load-sum-balanced matchings is a load-balanced matching.*

Theorem 6.3.8. *Algorithm* Load-Bal *runs in $O(t_{max}n_3 + (n_1 + m + n_2 + n_3)\min\{n_1, t_{sum}\})$ time where $t_{max} = \max\{t_k : l_k \in L\}$ and $t_{sum} = \sum_{l_k \in L} t_k$.*

Proof. The Ford-Fulkerson Algorithm [17] runs in $O(|E|f_{max})$ time where f_{max} is the maximum flow through the network. As in Theorem 6.3.2, there are a total of $|E| = n_1 + m + n_2 + n_3$ edges in $N(I)$. Although we restart the Ford-Fulkerson Algorithm a number of times over Algorithm Load-Bal's execution, we always begin from the previous flow. This is possible since the flow found using the Ford-Fulkerson Algorithm always increases in increments of one. Therefore, the time complexity of finding maximum flows over the whole algorithm is equal to finding the maximum flow once with maximum possible flow. This corresponds to $O((n_1 + m + n_2 + n_3)\min\{n_1, t_{sum}\})$ time where $t_{sum} = \sum_{l_k \in L} t_k$. Additionally, there are a maximum of t_{max} rounds from Line 5, giving a total time complexity of $O(t_{max}n_3)$ to update lecturer capacities on each round, where $t_{max} = \max\{t_k : l_k \in L\}$. Hence Algorithm Load-Bal runs in $O(t_{max}n_3 + (n_1 + m + n_2 + n_3)\min\{n_1, t_{sum}\})$ time. □

6.3.3.4 Maximum load-max-balanced matchings

The algorithm to find a maximum load-max-balanced matching (Algorithm Load-Max-Bal-Max), requires an amendment to Algorithm Load-Bal as follows. Replace Lines 15 to 22 of Algorithm 6.1 with the pseudocode outlined in Algorithm 6.2. On Lines 15 to 17 of Algorithm 6.2, we test if the matching M created from the existing flow f has a load-max-balanced score of 0. If it does, then M is optimal and it is returned. If not, then we perform one final augmentation of f on $N(I)$ where the capacity of each lecturer-sink edge (v_{l_k}, t) is set to $\min\{t_k + r_m(M), d_k\}$. A bound for each lecturer of $t_k + r_m(M)$, up to a maximum of d_k, allows each lecturer to achieve their maximum possible number of allocations, whilst keeping within the load-max-balanced score. Following this final augmentation, we return the matching M created from f.

Algorithm 6.2 Max-Load-Max-Bal(I), adaptation to Algorithm 6.1 which returns a maximum load-max-balanced matching for an instance I of SPA-STL (replacing Lines 15 to 22 of Algorithm 6.1).

Require: An instance I of SPA-STL.
Ensure: Returns a maximum load-max-balanced matching of I.
14: Create M from f
15: **if** $r_m(M) = 0$ **then**
16: **return** M
17: **end if**
18: **for** $(v_{l_k}, t) \in N(I)$ **do**
19: $c(v_{l_k}, t) \leftarrow \min\{t_k + r_m(M), d_k\}$
20: **end for**
21: $f \leftarrow$ augment$(f, N(I))$ ▷ Augment f to a maximum flow in $N(I)$.
22: Create M from f
23: **return** M

Recall Figure 6.6 shows the state of $N(I_2)$ after the Max Flow algorithm has been run for round 2 and the final augmentation of f on Line 19 for Algorithm Load-Bal. In Algorithm Load-Max-Bal-Max, we replace Figure 6.6b with Figure 6.7, which shows the flow f through $N(I_2)$ after a final augmentation on Line 21. Recall that in these figures, a bold line indicates a non-zero flow over an edge, and that all source-student and student-project edges have capacity 1. Thus thin and bold lines over source-student and student-project edges represent a flow of value 0 and 1 respectively. In this example, the increased capacity of edge (v_{l_2}, t) has allowed the flow to increase beyond that of Figure 6.6b, and so $M = \{(s_1, p_1), (s_2, p_2), (s_3, p_3), (s_4, p_3), (s_5, p_5), (s_6, p_4)\}$ with $r_m(M) = 2$ is a maximum load-max-balanced matching.

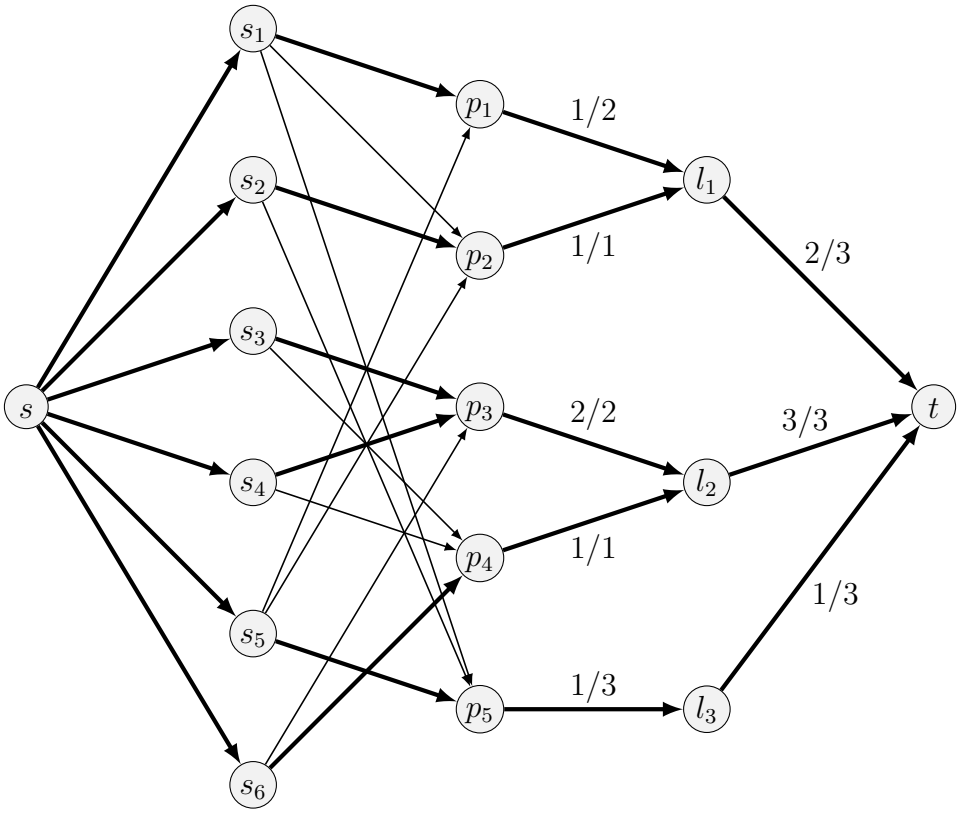


Figure 6.7: Flows through $N(I_2)$ after the final augmentation on Line 21 of Algorithm Load-Max-Bal-Max.

Theorem 6.3.9. *Algorithm* Load-Max-Bal-Max *produces a maximum load-max-balanced matching in* $O(t_{max}n_3 + (n_1 + m + n_2 + n_3)\min\{n_1, t_{sum} + n_3 r_m\})$ *time where* $t_{max} = \max\{t_k : l_k \in L\}$, $t_{sum} = \sum_{l_k \in L} t_k$ *and* r_m *is the load-max-balanced score of a load-max-balanced matching.*

Proof. There are two points in Algorithm 6.2 where we return a matching, namely, Lines 16 and 23.

Let M_1 be the matching returned on Line 16. Then $r_m(M_1) = 0$ and so it is not possible to increase the number of allocations for any lecturer whilst still retaining a load-max-balanced matching. Hence, M_1 must be a maximum load-max-balanced matching.

Let M_2 be the matching returned on Line 23. By the argument given in the proof of Theorem 6.3.6, we know that M created on Line 14 is a load-max-balanced matching. Then since the capacity of each lecturer-sink edge (v_{l_k}, t) is set to equal $t_k + r_m(M)$, the augmentation of f on Line 21 will also be associated with a load-max-balanced matching. Additionally, since f is a max flow, it must be associated with a load-max-balanced matching with maximum size. Hence, M_2 is a maximum load-max-balanced matching.

To calculate the time complexity of Algorithm Load-Max-Bal-Max we may use almost similar reasoning to that of Theorem 6.3.8. However, note that the maximum flow we may now

Student preferences:	Project details:	Lecturer preferences:	
s_1: p_1	p_1: lecturer l_1, $c_1 = 1$	l_1: s_1	$t_1 = 0, d_1 = 1$

Figure 6.8: SPA-STL instance I_3.

augment to is given by $f_{max} = t_{sum} + n_3 r_m$ where $t_{sum} = \sum_{l_k \in L} t_k$ and r_m is the load-max-balanced score of a load-max-balanced matching. Hence we have an overall time complexity for Algorithm Load-Max-Bal-Max of $O(t_{max} n_3 + (n_1 + m + n_2 + n_3) \min\{n_1, t_{sum} + n_3 r_m\})$ time where $t_{max} = \max\{t_k : l_k \in L\}$. □

6.3.4 Stability with load balancing

Recall the definition of SPA-S given in Section 2.5.2.1. The Student-Project Allocation problem with lecturer preferences over Students and Lecturer targets (SPA-SL) extends SPA-S to include lecturer targets. The definition of *stability* in SPA-SL is the same as in the SPA-S scenario. By the Unpopular Projects Theorem [6] (introduced in Section 2.5.2.1), each lecturer is assigned the same number of students in any instance of SPA-S, and also therefore SPA-SL. Hence, any stable matching in SPA-SL must also be a load-max-balanced stable matching, a load-sum balanced stable matching and a load-balanced stable matching.

We now look at the problem of finding these types of stable matchings in SPA-STL.

Unlike the SPA-S case, stable matchings in SPA-ST (and therefore SPA-STL) may be different sizes, and lecturers may have different numbers of students in different stable matchings. Additionally, for a given instance, there may be no load-max-balanced stable matching, load-sum-balanced stable matching or load-balanced stable matching such that all lecturers have a number of allocations less than or equal to their targets. This is in contrast to the case where stability is not present. Instance I_3 in Figure 6.8 shows a simple example of this. In I_3 there is only one stable matching, namely $M = \{(s_1, p_1)\}$. Thus M must be the load-max-balanced stable matching, load-sum-balanced stable matching and load-balanced stable matching. However, l_1 has a target of 0 allocations and therefore this example shows that it is possible for optimal stable matchings of these types to exist where lecturers have more allocations than their targets. This is in contrast with the load-max-balanced, load-sum-balanced and load-balanced matchings of I_3 which would be empty, and hence no lecturer with more allocations than their targets would exist relative to these matchings.

We now study the complexity of finding a load-max-balanced stable matching, load-sum-balanced stable matching and load-balanced stable matching in SPA-STL.

Let LMBS-SPA-STL be the problem of finding a load-max-balanced stable matching in an instance I of SPA-STL and let LSBS-SPA-STL be the problem of finding a load-sum-balanced stable matching in an instance I of SPA-STL. Finally, let LBS-SPA-STL be the problem of

finding a load-balanced stable matching in an instance I of SPA-STL. We define three special-case decision problems.

Definition 6.3.10.

- *We define* LMBS-SPA-STL-D*, the decision version of* LMBS-SPA-STL*, as follows. An instance I of* LMBS-SPA-STL-D *comprises an instance I of* SPA-STL *and a non-negative integer K. The problem is to decide whether there exists a stable matching M in I such that $r_m(M) \leq K$.*

- *We define* LSBS-SPA-STL-D*, the decision version of* LSBS-SPA-STL*, as follows. An instance I of* LSBS-SPA-STL-D *comprises an instance I of* SPA-STL *and a non-negative integer K. The problem is to decide whether there exists a stable matching M in I such that $r_s(M) \leq K$.*

- *We define* LBS-SPA-STL-D*, the decision version of* LBS-SPA-STL*, as follows. An instance I of* LBS-SPA-STL-D *comprises an instance I of* SPA-STL *and non-negative integers K and K'. The problem is to decide whether there exists a stable matching M in I such that $r_m(M) \leq K$ and $r_s(M) \leq K'$.*

Theorem 6.3.11. LMBS-SPA-STL-D *is* NP*-complete, even if $K = 0$, students preference lists are strictly ordered and of length 3 and each lecturer's preference list is either strictly ordered and of length 3, or, is a tie of length 2.*

Proof. First, we show that LMBS-SPA-STL-D is in NP. Given a matching M it is possible to determine whether M is stable in $O(m)$ time. This is done by iterating over preference lists of each student s_i, and determining whether each s_i-project pair on their preference list constitutes a blocking pair, with respect to M, as defined in Section 2.5.2.1. The load-max-balanced score of M may also be calculated and compared to 0 in $O(n_3)$ time. Thus LMBS-SPA-STL-D is in NP.

The Stable Marriage problem with Ties and Incomplete lists (SMTI) was described in Section 2.2.3. Let $(3,3)$-COM-SMTI be the problem of finding a maximum stable matching in an instance I' of SMTI, such that each man's preference list is strictly ordered and of length 3, and, each woman's preference list is either strictly ordered and of length 3, or is a tie of length 2. We define $(3,3)$-COM-SMTI-D a special-case decision problem of $(3,3)$-COM-SMTI, as follows. An instance I' of $(3,3)$-COM-SMTI-D comprises an instance I' of SMTI, with the restrictions described above. The problem is to decide whether there exists a stable matching M' in I' such that $|M'| = n$, where n is the number of men or women. $(3,3)$-COM-SMTI-D is NP-complete [57].

We reduce from $(3,3)$-COM-SMTI-D as follows.

First, we construct an instance I from I'. For each man m_i in I' add student s_i to I. For each woman w_j in I' add project p_j and lecturer l_j to I. For each woman w_j on m_i's preference list, we add project p_j to s_i's list. Similarly, for each man m_i on w_j's preference list, we add student s_i to l_j's list, ensuring we retain any ties. Each project p_j is set to have a capacity of 1 and is supervised by lecturer l_j who themselves has a target and capacity of 1.

We claim that I' has a stable matching M' of size $|M'| = n$ if and only if I has a stable matching M such that $r_m(M) = 0$.

Suppose I' has a stable matching M' of size $|M'| = n$. We create matching M by adding pair (s_i, p_j) to M for each $(m_i, w_j) \in M'$. This must be a valid matching since students in I and men in I' may only have a single allocation and projects and lecturers in I and women in I' may also only have a single allocation. Therefore, if no men or women are multiply assigned in M', then no student, project or lecturer can be multiply assigned in M. Note that in M, all students are allocated one project, and all projects and lecturers are allocated one student, by construction. We now show that M is stable. Suppose for contradiction that there is a blocking pair (s_i, p_j) of M in I. Then, s_i would prefer to be assigned to p_j than $M(s_i)$. Since $|M(l_j)| = 1$ by construction, and since each lecturer offers only one project, $s_i \notin M(l_j)$, and so l_j would prefer to be assigned to s_i than $M(l_j) = M(p_j)$. But, by construction, this would translate to a pair $(m_i, w_j) \notin M'$ who would prefer to be allocated to each other than their assigned partners in I'. Hence, (m_i, w_j) would block M', a contradiction, and so M is stable in I. By construction, we know that $|M| = |M'| = n$, and so all n lecturers must have exactly one allocation (equaling their target), giving $r_m(M) = 0$, as required.

Conversely, suppose that I has a stable matching M such that $r_m(M) = 0$. Create matching M' by adding pair (m_i, w_j) to M' for each $(s_i, p_j) \in M$. M' is a valid matching since students in I and men in I' may only have a single allocation and projects and lecturers in I and women in I' may also only have a single allocation. Therefore, if no student, project or lecturer is multiply assigned in M, then no man or woman can be multiply assigned in M'. Now we show M' is stable. Suppose for contradiction that there is some pair (m_i, w_j) blocking M' in I'. Then, m_i would prefer to be assigned to w_j than $M'(m_i)$ and w_j would prefer to be assigned to m_i than $M'(w_j)$. But, by construction, this would mean that s_i would prefer to be assigned to p_j than $M(s_i)$. Additionally, since the capacity of l_j is 1 and l_j offers only one project, $s_i \notin M(l_j)$, and so l_j would prefer to be assigned to s_i than $M(l_j) = M(p_j)$. Hence, (s_i, p_j) would block M, a contradiction, and so M' is stable in I'. We know that M has a load-max-balanced score of $r_m(M) = 0$, therefore all lecturers must have a one student allocated each. But since lecturers were created from women in I', it must be that $n = |M| = |M'|$, as required.

We have shown that I' has a stable matching M' of size $|M'| = n$ if and only if I has a stable matching M such that $r_m(M) = 0$. Since the reduction described above can be executed in

polynomial time, LMBS-SPA-STL-D is NP-hard. Additionally, since LMBS-SPA-STL-D is in NP, it follows that LMBS-SPA-STL-D is NP-complete. □

Corollary 6.3.12. LSBS-SPA-STL-D *is* NP*-complete, even if* $K = 0$*, students preference lists are strictly ordered and of length* 3 *and each lecturer's preference list is either strictly ordered and of length* 3*, or, is a tie of length* 2*.*

Proof. We may use an identical reduction to that given in Theorem 6.3.11 to show that I' has a stable matching M' of size $|M'| = n$ if and only if I has a stable matching M such that $r_s(M) = 0$. □

Corollary 6.3.13. LBS-SPA-STL-D *is* NP*-complete, even if* $K = 0$*,* $K' = 0$*, students preference lists are strictly ordered and of length* 3 *and each lecturer's preference list is either strictly ordered and of length* 3*, or, is a tie of length* 2*.*

Proof. We may use an identical reduction to that given in Theorem 6.3.11 to show that I' has a stable matching M' of size $|M'| = n$ if and only if I has a stable matching M such that $r_m(M) = 0$ and $r_s(M) = 0$. □

6.3.5 IP models

In Section 6.3.4 we showed that the problem of finding a load-balanced stable matching was NP-hard. In this section, we describe an IP model to solve this problem. By Corollary 6.3.7, a load-balanced matching may be returned by either finding a matching that has minimum load-sum-balanced score taken over all load-max-balanced matchings, or, finding a matching that has minimum load-max-balanced score taken over all load-sum-balanced matchings. Using this fact, we are able to find a load-balanced matching by first ensuring we have a load-max-balanced matching and, subject to this, a load-sum-balanced matching. First we present an IP model for finding a load-max-balanced stable matching.

6.3.5.1 IP model for a load-max-balanced stable matching

Figure 6.9 shows the IP model optimisation for finding a load-max-balanced matching. In this model we include Constraints 1-7 from Figure 6.1 which ensure stability. In Constraints 8 and 9, we let l_k^+ and l_k^- be greater than or equal to the difference between the number of allocations a lecturer has when compared to their target, over and under the target, respectively. Constraints 10 and 11 set h_m be greater than or equal to the maximum of these two values over all l_k^+ and l_k^-. h_m is therefore greater than or equal to the maximum absolute difference between a lecturer's target and their number of allocations, over all lecturers. A load-max-balanced matching is then a matching that minimises h_m.

minimise: h_m

subject to:

Constraints 1-7 of Figure 6.1, and

$$8. \quad l_k^+ \geq \sum_{s_i \in S} \sum_{p_j \in P_k} x_{ij} - t_k \qquad \forall l_k \in L$$

$$9. \quad l_k^- \geq t_k - \sum_{s_i \in S} \sum_{p_j \in P_k} x_{ij} \qquad \forall l_k \in L$$

$$10. \quad h_m \geq l_k^+ \qquad \forall l_k \in L$$

$$11. \quad h_m \geq l_k^- \qquad \forall l_k \in L$$

$$l_k^+, l_k^- \in \{-t_k, ..., 2t_k\} \qquad \forall l_k \in L$$

$$h_m \in \{0, ..., t_m\} \qquad t_m = \max\{t_k : l_k \in L\}$$

Figure 6.9: IP model for finding a load-max-balanced stable matching.

We now prove correctness for the IP model to find a load-max-balanced matching shown in Figure 6.9.

Theorem 6.3.14. *Given an instance I of* SPA-STL*, let J be the IP model as defined in Figure 6.9. A stable matching M in I corresponds to a feasible solution in J and vice versa.*

Proof. By Theorem 6.2.2, we know that a stable matching in I corresponds to a feasible solution in J and vice versa, when considering only Constraints 1-7.

We first show that a stable matching M in I corresponds to an feasible solution of J. Let M be a stable matching in I. We construct an assignment $f = \langle \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{l}^+, \boldsymbol{l}^-, h_m \rangle$ of J as follows. Set variables $\boldsymbol{x}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ as in Theorem 6.2.2. Set variables l_k^+ to equal $|M(l_k)| - t_k$ and l_k^- to equal $t_k - |M(l_k)|$, for all lecturers l_k. Set h_m to equal the maximum of l_k^+ and l_k^- over all l_k. By Theorem 6.2.2, Constraints 1-7 are satisfied. Additionally, by the assignments of $\boldsymbol{l}^+$, $\boldsymbol{l}^-$ and h_m above, Constraints 8-11 are satisfied. Therefore f is a feasible assignment of J.

Now we show that a feasible solution of J corresponds to a stable matching M in I. Let $f = \langle \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{l}^+, \boldsymbol{l}^-, h_m \rangle$ be an feasible solution of J. Construct matching M as in Theorem

6.2.2, that is, for each x_{ij} variable in J, if $x_{ij} = 1$ then add pair (s_i, p_j) to M in I. By Theorem 6.2.2, M is a stable matching of I.

Therefore, a stable matching M in I corresponds to a feasible solution in J and vice versa. □

Corollary 6.3.15. *Given an instance I of* SPA-STL*, let J be the IP model as defined in Figure 6.9. A load-max-balanced stable matching M in I corresponds to an optimal solution in J and vice versa.*

Proof. Suppose M is a load-max-balanced stable matching in I. Then, from the first half of the proof of Theorem 6.3.14, M corresponds to a feasible solution $f = \langle \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{l}^+, \boldsymbol{l}^-, h_m \rangle$ of J. Now, assume for contradiction that f is not optimal. Then there exists some feasible solution $g = \langle \boldsymbol{x}', \boldsymbol{\alpha}', \boldsymbol{\beta}', \boldsymbol{l}^{+\prime}, \boldsymbol{l}^{-\prime}, h'_m \rangle$ of J such that $h'_m < h_m$. But, by the second half of the proof of Theorem 6.3.14, g corresponds to a stable matching M' such that $r_m(M') = h'_m < h_m = r_m(M)$, contradicting the fact that M has minimum load-max-balanced score in $\mathcal{M}_S$. Hence a load-max-balanced stable matching M in I corresponds to an optimal solution in J.

Conversely, suppose $f = \langle \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{l}^+, \boldsymbol{l}^-, h_m \rangle$ is an optimal solution of J, and by the second half of the proof of Theorem 6.3.14, let M be the stable matching corresponding to f in I. Assume for contradiction, that there is some other stable matching M' such that $r_m(M') < r_m(M)$. Then, by the first half of the proof of Theorem 6.3.14, M' corresponds to a feasible solution $g = \langle \boldsymbol{x}', \boldsymbol{\alpha}', \boldsymbol{\beta}', \boldsymbol{l}^{+\prime}, \boldsymbol{l}^{-\prime}, h'_m \rangle$ of J such that $h'_m = r_m(M') < r_m(M) = h_m$, which contradicts the fact that f is optimal. Hence an optimal solution in J corresponds to a load-max-balanced stable matching M in I.

Therefore, a load-max-balanced stable matching M in I corresponds to an optimal solution in J and vice versa. □

Should we wish to apply subsequent optimisations to the IP model in Figure 6.9, we must add the constraint shown in Figure 6.10 in order to ensure we return a load-max-balanced stable matching.

12.	$h_m \leq r_m$	where r_m is the load-max-balanced score

Figure 6.10: IP model constraint for finding a load-max-balanced stable matching.

6.3.5.2 IP model for a load-balanced stable matching

The additional IP model constraints and objective function required to find a load-balanced matching can be seen Figure 6.11. This is designed to be solved after the IP model in Figure

minimise: $\sum_{l_k \in L} h_k$

subject to:

Constraints 1-11 of Figure 6.9, and

13. $h_k \geq l_k^+ \quad \forall l_k \in L$
14. $h_k \geq l_k^- \quad \forall l_k \in L$

$h_k \in \{0, ..., t_m\} \qquad t_m = \max\{t_k : l_k \in L\}, \forall l_k \in L$

Figure 6.11: IP model for finding a load-balanced stable matching.

6.9 has been solved and Constraint 12 of Figure 6.10 has been added to the model. Our objective in this model is to minimise the sum of h_k variables, where h_k is set to be greater than or equal to the maximum of l_k^+ and l_k^-, for each $l_k \in L$.

We now prove correctness for the IP model to find a load-balanced matching shown in Figure 6.11.

Theorem 6.3.16. *Given an instance I of* SPA-STL*, let J be the IP model as defined in Figure 6.11. A load-max-balanced stable matching M in I corresponds to a feasible solution in J and vice versa.*

Proof. By Theorem 6.3.14, we know that a stable matching in I corresponds to a feasible solution in J and vice versa, when considering only Constraints $1 - 11$.

Assume instance I contains a load-max-balanced stable matching M. We first show that M corresponds to a feasible solution in J. We construct an assignment $f = \langle \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{l}^+, \boldsymbol{l}^-, h_m, \boldsymbol{h} \rangle$ in J where $\langle \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{l}^+, \boldsymbol{l}^-, h_m \rangle$ are as constructed as in the proof Theorem 6.3.14, and $\boldsymbol{h}$ comprises the assignment of h_k to equal the maximum of l_k^+ and l_k^- for each l_k. By Theorem 6.3.14 and Corollary 6.3.15, it is clear that constraints 1-12 are satisfied. By the assignments of $\boldsymbol{h}$, Constraints 13 and 14 are also satisfied. Therefore f is a feasible assignment of J.

Now we show that a feasible solution of J corresponds to a load-max-balanced stable matching M in I. Let $f = \langle \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{l}^+, \boldsymbol{l}^-, h_m, \boldsymbol{h} \rangle$ be a feasible solution of J. Construct M as

in the proof of Theorem 6.2.2, on the basis of the $\boldsymbol{x}$ variables. Then by Theorem 6.3.14, Corollary 6.3.15 and Constraint 12, M is a load-max-balanced stable matching in I.

Therefore, a load-max-balanced stable matching M corresponds to an optimal solution in J and vice versa. □

Corollary 6.3.17. *Given an instance I of* SPA-STL*, let J be the IP model as defined in Figure 6.11. A load-balanced stable matching M in I corresponds to an optimal solution in J and vice versa.*

Proof. Suppose M is a load-balanced stable matching in I. Then, from the first half of the proof of Theorem 6.3.16, M corresponds to a feasible solution $f = \langle \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{l}^+, \boldsymbol{l}^-, h_m, \boldsymbol{h} \rangle$ of J. Now assume for contradiction that f is not optimal. Then there exists some solution $g = \langle \boldsymbol{x}', \boldsymbol{\alpha}', \boldsymbol{\beta}', \boldsymbol{l}^{+\prime}, \boldsymbol{l}^{-\prime}, h'_m, \boldsymbol{h}' \rangle$ of J such that $\sum_{l_k \in L} h'_k < \sum_{l_k \in L} h_k$. But, by the second half of the proof of Theorem 6.3.16, g then corresponds to a load-max-balanced stable matching M' such that $r_s(M') = \sum_{l_k \in L} h'_k < \sum_{l_k \in L} h_k = r_s(M)$, contradicting the fact that M has minimum load-sum-balanced score in $\mathcal{M}_S$. Hence a load-balanced stable matching M in I corresponds to an optimal solution of J.

Conversely, suppose $f = \langle \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{l}^+, \boldsymbol{l}^-, h_m, \boldsymbol{h} \rangle$ is an optimal solution in J, and by the second half of the proof of Theorem 6.3.16, let M be the load-max-balanced stable matching corresponding to f in I. Assume for contradiction that there exists some other load-max-balanced stable matching M' such that $r_s(M') < r_s(M)$. Then by the first half of the proof of Theorem 6.3.16 there must be some other solution $g = \langle \boldsymbol{x}', \boldsymbol{\alpha}', \boldsymbol{\beta}', \boldsymbol{l}^{+\prime}, \boldsymbol{l}^{-\prime}, h'_m, \boldsymbol{h}' \rangle$ of J such that $\sum_{l_k \in L} h'_k = r_s(M') < r_s(M) = \sum_{l_k \in L} h_k$, which contradicts the fact that f is optimal. Hence an optimal solution of J corresponds to a load-balanced stable matching M in I.

Therefore, a load-balanced stable matching M corresponds to an optimal solution in J and vice versa. □

6.4 Conclusions and future work

In this chapter we first presented experimental results evaluating the performance of the $\frac{3}{2}$-approximation algorithm introduced in Chapter 5. We found that, in all instances tested, the approximation algorithm constructed a stable matching within 92% of the size of optimal, easily surpassing the $\frac{3}{2}$ bound. Secondly, we presented two new efficient algorithms for finding a load-sum-balanced matching and a load-balanced matching, and showed how the second of these can be adapted to find a maximum load-max-balanced matching in polynomial time. It remains open as to whether faster algorithms exist to find matchings of these

types, and future work may also involve testing the performance of the algorithms empirically. Finally, we showed that the problems of finding a load-max-balanced stable matching, load-sum-balanced stable matching and load-balanced stable matching are NP-hard.

We define the *Hospitals/Residents problem with Ties and Hospital targets* (HRTH) as a special case of HRT in which hospitals have a target number of residents they wish to be allocated. The NP-completeness proofs given in Section 6.3.4 for the decision problem variants of finding a load-max-balanced stable matching, load-sum-balanced stable matching and load-balanced stable matching may be easily adapted to the HRTH setting. It remains open to derive approximability results for stable matchings of these types, in either HRTH or SPA-STL. Additionally, future work may involve implementing the IP models described in this chapter, in order to determine the effect of varying instance parameters such as instance size and probability of ties on various load balancing properties.

Chapter 7

Conclusions and open problems

In this chapter, we highlight the main contributions of this thesis and discuss potential future theoretical directions.

Two new notions of fairness for degree-based stable matchings in SMI were presented in Chapter 3. These comprised the regret-equal stable matching, which is a stable matching that minimises the difference in degrees between the set of men and the set of women, and the min-regret sum stable matching, which is the stable matching that minimises the sum of degrees of the set of men and the set of women. Polynomial-time algorithms to find such fair stable matchings were developed, and the performance of two algorithms to find a regret-equal stable matching was compared against an existing exponential-time enumeration algorithm over randomly-generated instances. Additional experiments compared the properties of several optimal stable matchings.

This work may be extended in a number of ways. Recall that for instances of SMI, the set of men and the set of women have a degree and cost measure associated with them. The existing four definitions of fair stable matchings, along with the additional two described above, complete the table of cost- and degree-based fairness definitions, summarised in Table 3.1. This work may be extended by investigating new measures, such as the square of costs or the square of degrees, creating new columns in Table 3.1. Further measures may take into account other aspects of how men and women are ranked. An example of this may be a measure that, when all men and women are listed side by side in order of matching rank, calculates the difference in rank for each side-by-side pair. Optimisations may then include minimising the maximum of these differences or the sum of these differences. Finding optimal stable matchings of these types in more general matching problems is also of interest. As described in Section 3.8, it is an open problem as to whether there exists a polynomial-time algorithm to find the equivalent of a sex-equal stable matching in instances of SMI-GW and HR-GR. It would also be of interest to determine the complexity of finding various fair stable matchings in an instance of SR in which roommates were partitioned into groups.

Student preferences:	Project details:	Lecturer preferences:	
s_1: $(p_1\ \underline{\mathbf{p_2}}\ p_3)$	p_1: lecturer l_1, $c_1 = 1$	l_1: $\underline{s_2}\ (s_1\ \mathbf{s_3})$	$d_1 = 1$
s_2: $(\underline{p_1}\ p_2\ \mathbf{p_3})$	p_2: lecturer l_2, $c_2 = 1$	l_2: $s_2\ \underline{\mathbf{s_1}}\ s_3$	$d_2 = 1$
s_3: $\underline{p_3}\ \mathbf{p_1}\ p_2$	p_3: lecturer l_3, $c_3 = 1$	l_3: $(\underline{s_3}\ \mathbf{s_2})\ s_1$	$d_3 = 1$

Figure 7.1: Example of a blocking coalition $C = \{s_2, s_3\}$ relating to stable matching M shown in bold, in an instance I_0 of SPA-ST.

In Chapter 4, we investigated profile-based stable matchings in SMI. In particular we adapted an existing algorithm to find rank-maximal and generous stable matchings, to work with polynomially-bounded weight vectors rather than exponential weights. This vector-based algorithm was shown experimentally to have a far reduced memory requirement than its exponential weight counterpart. It would be interesting to determine whether improvements could be made to the time complexity of this algorithm by showing how vector-based calculations could work with Orlin's [64] Max Flow algorithm. Additionally, it remains open as to whether Feder [15]'s faster algorithm for finding a rank-maximal stable matching, based on weighted SAT, can be adapted to work in a vector-based setting. In this chapter we additionally showed that it is NP-hard to find rank-maximal or generous stable matching in SR, even in restricted cases.

In Chapter 5 and the first half of Chapter 6 we presented a $\frac{3}{2}$-approximation algorithm for MAX-SPA-ST, a new IP model for MAX-SPA-ST and evaluations of our approximation algorithm. This work may be extended by developing an algorithm with an improved approximation factor, or by improving the current best inapproximability lower bound of $\frac{33}{29}$ [74]. A new direction of research may be to investigate *blocking coalitions*, in which the assignments of a series of students are permuted in order to give benefit to some students and lecturers without harming others. An example of a blocking coalition may be seen in Figure 7.1. In this instance of SPA-ST, stable matching $M = \{(s_1, p_2), (s_2, p_3), (s_3, p_1)\}$ is shown in bold, and $C = \{s_2, s_3\}$ is a blocking coalition. Permuting the assignments of students s_2 and s_3 in C results in the stable matching $M' = \{(s_1, p_2), (s_2, p_1), (s_3, p_3)\}$, underlined in Figure 7.1, in which one student and one lecturer improve their rank in M' compared to M and no student or lecturer worsens their rank. It is possible to find a stable and coalition-free matching in polynomial time by first finding a stable matching and then finding and satisfying blocking coalitions repeatedly [12]. It would be interesting to determine whether there exists a faster direct polynomial-time algorithm to find a stable and coalition-free matching in this setting.

Finally, in the second half of Chapter 6, we investigated lecturer load balancing in a new area, namely SPA-STL. We defined new notions of what it means to have a load-balanced matching and presented polynomial-time algorithms to find such matchings. Similar to the definitions of fairness in Chapter 3, we may wish to define new measures associated with lecturer load balancing such as the square of differences between a lecturer's target and their

number of allocations, and determine whether matchings that minimise such measures can be found in polynomial time. We additionally showed that in the presence of stability, load-balanced matchings are NP-hard to find and presented new IP models for these problems. As mentioned in Section 6.4, it remains open to develop approximation algorithms, or prove inapproximability results for finding load-balanced stable matchings in either SPA-STL or HRTH.

Bibliography

[1] A. Abdulkadiroğlu and T. Sönmez. Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica*, 66(3):689–701, 1998.

[2] A. Abdulkadiroğlu, P.A. Pathak, A.E. Roth, and T. Sönmez. The Boston public school match. *American Economic Review*, 95(2):368–371, 2005.

[3] D.J. Abraham, R.W. Irving, and D.F. Manlove. The Student-Project Allocation Problem. In *Proceedings of ISAAC '03: the 14th Annual International Symposium on Algorithms and Computation*, volume 2906 of *Lecture Notes in Computer Science*, pages 474–484. Springer, 2003.

[4] D.J. Abraham, K. Cechlárová, D.F. Manlove, and K. Mehlhorn. Pareto optimality in house allocation problems. In *Proceedings of ISAAC '04: the 15th Annual International Symposium on Algorithms and Computation*, volume 3341 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2004.

[5] D.J. Abraham, R.W. Irving, T. Kavitha, and K. Mehlhorn. Popular matchings. In *Proceedings of SODA '05: the 16th ACM-SIAM Symposium on Discrete Algorithms*, pages 424–432. ACM-SIAM, 2005.

[6] D.J. Abraham, R.W. Irving, and D.F. Manlove. Two algorithms for the Student-Project allocation problem. *Journal of Discrete Algorithms*, 5(1):79–91, 2007. Preliminary version appeared as [3].

[7] V. Bansal, A. Agrawal, and V.S. Malhotra. Polynomial time algorithm for an optimal stable assignment with multiple partners. *Theoretical Computer Science*, 379(3):317–328, 2007.

[8] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, 1996.

[9] C.T. Cheng. The generalized median stable matchings: Finding them is not that easy. In *Proceedings of LATIN '08: the 8th Latin-American Theoretical INformatics symposium*, volume 4957 of *Lecture Notes in Computer Science*, pages 568–579. Springer, 2008.

[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill/MIT, 3rd edition, 2009.

[11] Y.A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.*, 11(5):1277–1280, 1970.

[12] A. Erdil and H. Erkin. What's the matter with tie-breaking? Improving efficiency in school choice. *American Economic Review*, 98:669–689, 2008.

[13] T. Feder. A new fixed point approach for stable networks and stable marriages. In *Proceedings of STOC '89: the 21st ACM Symposium on Theory of Computing*, pages 513–522. ACM, 1989.

[14] T. Feder. *Stable Networks and Product Graphs*. PhD thesis, Stanford University, 1990. Published in *Memoirs of the American Mathematical Society*, vol. 116, no. 555, 1995.

[15] T. Feder. A new fixed point approach for stable networks and stable marriages. *Journal of Computer and System Sciences*, 45:233–284, 1992. Preliminary version appeared as [13].

[16] T. Feder. Network flow and 2-satisfiability. *Algorithmica*, 11(3):291–319, 1994.

[17] L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[18] H.N. Gabow and R.E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18:1013–1036, 1989.

[19] D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[20] D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985.

[21] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[22] S. Gupta, S. Roy, S. Saurabh, and M. Zehavi. Balanced stable marriage: How close is close enough? In *Proceedings of WADS '19: the 16th Algorithms and Data Structures Symposium*, Lecture Notes in Computer Science, pages 423–437. Springer, 2019.

[23] D. Gusfield. Three fast algorithms for four problems in stable marriage. *SIAM Journal on Computing*, 16(1):111–128, 1987.

[24] D. Gusfield. The structure of the stable roommate problem – efficient representation and enumeration of all stable assignments. *SIAM Journal on Computing*, 17(4):742–769, 1988.

[25] D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.

[26] C.-C. Huang, T. Kavitha, K. Mehlhorn, and D. Michail. Fair matchings and related problems. *Algorithmica*, 74:1184–1203, 2016.

[27] A. Hylland and R. Zeckhauser. The efficient allocation of individuals to positions. *Journal of Political Economy*, 87(2):293–314, 1979.

[28] IBM. CPLEX optimizer. `https://www.ibm.com/analytics/cplex-optimizer`, 2019.

[29] R.W. Irving. An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6:577–595, 1985.

[30] R.W. Irving and P. Leather. The complexity of counting stable marriages. *SIAM Journal on Computing*, 15(3):655–667, 1986.

[31] R.W. Irving and D.F. Manlove. The Stable Roommates Problem with Ties. *Journal of Algorithms*, 43:85–105, 2002.

[32] R.W. Irving, P. Leather, and D. Gusfield. An efficient algorithm for the "optimal" stable marriage. *Journal of the ACM*, 34(3):532–543, 1987.

[33] R.W. Irving, D.F. Manlove, and S. Scott. The Hospitals/Residents problem with Ties. In *Proceedings of SWAT '00: the 7th Scandinavian Workshop on Algorithm Theory*, volume 1851 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2000.

[34] K. Iwama, S. Miyazaki, and H. Yanagisawa. Approximation algorithms for the sex-equal stable marriage problem. In *Proceedings of WADS '07: the 10th International Workshop on Algorithms and Data Structures*, volume 4619 of *Lecture Notes in Computer Science*, pages 201–213. Springer, 2007.

[35] K. Iwama, S. Miyazaki, and H. Yanagisawa. Approximation algorithms for the sex-equal stable marriage problem. *ACM Transactions on Algorithms*, 7(1), 2010. Article number 2. Preliminary version appeared as [34].

[36] K. Iwama, S. Miyazaki, and H. Yanagisawa. Improved approximation bounds for the student-project allocation problem with preferences over projects. In *Proceedings of*

TAMC '11: the 8th Annual Conference on Theory and Applications of Models of Computation, volume 6648 of *Lecture Notes in Computer Science*, pages 440–451. Springer, 2011.

[37] K. Iwama, S. Miyazaki, and H. Yanagisawa. Improved approximation bounds for the student-project allocation problem with preferences over projects. *Journal of Discrete Algorithms*, 13:59–66, 2012. Preliminary version appeared as [36].

[38] A. Kato. Complexity of the sex-equal stable marriage problem. *Japan Journal of Industrial and Applied Mathematics*, 10:1–19, 1993.

[39] T. Kavitha, K. Mehlhorn, D. Michail, and K. Paluch. Strongly stable matchings in time $O(nm)$ and extension to the Hospitals-Residents problem. In *Proceedings of STACS '04: the 21st International Symposium on Theoretical Aspects of Computer Science*, volume 2996 of *Lecture Notes in Computer Science*, pages 222–233. Springer, 2004.

[40] Z. Király. Linear time local approximation algorithm for maximum stable marriage. In *Proceedings of MATCH-UP '12: the 2nd International Workshop on Matching Under Preferences*, pages 99–110, 2012.

[41] D.E. Knuth. *Stable Marriage and its Relation to Other Combinatorial Problems*, volume 10 of *CRM Proceedings and Lecture Notes*. American Mathematical Society, 1997. English translation of *Mariages Stables*, Les Presses de L'Université de Montréal, 1976.

[42] A. Kunysz. The strongly stable Roommates problem. In *Proceedings of ESA '16: the 24th Annual European Symposium on Algorithms*, number 60 in Leibniz International Proceedings in Informatics, pages 1–15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[43] B. Lazarov. Matching Algorithm Toolkit web application. `https://matwa.optimalmatching.com`, 2019. [Online; accessed 25 May 2020].

[44] C. Lennon and B. Pittel. On the likely number of solutions for the Stable Marriage problem. *Combinatorics, Probability and Computing*, 18:371–421, 2009.

[45] D. Maier and J.A. Storer. A note on the complexity of the superstring problem. Technical Report 233, Dept. Electrical Engineering and Computer Science, Princeton University, USA, 1977.

[46] D.F. Manlove. Stable marriage with ties and unacceptable partners. Technical Report TR-1999-29, University of Glasgow, Department of Computing Science, January 1999.

[47] D.F. Manlove. *Algorithmics of Matching Under Preferences*. World Scientific, 2013.

[48] D.F. Manlove and G. O'Malley. Student project allocation with preferences over projects. In *Proceedings of ACiD '05: the 1st Algorithms and Complexity in Durham workshop*, volume 4 of *Texts in Algorithmics*, pages 69–80. KCL Publications, 2005.

[49] D.F. Manlove and C.T.S. Sng. Popular matchings in the Capacitated House Allocation problem. In *Proceedings of ESA '06: the 14th Annual European Symposium on Algorithms*, volume 4168 of *Lecture Notes in Computer Science*, pages 492–503. Springer, 2006.

[50] D.F. Manlove, R.W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. Technical Report TR-1999-43, University of Glasgow, School of Computing Science, 1999.

[51] D.F. Manlove, R.W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276(1-2):261–279, 2002. Full version available as [50].

[52] D.F. Manlove, I. McBride, and J. Trimble. "Almost-stable" matchings in the Hospitals / Residents problem with couples. *Constraints*, 22:50–72, 2017.

[53] E. McDermid. A 3/2 approximation algorithm for general stable marriage. In *Proceedings of ICALP '09: the 36th International Colloquium on Automata, Languages and Programming*, volume 5555 of *Lecture Notes in Computer Science*, pages 689–700. Springer, 2009.

[54] E. McDermid and R.W. Irving. Popular matchings: Structure and algorithms. In *Proceedings of COCOON '09: the 15th Annual International Computing and Combinatorics Conference*, volume 5609 of *Lecture Notes in Computer Science*, pages 506–515. Springer, 2009.

[55] E. McDermid and R.W. Irving. Popular matchings: Structure and algorithms. *Journal of Combinatorial Optimization*, 22(3):339–358, 2011. Preliminary version appeared as [54].

[56] E. McDermid and R.W. Irving. Sex-equal stable matchings: Complexity and exact algorithms. *Algorithmica*, 68:545–570, 2014.

[57] E.J. McDermid and D.F. Manlove. Keeping partners together: Algorithmic results for the hospitals / residents problem with couples. *Journal of Combinatorial Optimization*, 19(3):279–303, 2010.

[58] D.G. McVitie and L.B. Wilson. The stable marriage problem. *Communications of the ACM*, 14(7):486–490, 1971.

[59] S. Mitchell, M. O'Sullivan, and I. Dunning. PuLP: A linear programming toolkit for Python. *Optimization Online*, 2011.

[60] S. Olaosebikan and D.F. Manlove. Super-stability in the Student-Project Allocation problem with ties. In *Proceedings. COCOA '18: the 12th International Conference on Combinatorial Optimization and Applications*, Lecture Notes in Computer Science, pages 357–371. Springer, 2018.

[61] S. Olaosebikan and D.F. Manlove. An algorithm for strong stability in the Student-Project Allocation problem with ties. In *Proceedings of CALDAM '20: the 6th International Conference on Algorithms and Discrete Applied Mathematics*, Lecture Notes in Computer Science, pages 384–399. Springer, 2020.

[62] Oracle. Class BigInteger. `https://docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html`, 2018. [Java version 8. Online; accessed 07 August 2018].

[63] Oracle. Primitive data types. `https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html`, 2019. [Java version 8. Online; accessed 13 April 2020].

[64] J.B. Orlin. Max flows in O(nm) time, or better. In *Proceedings of STOC '13: 45th annual ACM symposium on Theory of computing*, pages 765–774. ACM, 2013.

[65] E. Peranson and R.R. Randlett. The NRMP matching algorithm revisited: Theory versus practice. *Academic Medicine*, 70(6):477–484, 1995.

[66] E. Ronn. NP-complete stable matching problems. *Journal of Algorithms*, 11:285–304, 1990.

[67] A.E. Roth. The evolution of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy*, 92(6):991–1016, 1984.

[68] A.E. Roth. On the allocation of residents to rural hospitals: a general property of two-sided matching markets. *Econometrica*, 54:425–427, 1986.

[69] A.E. Roth and M.A.O. Sotomayor. *Two-Sided Matching: a Study in Game-Theoretic Modeling and Analysis*, volume 18 of *Econometric Society Monographs*. Cambridge University Press, 1990.

[70] A.E. Roth and X. Xing. Jumping the gun: imperfections and institutions related to the timing of market transactions. *American Economic Review*, 84(4):992–1044, 1994.

[71] D.D. Sleator and R.E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[72] O. Tange. GNU parallel - the command-line power tool. *The USENIX Magazine*, pages 42–47, 2011.

[73] C.-P. Teo and J. Sethuraman. The geometry of fractional stable matchings and its applications. *Mathematics of Operations Research*, 23(4):874–891, 1998.

[74] H. Yanagisawa. *Approximation Algorithms for Stable Marriage Problems*. PhD thesis, Kyoto University, School of Informatics, 2007.

[75] Y. Zhang. *The determinants of national college entrance exam performance in China - with an analysis of private tutoring*. PhD thesis, Columbia University, 2011.

[76] L. Zhou. On a conjecture by Gale about one-sided matching problems. *Journal of Economic Theory*, 52(1):123–135, 1990.

Appendix A

Degree-based stable matchings in SMI – supplementary material

A.1 Experimental work supplement

This section contains tables of results for experiments conducted on instances of SMI that are referred to in Section 3.7.

Case	DI_{av}	DI_{med}	DI_5	DI_{95}	SP_{av}	SP_{med}	SP_5	SP_{95}	EN_{av}	EN_{med}	EN_5	EN_{95}
S10	204.9	206.0	180.0	226.0	204.6	205.0	177.0	228.0	244.8	244.0	215.9	270.0
S20	214.3	215.0	191.0	236.0	217.6	218.0	187.0	245.0	256.1	257.0	227.9	283.0
S30	220.7	221.5	195.0	243.0	234.7	234.0	205.0	264.0	264.0	265.5	236.0	288.0
S40	234.1	235.0	208.0	255.0	259.8	257.0	218.0	307.0	279.9	281.0	248.0	305.0
S50	237.1	238.0	213.0	259.0	288.4	282.0	233.0	365.1	288.7	289.0	260.0	317.0
S60	252.5	253.5	229.0	276.0	335.5	321.5	249.9	460.4	311.3	312.0	281.0	340.0
S70	261.3	262.0	234.0	284.0	378.5	353.0	264.9	559.1	320.5	321.0	285.0	353.0
S80	269.7	270.0	246.0	294.0	460.4	417.5	284.0	782.3	334.5	334.0	296.9	373.0
S90	288.4	290.0	262.0	312.0	548.5	491.0	303.9	1006.1	360.9	360.0	325.0	400.0
S100	290.7	291.0	259.9	321.0	654.7	539.0	317.0	1341.3	369.3	369.5	329.0	412.0
S200	362.4	354.0	315.0	441.0	4272.8	3505.5	481.8	11161.4	533.5	514.0	420.9	695.3
S300	447.1	440.0	425.0	494.0	18866.5	13587.5	728.7	56383.6	789.1	751.0	606.9	1047.1
S400	541.3	540.0	520.0	562.0	54243.1	39662.5	4283.5	171349.4	1278.7	1127.0	891.9	2136.6
S500	697.6	698.0	662.9	734.0	159924.0	112320.5	7650.9	491541.9	1961.7	1761.5	1201.4	3156.3
S600	965.6	968.0	925.9	1010.0	369995.5	253337.0	19624.1	1043107.8	2835.6	2546.0	1767.9	4707.9
S700	1154.3	1160.0	1080.0	1219.0	745791.4	518138.0	11246.6	2256581.1	3986.8	3194.5	2218.7	7620.5
S800	1396.7	1402.0	1312.0	1485.0	1302405.0	956639.0	76865.6	3600000.0	5708.3	4496.5	2801.7	12450.0
S900	1701.9	1712.0	1585.9	1836.0	1868537.4	1781841.0	118124.7	3600000.0	9043.3	5958.0	3439.9	20366.2
S1000	2087.9	2112.0	1855.0	2293.0	2220264.5	2449492.0	152354.6	3600000.0	11802.0	8092.5	4234.8	28824.8

Table A.1: A comparison of time taken to execute Algorithm REDI, Algorithm RESP and Algorithm ENUM. We abbreviate the algorithms as follows: Algorithm REDI (DI), Algorithm RESP (SP) and Algorithm ENUM (EN). Here DI_{av}, DI_{med}, DI_5 and DI_{95} represent the mean, median, 5th percentile and 95th percentile of the time taken to execute Algorithm REDI for a given instance type. Similar notation is used for Algorithms RESP and ENUM. Times are in ms.

Case	Balanced	Sex-equal	Egalitar-ian	Minimum regret	Regret-equal	Min-regret sum	Algorithm REDI	Algorithm RESP
S10	32.1	32.1	33.3	32.8	32.8	35.6	33.4	33.4
S20	90.5	90.9	95.1	94.2	94.0	104.5	96.9	96.7
S30	165.6	166.2	177.5	174.9	176.5	199.4	181.5	181.4
S40	254.9	255.8	273.0	270.7	270.2	312.7	278.4	278.2
S50	357.2	358.3	382.0	378.9	379.5	439.8	393.0	393.3
S60	466.4	467.7	495.5	495.5	496.5	573.3	516.2	515.6
S70	588.8	590.8	626.5	626.2	629.1	739.3	651.4	651.4
S80	720.1	722.3	769.7	764.9	769.9	901.8	798.9	797.8
S90	861.1	863.3	921.7	906.6	914.9	1054.9	952.7	951.4
S100	1004.7	1007.2	1073.4	1062.7	1063.1	1245.0	1103.2	1103.3
S200	2844.8	2849.2	3000.2	3014.6	3008.9	3553.4	3134.9	3131.4
S300	5224.1	5230.1	5510.4	5488.6	5550.5	6348.2	5741.3	5736.6
S400	8036.5	8045.4	8471.1	8503.0	8541.4	9743.7	8835.2	8837.8
S500	11215.7	11223.6	11740.4	11891.9	11857.3	13577.3	12352.0	12352.5
S600	14757.4	14770.0	15423.7	15474.0	15543.2	18188.9	16061.1	16070.5
S700	18576.7	18590.2	19407.7	19525.4	19553.7	22512.5	20217.5	20193.7
S800	22718.1	22731.4	23707.2	23851.3	23975.3	27652.9	24824.3	24766.9
S900	27098.2	27113.3	28198.3	28678.0	28667.4	32719.2	29707.8	29625.7
S1000	31684.8	31702.0	32976.9	33364.7	33393.2	38599.1	34551.0	34468.1

Table A.2: Mean balanced score for six different optimal stable matchings and outputs from Algorithms REDI and RESP.

Case	Balanced	Sex-equal	Egalitar-ian	Minimum regret	Regret-equal	Min-regret sum	Algorithm REDI	Algorithm RESP
S10	5.4	5.3	8.9	7.0	6.5	12.2	7.6	7.6
S20	10.5	10.0	22.7	17.8	16.7	36.2	21.6	21.4
S30	13.9	13.1	43.7	32.1	32.6	75.8	41.2	41.3
S40	17.4	16.1	63.2	48.8	44.0	121.8	58.7	58.5
S50	22.6	21.5	84.3	65.1	63.2	169.8	87.3	87.8
S60	25.2	23.1	98.8	80.7	77.8	213.6	113.1	112.4
S70	27.9	25.5	121.4	99.8	98.8	290.9	139.2	139.4
S80	29.8	27.6	149.6	118.0	118.5	347.5	170.1	168.8
S90	36.8	33.5	182.8	125.6	134.0	378.8	201.5	199.9
S100	36.0	32.6	200.8	150.1	141.1	456.9	213.7	214.6
S200	71.9	66.4	440.8	399.6	369.1	1306.0	598.0	593.3
S300	101.9	92.1	762.3	616.6	709.7	2067.9	1060.2	1053.4
S400	135.3	126.2	1117.8	1017.1	1061.0	3131.9	1595.2	1600.6
S500	154.8	142.5	1345.4	1431.4	1343.2	4322.4	2255.0	2256.4
S600	188.3	173.4	1694.3	1562.5	1641.7	6111.8	2596.3	2611.7
S700	223.3	207.4	2071.3	2014.9	2053.9	7077.1	3268.0	3223.0
S800	233.0	215.2	2437.2	2388.6	2597.4	8836.2	4164.2	4072.5
S900	256.0	240.1	2705.8	3227.4	3179.2	10125.8	5077.3	4948.9
S1000	284.0	265.0	3147.4	3438.4	3454.8	12400.0	5583.6	5421.9

Table A.3: Mean sex-equal score for six different optimal stable matchings and outputs from Algorithms REDI and RESP.

Case	Balanced	Sex-equal	Egalitar-ian	Minimum regret	Regret-equal	Min-regret sum	Algorithm REDI	Algorithm RESP
S10	58.6	59.0	57.7	58.3	58.8	58.9	59.1	59.2
S20	170.3	171.7	167.4	169.5	170.7	172.2	172.2	172.1
S30	316.9	319.4	311.3	315.5	319.3	322.5	321.8	321.6
S40	492.1	495.5	482.7	488.7	493.6	502.4	498.0	497.8
S50	691.5	695.0	679.7	687.9	692.6	708.1	698.8	698.7
S60	907.2	912.3	892.1	903.2	909.7	930.1	919.4	918.8
S70	1149.3	1156.1	1131.7	1145.4	1154.5	1185.7	1163.7	1163.4
S80	1410.2	1417.1	1389.8	1403.3	1415.2	1452.9	1427.7	1426.9
S90	1684.8	1693.2	1660.6	1675.1	1687.7	1727.1	1703.8	1702.9
S100	1973.3	1981.8	1945.9	1963.4	1976.0	2028.7	1992.6	1991.9
S200	5617.4	5632.0	5559.5	5601.3	5625.2	5790.9	5671.9	5669.5
S300	10346.3	10368.1	10258.5	10314.5	10358.0	10611.2	10422.3	10419.9
S400	15937.5	15964.5	15824.3	15932.0	15978.1	16332.6	16075.3	16075.0
S500	22276.4	22304.7	22135.5	22276.0	22308.6	22800.7	22449.0	22448.6
S600	29326.0	29366.7	29153.0	29297.7	29371.4	30233.8	29525.9	29529.3
S700	36929.9	36972.9	36744.2	36933.6	36975.4	37910.1	37167.1	37164.3
S800	45203.0	45247.6	44977.2	45195.9	45264.7	46423.0	45484.3	45461.3
S900	53940.0	53986.4	53690.7	54003.9	54053.3	55265.3	54338.2	54302.4
S1000	63085.6	63139.0	62806.5	63145.7	63204.4	64752.1	63518.5	63514.3

Table A.4: Mean cost for six different optimal stable matchings and outputs from Algorithms REDI and RESP.

Case	Balanced	Sex-equal	Egalitar-ian	Minimum regret	Regret-equal	Min-regret sum	Algorithm REDI	Algorithm RESP
S10	7.9	7.9	7.8	7.6	7.7	7.8	7.7	7.7
S20	14.4	14.5	14.3	13.5	13.8	14.1	13.8	13.8
S30	21.0	21.4	20.7	19.3	19.8	20.2	19.9	19.8
S40	26.7	27.1	26.1	24.3	25.1	25.7	25.2	25.1
S50	31.7	32.0	31.2	28.7	29.5	30.4	29.6	29.5
S60	36.6	37.0	35.5	33.1	34.1	35.1	34.2	34.1
S70	41.2	41.9	40.6	37.4	38.5	39.8	38.6	38.5
S80	45.1	45.6	44.6	41.1	42.4	44.0	42.5	42.4
S90	49.1	49.7	48.2	44.6	45.9	47.2	46.1	45.9
S100	53.4	54.1	52.0	48.1	49.7	51.2	49.9	49.8
S200	87.5	88.0	85.2	79.3	82.0	84.8	82.2	82.0
S300	116.5	117.5	114.0	104.3	107.3	110.8	107.3	107.3
S400	139.2	139.6	136.8	125.5	129.1	133.4	129.3	129.1
S500	160.9	161.2	159.8	146.7	151.1	154.4	151.1	151.1
S600	182.1	182.6	178.7	166.0	170.8	177.0	170.9	170.7
S700	198.5	198.9	197.4	181.8	186.8	191.9	186.8	186.7
S800	217.7	218.6	214.3	197.9	202.2	210.7	202.5	199.8
S900	239.6	239.4	235.7	215.6	222.4	228.7	222.5	216.0
S1000	252.7	253.7	252.2	232.5	239.5	246.3	239.5	232.8

Table A.5: Mean degree for six different optimal stable matchings and outputs from Algorithms REDI and RESP.

Case	Balanced	Sex-equal	Egalitar-ian	Minimum regret	Regret-equal	Min-regret sum	Algorithm REDI	Algorithm RESP
S10	1.8	1.8	2.2	1.6	1.5	2.7	1.5	1.5
S20	3.2	3.3	4.1	2.6	2.2	5.2	2.2	2.2
S30	5.1	5.2	6.3	3.5	2.9	8.2	2.9	2.9
S40	6.2	6.3	7.5	4.3	3.2	10.6	3.2	3.2
S50	7.2	7.2	8.5	4.7	3.6	12.3	3.6	3.6
S60	8.2	8.4	9.4	5.5	4.0	14.1	4.0	4.0
S70	9.2	9.5	11.0	6.1	4.0	16.5	4.0	4.0
S80	9.5	9.6	11.9	6.4	4.3	18.1	4.3	4.3
S90	10.1	10.4	12.0	6.0	4.2	18.1	4.2	4.2
S100	11.1	11.2	13.1	7.1	4.7	20.1	4.7	4.7
S200	16.5	16.9	18.4	10.7	6.9	33.4	6.9	6.9
S300	23.7	24.0	25.7	13.3	8.7	39.6	8.7	8.7
S400	25.4	25.7	28.3	14.9	9.0	46.0	9.0	9.0
S500	28.0	27.9	30.8	17.9	10.8	52.3	10.8	10.8
S600	31.0	30.9	34.5	18.6	11.3	63.7	11.3	11.1
S700	32.2	32.3	36.1	20.4	12.2	63.2	12.2	12.0
S800	37.0	37.2	39.0	21.1	13.9	71.3	13.9	11.0
S900	42.7	41.8	45.5	24.7	14.3	77.0	14.3	8.1
S1000	40.4	40.1	45.9	25.9	14.2	84.6	14.2	6.8

Table A.6: Mean regret-equality score for six different optimal stable matchings and output from Algorithms REDI and RESP.

Case	Balanced	Sex-equal	Egalitar-ian	Minimum regret	Regret-equal	Min-regret sum	Algorithm REDI	Algorithm RESP
S10	13.8	14.0	13.3	13.1	13.9	12.9	13.9	13.9
S20	25.5	25.8	24.5	23.6	25.3	22.9	25.5	25.4
S30	36.9	37.6	35.0	33.4	36.8	32.3	36.9	36.8
S40	47.2	47.8	44.6	42.3	47.0	40.8	47.1	47.0
S50	56.3	56.7	53.7	50.2	55.3	48.4	55.5	55.4
S60	64.9	65.6	61.5	57.9	64.2	56.0	64.4	64.2
S70	73.3	74.3	70.1	65.6	73.0	63.1	73.1	73.1
S80	80.7	81.6	77.0	72.5	80.4	69.8	80.8	80.4
S90	88.1	89.0	84.3	79.2	87.6	76.3	87.9	87.6
S100	95.7	96.9	90.8	85.4	94.8	82.3	95.1	94.9
S200	158.4	159.2	151.7	142.1	157.0	136.3	157.4	157.1
S300	209.4	210.9	202.2	187.6	205.8	182.0	206.0	205.9
S400	252.9	253.4	245.3	227.6	249.2	220.7	249.6	249.2
S500	293.8	294.5	288.8	265.2	291.4	256.5	291.5	291.4
S600	333.1	334.2	322.8	300.9	330.3	290.3	330.5	330.2
S700	364.8	365.6	358.8	330.0	361.3	320.6	361.4	361.4
S800	398.3	400.0	389.6	363.3	390.5	350.0	391.1	388.6
S900	436.5	436.9	426.0	391.4	430.5	380.3	430.8	424.0
S1000	465.0	467.3	458.5	420.9	464.7	408.0	464.9	458.7

Table A.7: Mean regret sum for six different optimal stable matchings and outputs from Algorithms REDI and RESP.

Case	Balanced	Sex-equal	Egalitarian	Minimum regret	Regret-equal	Min-regret sum
S10	1.1	1.0	1.1	1.6	1.3	1.4
S20	1.0	1.0	1.1	2.1	1.7	1.7
S30	1.0	1.0	1.1	2.7	1.9	1.8
S40	1.0	1.0	1.1	3.3	2.3	2.1
S50	1.0	1.0	1.1	3.9	2.7	2.4
S60	1.0	1.0	1.1	4.7	3.3	3.0
S70	1.0	1.0	1.0	4.9	3.3	2.8
S80	1.0	1.0	1.1	5.4	3.5	3.3
S90	1.0	1.0	1.0	6.3	4.3	3.3
S100	1.0	1.0	1.1	7.1	4.4	3.7
S200	1.0	1.0	1.0	11.2	7.1	6.3
S300	1.0	1.0	1.0	16.3	9.7	8.8
S400	1.0	1.0	1.0	18.1	11.9	10.3
S500	1.0	1.0	1.0	23.7	15.8	13.1
S600	1.0	1.0	1.0	30.0	17.5	15.2
S700	1.0	1.0	1.0	33.2	20.3	19.0
S800	1.0	1.0	1.0	35.1	21.2	16.4
S900	1.0	1.0	1.0	48.6	28.2	21.2
S1000	1.0	1.0	1.0	47.7	24.5	20.2

Table A.8: Mean number of optimal stable matchings per instance.

Case	0	1	2	3	4	5	6
S10	0.7	0.4	0.5	0.4	0.3	0.3	0.4
S20	3.2	1.1	1.0	0.6	0.3	0.2	0.2
S30	5.9	1.7	1.4	0.6	0.4	0.2	0.1
S40	10.6	2.3	1.8	0.8	0.4	0.2	0.1
S50	14.7	2.8	2.1	0.9	0.4	0.2	0.0
S60	20.0	3.6	2.6	1.0	0.3	0.1	0.0
S70	25.0	4.1	2.7	0.9	0.3	0.1	0.0
S80	32.0	4.2	3.3	0.9	0.3	0.1	0.0
S90	37.9	5.1	3.6	0.9	0.3	0.1	0.0
S100	43.8	5.8	3.5	1.2	0.3	0.1	0.0
S200	121.5	9.9	6.0	1.4	0.3	0.1	0.0
S300	194.4	14.1	8.4	1.8	0.3	0.1	0.0
S400	319.5	17.3	9.2	2.2	0.2	0.1	0.0
S500	404.3	23.5	12.5	2.0	0.2	0.0	0.0
S600	501.3	26.4	16.3	1.8	0.3	0.0	0.0
S700	620.7	29.1	16.0	4.4	0.2	0.0	0.0
S800	765.0	28.8	17.7	3.4	0.3	0.0	0.0
S900	905.8	44.5	23.8	2.7	0.2	0.0	0.0
S1000	1008.5	45.8	20.3	2.8	0.2	0.0	0.0

Table A.9: Mean number of stable matchings that satisfy c optimality criteria, where c varies on the x-axis.

Appendix B

Profile-based stable matchings in SMI – supplementary material

B.1 Experimental work supplement

This section contains tables of results for the experiments conducted in Section 4.7.

	Cost			Sex-equal score		
Case	Min	Max	Mean	Min	Max	Mean
S10	40	78	58.0	0	41	5.2
S20	113	215	167.2	0	71	9.9
S30	243	371	311.2	0	137	13.6
S40	396	572	484.3	0	130	16.1
S50	567	821	679.5	0	218	19.8
S60	730	1057	896.2	0	310	23.3
S70	955	1299	1133.2	0	255	24.2
S80	1164	1609	1390.0	0	217	27.2
S90	1447	1909	1660.9	0	178	31.6
S100	1663	2179	1947.0	0	314	33.2
S200	4865	6257	5554.9	0	411	62.0
S300	9444	11301	10250.0	0	528	93.4
S400	14755	16999	15847.1	0	885	116.3
S500	20610	23767	22137.6	0	1158	136.3
S600	27502	31147	29147.8	0	1311	170.8
S700	35117	38975	36772.6	0	1352	206.2
S800	42372	47962	44930.2	0	1809	230.0
S900	50650	56289	53658.2	0	1812	251.3
S1000	59776	65571	62875.9	0	2295	269.4

Table B.1: Optimal costs and sex-equal scores.

	f			l_{10}			Degree			Cost			Sex-equal score		
Case	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
S10	0	13	6.9	0.0	3.0	0.4	4	10	8.6	40	87	60.4	0	63	16.5
S20	2	19	10.6	0.0	4.0	0.4	9	20	17.0	124	275	182.3	0	205	64.8
S30	6	22	13.5	0.0	4.0	0.5	12	30	25.3	247	496	349.3	0	374	142.4
S40	6	26	16.0	0.0	4.0	0.6	18	40	34.0	407	810	564.2	1	650	265.6
S50	7	32	18.2	0.0	6.0	0.6	22	50	42.4	604	1150	809.7	1	902	405.0
S60	11	35	20.4	0.0	4.0	0.6	25	60	51.4	742	1617	1095.5	13	1332	596.8
S70	11	36	22.6	0.0	7.0	0.6	29	70	60.2	1040	2054	1421.3	2	1730	818.4
S80	11	42	24.6	0.0	4.0	0.7	34	80	68.9	1328	2586	1780.0	3	2106	1067.6
S90	14	43	26.6	0.0	5.0	0.7	41	90	78.4	1501	3156	2186.0	7	2644	1363.4
S100	13	45	28.7	0.0	5.0	0.7	40	100	87.2	1807	3609	2617.4	32	2983	1693.5
S200	24	73	45.8	0.0	7.0	0.9	98	200	177.6	5617	12532	8616.7	1277	11096	6494.2
S300	33	96	61.6	0.0	6.0	1.0	110	300	269.4	10333	24028	17608.0	1677	21562	14213.9
S400	46	110	76.5	0.0	8.0	1.1	230	400	361.3	19149	39409	29521.9	10331	35863	24778.8
S500	55	132	90.4	0.0	7.0	1.1	271	500	454.4	26729	60640	43725.3	15185	56490	37529.6
S600	67	155	105.3	0.0	9.0	1.2	365	600	546.5	41102	81496	61248.3	28874	76436	53676.6
S700	65	162	118.5	0.0	9.0	1.3	396	700	641.9	52098	108000	80778.1	37230	101646	71714.5
S800	77	178	131.4	0.0	8.0	1.3	402	800	732.9	60915	134559	102579.6	40623	126963	91993.6
S900	94	198	144.5	0.0	8.0	1.3	491	900	824.0	86785	183870	127944.3	69419	175936	115909.3
S1000	104	208	158.4	0.0	8.0	1.4	652	1000	921.2	104836	205341	154730.8	83732	195439	141113.6

Table B.2: Results for rank-maximal stable matchings over various measures.

	f			l_{50}			Degree			Cost			Sex-equal score		
Case	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
S10	0	12	6.0	0.0	6.0	2.4	4	10	7.6	40	81	58.8	0	41	8.3
S20	2	17	8.8	0.0	7.0	2.4	8	20	13.8	113	225	170.0	0	93	22.2
S30	3	20	10.9	0.0	7.0	2.2	12	30	19.3	243	396	317.2	0	161	40.7
S40	3	21	12.4	0.0	8.0	1.8	14	40	24.2	397	626	492.8	0	294	60.5
S50	5	28	14.0	0.0	7.0	1.4	18	49	28.7	567	875	691.0	0	398	85.2
S60	6	28	15.2	0.0	7.0	1.2	21	56	33.0	730	1105	910.5	0	470	106.9
S70	6	29	16.8	0.0	6.0	0.9	24	68	37.0	955	1333	1151.6	0	602	140.0
S80	7	28	17.8	0.0	5.0	0.7	24	76	40.6	1164	1683	1411.4	0	670	163.2
S90	9	32	18.9	0.0	7.0	0.5	26	75	44.4	1455	1981	1683.9	0	825	186.4
S100	8	34	20.0	0.0	5.0	0.4	30	74	47.8	1704	2276	1974.6	1	951	219.4
S200	13	55	28.2	0.0	2.0	0.0	51	119	78.5	4865	6375	5622.4	1	2428	566.6
S300	19	55	34.7	0.0	1.0	0.0	70	165	104.3	9460	12079	10366.3	2	6095	1015.7
S400	22	62	40.0	0.0	1.0	0.0	86	203	126.3	14802	17981	16005.4	0	7130	1503.9
S500	24	76	44.8	0.0	0.0	0.0	107	223	147.7	20625	24592	22358.6	2	9214	2056.5
S600	30	71	49.4	0.0	0.0	0.0	124	292	165.9	27514	32564	29406.9	2	11850	2603.3
S700	33	80	52.7	0.0	0.0	0.0	135	264	183.7	35117	39710	37086.4	2	14023	3201.3
S800	35	83	57.3	0.0	0.0	0.0	145	291	200.3	42579	49508	45308.3	0	19508	3883.3
S900	30	81	59.5	0.0	0.0	0.0	160	305	216.4	50957	59447	54104.8	2	22087	4693.8
S1000	40	89	63.5	0.0	0.0	0.0	176	350	230.6	60181	69426	63364.8	2	26076	5163.1

Table B.3: Results for generous stable matchings over various measures.

	f			l_{20}			Degree			Cost			Sex-equal score		
Case	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
S10	0	12	6.1	0.0	3.0	0.5	4	10	8.2	40	79	59.9	0	41	9.5
S20	2	18	8.9	0.0	4.0	0.4	8	20	15.3	113	224	173.7	0	121	27.3
S30	4	20	11.0	0.0	5.0	0.3	12	30	22.0	243	416	323.9	0	228	50.8
S40	3	22	12.6	0.0	3.0	0.3	17	40	28.2	402	693	504.4	0	403	83.8
S50	5	25	14.2	0.0	5.0	0.2	19	50	33.9	570	924	709.6	0	537	123.9
S60	6	30	15.6	0.0	3.0	0.2	23	60	39.8	756	1232	938.0	0	736	173.1
S70	8	30	17.0	0.0	2.0	0.2	25	70	45.0	970	1487	1186.7	0	973	222.8
S80	7	30	18.2	0.0	2.0	0.1	28	80	50.0	1164	1858	1457.6	0	1171	280.2
S90	9	32	19.5	0.0	4.0	0.1	31	90	55.4	1447	2484	1744.8	1	1850	355.7
S100	7	34	20.5	0.0	2.0	0.1	34	100	60.5	1663	2604	2045.8	1	1553	415.6
S200	14	52	29.5	0.0	2.0	0.1	57	199	105.2	5006	8226	5917.3	1	6108	1477.0
S300	21	69	36.8	0.0	3.0	0.0	79	294	144.4	9541	17224	11046.9	4	13374	3115.2
S400	21	72	42.8	0.0	3.0	0.0	99	393	178.1	14934	26115	17149.7	17	20951	5052.7
S500	25	99	48.3	0.0	9.0	0.0	110	496	212.0	20725	48487	24257.5	15	43025	7752.5
S600	31	102	53.5	0.0	4.0	0.0	127	595	244.3	28028	58399	32052.6	14	50581	10459.2
S700	32	113	58.0	0.0	2.0	0.0	141	665	276.7	35634	71039	40774.9	34	60999	13863.8
S800	37	129	63.0	0.0	3.0	0.0	162	797	304.0	42713	105477	50215.3	10	95243	18093.7
S900	36	136	66.0	0.0	2.0	0.0	174	833	331.2	51568	117599	60037.6	46	104663	21334.7
S1000	44	163	71.5	0.0	3.0	0.0	191	975	362.5	60270	155476	71456.3	11	142570	27270.2

Table B.4: Results for median stable matchings over various measures.

		Exponential weight				Vector-based weight			
Case	N_I	Mean	Median	5th	95th	Mean	Median	5th	95th
S10	821	124.9	116.0	43.0	249.0	243.1	228.0	126.0	424.0
S20	970	447.4	420.0	99.0	870.6	573.2	548.0	204.0	1028.0
S30	992	1039.3	1007.0	303.3	1853.3	964.1	952.0	380.0	1589.8
S40	999	1945.7	1905.0	641.0	3294.2	1582.7	1580.0	634.8	2514.4
S50	1000	3107.8	3071.5	1241.8	4959.9	2145.9	2138.0	1035.4	3248.3
S60	1000	4575.3	4534.0	2021.5	7405.2	2733.3	2708.0	1464.0	4168.1
S70	1000	6578.8	6552.5	3251.9	9954.4	3693.3	3662.0	1978.6	5504.5
S80	1000	8757.7	8792.0	4606.9	13191.3	4588.7	4560.0	2639.4	6512.8
S90	1000	11142.1	11133.0	5852.9	16843.2	5332.8	5336.0	3085.6	7521.6
S100	1000	14017.8	14017.0	7619.8	20304.6	6178.5	6160.0	3824.0	8480.8
S200	1000	59602.8	60321.5	36976.8	80033.2	17044.9	17140.0	12061.4	21662.1
S300	1000	136097.0	137972.5	90976.8	177193.6	31607.3	31892.0	23001.8	39012.4
S400	1000	244864.3	248596.0	173835.2	307019.8	46661.3	46880.0	37165.4	54896.6
S500	1000	378996.3	381381.5	274100.7	472220.2	61405.3	61654.0	49173.8	72612.8
S600	1000	546143.5	549124.5	397475.9	670184.7	84767.9	85786.0	67468.6	99364.5
S700	1000	737304.2	742278.5	565928.8	892021.2	103445.1	103565.0	86707.3	119857.8
S800	999	945159.5	954683.0	704833.1	1149060.4	121385.7	121794.0	101562.0	139396.8
S900	1000	1194184.0	1202831.0	935470.5	1432438.2	141469.8	141310.0	121259.1	160431.9
S1000	999	1472310.9	1488288.0	1126992.0	1749723.5	161565.1	162486.0	136646.2	181580.0
S2000	5	5634355.8	5817101.0	4822710.8	6399964.2	417891.2	414424.0	399376.0	443734.4
S3000	5	11823955.4	11868313.0	11037121.2	12496744.2	738962.0	735218.0	696282.8	776654.0
S4000	5	19613652.4	18888936.0	18401424.6	21468356.2	1029854.0	1029180.0	980168.8	1071605.2
S5000	3	30767743.3	30472468.0	30221465.2	31520714.2	1434044.0	1430948.0	1424421.2	1445834.0

Table B.5: Comparison of the minimum number of bits required to store edge capacities of a network (exponential weight edge capacities) and vb-network (vector-based weight edge capacities). In this table, 5th and 95th refer to the 5th and 95th percentiles respectively, and N_I denotes the number of instances that did not timeout and had at least one rotation, and were thus used in space requirement calculations.

Appendix C

Large stable matchings in SPA-ST – supplementary material

C.1 Further discussion on Király's $\frac{3}{2}$-approximation algorithm for SMTI

Let I be an instance of SPA-ST. In Section 5.3 we showed that converting I to an instance of SMTI and then using Király's approximation algorithm does not necessarily result in a matching M in I that is a $\frac{3}{2}$-approximation to a maximum stable matching, even in the case where M is stable. In this section we give some intuition as to why this is the case, by comparing the effect of conversion to an SMTI instance and then use of Király's approximation algorithm, when applied to instance I_1 from Section 5.3 and when applied to a new instance I_2.

Figure C.1a shows the SPA-ST instance I_2 which is the same as the instance in Figure 5.2a but with the capacities of projects p_1 and p_2 reduced to 1. Figure C.1c shows I_2 converted into an SMTI instance I_2'' using the same two-stage process as in Figure 5.2. Finally, Table C.1 shows the algorithm trace for instance I_2'' using Király's $\frac{3}{2}$-approximation algorithm for SMTI [40]. We can see from Figure C.1 that, in contrast to the case for instance I_1, this process yields a stable matching $M = \{(s_1, p_3), (s_2, p_4), (s_4, p_2)\}$ in I_2 that is a $\frac{3}{2}$-approximation to a maximum stable matching (this must be the case since there are only 4 students in the instance).

The main first difference in the traces can be seen on line 14 of Table 5.1 and line 11 of Table C.1. On line 11 of Table C.1, m_2 applies to w_4 as an advantaged man, giving him the ability to take w_4 from m_3. This shows the benefit of having a tie including m_2 and m_3 at the beginning of w_4's list - either of these men being assigned to w_4 would be equally useful in a stable matching. Therefore allowing m_2 to take w_4 from m_3 gives m_3 a chance to get

Students preferences:
s_1: p_3
s_2: $p_4\ p_1\ p_2$
s_3: p_3
s_4: $(p_2\ p_3)\ p_4\ p_1$

Project details:
p_1: lecturer l_1, $c_1 = 1$
p_2: lecturer l_1, $c_2 = 1$
p_3: lecturer l_2, $c_3 = 2$
p_4: lecturer l_2, $c_4 = 1$

Lecturer preferences:
l_1: $s_2\ s_4$ $d_1 = 2$
l_2: $s_4\ (s_1\ s_2\ s_3)$ $d_2 = 2$

(a) SPA-ST instance I_2. Same as instance I in Figure 5.2a except that projects 1 and 2 have an capacity of 1.

Resident preferences:
r_1: h_3
r_2: $h_4\ h_1\ h_2$
r_3: h_3
r_4: $(h_2\ h_3)\ h_4\ h_1$
r_5: $(h_3\ h_4)$

Hospital preferences:
h_1: $r_2\ r_4$ $c'_1 = 1$
h_2: $r_2\ r_4$ $c'_2 = 1$
h_3: $r_5\ r_4\ (r_1\ r_3)$ $c'_3 = 2$
h_4: $r_5\ r_4\ r_2$ $c'_4 = 1$

(b) HRT instance I'_2 converted from the SPA-ST instance in Figure C.1a.

Women's preferences:
w_1: $(m_3\ m_5)$
w_2: $m_4\ m_1\ m_2$
w_3: $(m_3\ m_5)$
w_4: $(m_2\ m_3\ m_5)\ m_4\ m_1$
w_5: $(m_3\ m_5\ m_4)$

Men's preferences:
m_1: $w_2\ w_4$
m_2: $w_2\ w_4$
m_3: $w_5\ w_4\ (w_1\ w_3)$
m_4: $w_5\ w_4\ w_2$
m_5: $w_5\ w_4\ (w_1\ w_3)$

(c) SMTI instance I''_2 converted from the HRT instance in Figure C.1b.

Figure C.1: Conversion of an SPA-ST instance to an SMTI instance.

another partner, increasing the size of matching eventually attained. On line 14 of Table 5.1, m_2 becomes 'stuck' on w_5, one of the women derived from a dummy resident. This stops m_2 being able to ever apply to w_4 as an advantaged man and the benefits of having m_2 and m_3 tied at the beginning of w_4's preference list are not realised.

	Action	m_1	m_2	m_3	m_4	m_5
1	m_5 applies to w_5, accepted					w_5
2	m_4 applies to w_5, rejected, m_4 removes w_5					w_5
3	m_4 applies to w_4, accepted				w_4	w_5
4	m_3 applies to w_5, rejected, m_3 removes w_5				w_4	w_5
5	m_3 applies to w_4, accepted, m_4 removes w_4			w_4		w_5
6	m_4 applies to w_2, accepted			w_4	w_2	w_5
7	m_2 applies to w_2, rejected, m_2 removes w_2			w_4	w_2	w_5
8	m_2 applies to w_4, rejected, m_2 removes w_4			w_4	w_2	w_5
9	m_2 advantaged			w_4	w_2	w_5
10	m_2 applies to w_2, rejected, m_2 removes w_2			w_4	w_2	w_5
11	m_2 applies to w_4, accepted, m_3 removes w_4		w_4		w_2	w_5
12	m_3 applies w_1, accepted		w_4	w_1	w_2	w_5
13	m_1 applies to w_2, rejected, m_1 removes w_2		w_4	w_1	w_2	w_5
14	m_1 applies to w_4, rejected, m_1 removes w_4		w_4	w_1	w_2	w_5
15	m_1 advantaged		w_4	w_1	w_2	w_5
16	m_1 applies to w_2, rejected, m_1 removes w_2		w_4	w_1	w_2	w_5
17	m_1 applies to w_4, rejected, m_1 removes w_4		w_4	w_1	w_2	w_5
18	m_1 inactive	$-$	w_4	w_1	w_2	w_5

Table C.1: Trace of running Király's SMTI $\frac{3}{2}$-approximation algorithm for instance I_2'' in Figure 5.2c. In this table, the phrase "m_i removes w_j" indicates that man m_i removes woman w_j from their preference list.

Appendix D

Experiments and further results for SPA-ST – supplementary material

D.1 Experimental work supplement

In this Section we present tables of results for the experiments conducted in Section 6.2.3.

	Approx			Minimum			Maximum		
Case	Median	5th	95th	Median	5th	95th	Median	5th	95th
SIZE1	97.0	93.0	99.0	92.0	89.0	95.0	98.0	95.0	100.0
SIZE2	193.0	188.0	197.0	183.0	179.0	188.0	196.0	191.0	199.0
SIZE3	289.0	283.0	294.0	275.0	269.0	280.0	294.0	288.0	298.0
SIZE4	385.0	379.0	391.0	366.0	360.0	373.0	392.0	386.0	397.0
SIZE5	481.0	474.0	488.0	458.0	450.0	465.0	490.0	483.0	496.0
SIZE6	577.0	570.0	585.0	549.0	541.0	557.0	588.0	580.0	594.0
SIZE7	673.0	665.0	681.0	641.0	632.0	649.0	686.0	678.0	693.0
SIZE8	770.0	761.0	778.0	732.0	723.0	741.0	784.0	775.0	791.0
SIZE9	866.0	856.0	875.0	823.0	813.0	833.0	882.0	873.0	890.0
SIZE10	962.0	952.0	971.0	915.0	904.0	925.0	980.0	970.0	988.0

Table D.1: Comparison of the size of the stable matching returned by the approximation algorithm, and the minimum and maximum stable matching sizes, with increasing instance size.

	Approx			Minimum			Maximum		
Case	Median	5th	95th	Median	5th	95th	Median	5th	95th
TIES1	284.0	278.0	290.0	284.0	278.0	290.0	284.0	278.0	290.0
TIES2	285.0	279.0	290.0	282.0	276.0	288.0	286.0	280.0	291.0
TIES3	286.0	280.0	291.0	280.0	274.0	286.0	288.0	282.0	293.0
TIES4	287.0	281.0	292.0	278.0	272.0	283.0	290.0	284.0	295.0
TIES5	288.0	282.0	293.0	275.0	269.0	281.0	292.0	287.0	297.0
TIES6	289.0	284.0	294.0	272.0	266.0	278.0	294.0	289.0	298.0
TIES7	290.0	285.0	295.0	269.0	263.0	275.0	296.0	291.0	299.0
TIES8	292.0	286.0	296.0	266.0	260.0	272.0	298.0	294.0	300.0
TIES9	293.0	288.0	297.0	263.0	257.0	269.0	299.0	296.0	300.0
TIES10	294.0	289.0	298.0	259.0	253.0	265.0	299.0	297.0	300.0
TIES11	295.0	291.0	298.0	255.0	249.0	261.0	300.0	298.0	300.0

Table D.2: Comparison of the size of the stable matching returned by the approximation algorithm, and the minimum and maximum stable matching sizes, with increasing probability of ties.

	Approx			Minimum			Maximum		
Case	Median	5th	95th	Median	5th	95th	Median	5th	95th
PREF1	215.0	205.0	225.0	215.0	205.0	225.0	215.0	205.0	225.0
PREF2	262.0	254.0	270.0	249.0	241.0	257.0	264.0	256.0	272.0
PREF3	281.0	274.0	287.0	267.0	260.0	273.0	285.0	278.0	291.0
PREF4	290.0	285.0	295.0	277.0	271.0	283.0	294.0	289.0	298.0
PREF5	295.0	291.0	298.0	284.0	279.0	289.0	298.0	295.0	300.0
PREF6	298.0	294.0	300.0	289.0	284.0	293.0	299.0	297.0	300.0
PREF7	299.0	296.0	300.0	292.0	288.0	296.0	300.0	299.0	300.0
PREF8	300.0	298.0	300.0	295.0	291.0	297.0	300.0	299.0	300.0
PREF9	300.0	299.0	300.0	296.0	293.0	299.0	300.0	299.0	300.0
PREF10	300.0	299.0	300.0	297.0	295.0	299.0	300.0	300.0	300.0

Table D.3: Comparison of the size of the stable matching returned by the approximation algorithm, and the minimum and maximum stable matching sizes, with increasing student preference list length.

Case	Minimum A/Max	% A=Max	% A≥ 0.98Max	Mean size A	Min	Max	A/Max	Min/Max	Mean time (ms) A	Min	Max
SIZE1	0.9286	17.8	62.7	96.4	92.0	97.8	0.9859	0.9408	43.3	147.6	137.8
SIZE2	0.9585	1.6	62.6	192.6	183.4	195.7	0.9840	0.9373	51.2	230.6	210.6
SIZE3	0.9556	0.1	63.7	288.7	274.9	293.7	0.9831	0.9361	56.6	346.4	313.4
SIZE4	0.9644	0.0	65.6	384.9	366.4	391.7	0.9827	0.9354	59.7	488.7	429.3
SIZE5	0.9654	0.0	66.5	481.0	457.7	489.6	0.9824	0.9349	62.8	660.3	555.6
SIZE6	0.9641	0.0	66.8	577.2	549.3	587.7	0.9821	0.9346	66.4	862.3	713.0
SIZE7	0.9679	0.0	65.4	673.3	640.5	685.7	0.9819	0.9341	69.8	1127.8	900.6
SIZE8	0.9684	0.0	67.4	769.5	732.0	783.8	0.9818	0.9340	73.0	1437.3	1098.2
SIZE9	0.9653	0.0	68.6	865.6	823.4	881.7	0.9817	0.9339	76.5	1784.3	1343.9
SIZE10	0.9701	0.0	68.0	961.7	914.7	979.7	0.9816	0.9337	86.6	2281.2	1651.0

Table D.4: Comparisons of the matching output by the approximation algorithm, and IP model implementation outputs, with increasing instance size.

Case	Minimum A/Max	% A=Max	% A≥ 0.98Max	Mean size A	Min	Max	A/Max	Min/Max	Mean time (ms) A	Min	Max
TIES1	1.0000	100.0	100.0	284.0	284.0	284.0	1.0000	1.0000	59.2	184.0	186.9
TIES2	0.9792	38.0	100.0	284.9	282.0	285.8	0.9968	0.9866	61.2	192.4	194.7
TIES3	0.9722	12.1	99.3	285.9	279.9	287.9	0.9933	0.9722	61.7	201.0	203.1
TIES4	0.9655	3.4	95.2	287.0	277.6	289.9	0.9900	0.9576	62.3	213.3	214.5
TIES5	0.9626	1.0	82.5	288.0	275.1	291.9	0.9865	0.9423	62.9	234.3	231.0
TIES6	0.9558	0.4	66.7	289.2	272.4	294.0	0.9837	0.9266	64.2	274.2	260.6
TIES7	0.9486	0.2	52.9	290.3	269.4	295.7	0.9816	0.9111	64.3	358.3	311.3
TIES8	0.9527	0.2	46.4	291.4	266.2	297.2	0.9803	0.8957	64.2	577.3	380.7
TIES9	0.9467	0.2	50.4	292.5	262.7	298.3	0.9805	0.8804	65.2	1234.1	427.5
TIES10	0.9529	0.5	61.9	293.7	258.9	299.1	0.9821	0.8656	59.6	2903.4	409.1
TIES11	0.9467	1.0	74.2	294.8	254.8	299.5	0.9842	0.8506	60.4	5756.9	377.4

Table D.5: Comparisons of the matching output by the approximation algorithm, and IP model implementation outputs, with increasing probability of ties.

Case	Minimum A/Max	% A=Max	% A≥ 0.98Max	Mean size A	Min	Max	A/Max	Min/Max	Mean time (ms) A	Min	Max
PREF1	1.0000	100.0	100.0	215.0	215.0	215.0	1.0000	1.0000	74.3	107.5	105.1
PREF2	0.9699	12.3	99.0	262.1	249.1	264.1	0.9926	0.9433	67.5	133.8	128.7
PREF3	0.9617	1.2	84.0	280.9	266.4	284.7	0.9867	0.9361	68.1	181.4	174.0
PREF4	0.9623	1.0	82.8	290.0	277.0	293.9	0.9866	0.9426	69.1	249.7	242.6
PREF5	0.9661	4.2	95.1	294.8	283.9	297.7	0.9902	0.9537	68.3	346.7	340.3
PREF6	0.9732	15.7	99.5	297.3	288.7	299.1	0.9940	0.9653	66.1	472.4	440.6
PREF7	0.9767	36.2	100.0	298.7	292.1	299.7	0.9966	0.9746	64.5	638.3	550.9
PREF8	0.9833	58.2	100.0	299.3	294.4	299.9	0.9982	0.9819	64.1	811.9	660.3
PREF9	0.9866	75.5	100.0	299.7	296.1	299.9	0.9991	0.9873	63.4	1032.2	789.1
PREF10	0.9900	87.3	100.0	299.8	297.4	300.0	0.9995	0.9913	104.3	1239.4	931.0

Table D.6: Comparisons of the matching output by the approximation algorithm, and IP model implementation outputs, with increasing student preference list length.

Case	Instances completed A	Min	Max	Mean time (ms) A	Min	Max
SCALS1	10	10	10	1393.8	127980.3	227764.3
SCALS2	10	10	9	5356.7	353272.3	1166441.0
SCALS3	10	10	0	13095.3	785421.2	1800000.0
SCALS4	10	7	0	18883.5	1283453.5	1800000.0
SCALS5	10	7	0	20993.0	1455410.1	1800000.0
SCALP1	10	0	9	193.3	1800000.0	264818.6
SCALP2	10	1	10	189.4	1762884.4	631225.2
SCALP3	10	0	3	196.6	1800000.0	1524675.3
SCALP4	10	0	1	248.5	1800000.0	1779420.1
SCALP5	10	0	0	283.7	1800000.0	1800000.0
SCALP6	10	0	0	288.4	1800000.0	1800000.0

Table D.7: Scalability experiment results.